FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Integrated HEX-Algorithms

and Applications in Machine Learning

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Tobias Kaminski, MSc.
Matrikelnummer 01528618

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Zweitbetreuung: Privatdoz. Dipl.-Ing. Dr.techn. Nysret Musliu

Diese Dissertation haben begutachtet:

Assoc. Prof. Dr. Joohyung Lee

Prof. Dr. Torsten Schaub

Wien, 21. Oktober 2020

Tobias Kaminski, MSc.

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Integrated HEX-Algorithms

and Applications in Machine Learning

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Tobias Kaminski, MSc.
Registration Number 01528618

to the Faculty of Informatics
at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Second advisor: Privatdoz. Dipl.-Ing. Dr.techn. Nysret Musliu

The dissertation has been reviewed by:

Assoc. Prof. Dr. Joohyung Lee	Prof. Dr. Torsten Schaub

Vienna, 21st October, 2020

Tobias Kaminski, MSc.

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Tobias Kaminski, MSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Oktober 2020

Tobias Kaminski, MSc.

Acknowledgements

Most of all, I am deeply grateful for the guidance and support I received from my supervisor Thomas Eiter during the entire course of my doctoral studies. Without his unrelenting help and encouragement, this dissertation would not have been possible. The dedication and attention he constantly shows his doctoral students is exceptional. It was always truly impressive how he instantly grasped the core of the different obstacles I encountered in my research and offered invaluable advice.

I am also indebted to my colleagues and co-authors Christoph Redl and Antonius Weinzierl, who introduced me to our common research project and guided me with their experience throughout my doctorate. Antonius devoted a lot of time to introducing me to our research field and the academic world in general, and Christoph's expertise on HEX-programs was an indispensable asset in our joint work.

In addition, I would like to extend my special thanks to Katsumi Inoue for hosting me twice at the National Institute of Informatics in Tokyo and for a very fruitful and gratifying collaboration. I also sincerely appreciated the backing from my co-advisor Nysret Musliu, and the assistance and advice I received from Peter Schüller. Moreover, I thank the examination committee, and in particular the international examiners, for their efforts in reviewing my dissertation.

I was extremely fortunate being able to conduct my doctoral studies in the context of the LogiCS doctoral college, which offers exceptional academic as well as social support to students, and enabled me to present my work at a number of international conferences. I am very thankful to the Austrian Science Fund for making this possible and for funding my doctoral research. Many thanks to the many people who are essential for the smooth operation of the doctoral college and the institute, in particular Anna Prianichnikova, Eva Nedoma, Beatrix Buhl and Juliane Auerböck.

It was a unique experience and a great pleasure to work in a large international group of kind people, which I am very grateful for. Especially, I thank my friends and doctoral colleagues Adrian, Adrián, Anna, Jan, Marijana, Matthias and Zeynep, for countless cheerful discussions, lunches, coffee breaks and conference trips together.

Warmest thanks go to my friends Alina, Ilina, Medina, Serge, Sopo and all other Angels. You made me feel at home in Vienna and you made the journey truly joyful.

Finally, I am extremely grateful to my parents and my sister, who are always there for me and support me in any new endeavour I embark on.

Abstract

Due to current trends in distributed systems and information integration, there is an increasing need for accessing external information sources from within knowledge representation formalisms such as *answer set programming* (*ASP*). For instance, it might be necessary to integrate information derived from a (possibly remote) *description logics* (*DL*)-ontology into the computation of an answer set. If the derivation in the ontology is relative to information in the ASP-part, a bidirectional exchange between a DL-reasoner and an ASP-solver is required. This kind of interaction is not provided by ordinary ASP, and pre-computing all possible derivations from the ontology and adding them to the answer set program is often not feasible. Motivated by this, the HEX-formalism has been developed, where external sources can be referenced in a program, and are evaluated during solving. The approach is related to *SAT modulo theories* (*SMT*), but the focus is more on techniques for evaluating general external sources represented by arbitrary computations, i.e. it enables an API-like approach such that a user can define plugins without expert knowledge on solver construction.

HEX-programs are very expressive since the bidirectional exchange of information between a logic program and external sources encompasses the formalization of non-monotonic and recursive aggregates. Consequently, HEX is suited for a wide range of applications, but also requires sophisticated evaluation algorithms to deal with the complexity that goes along with the high expressiveness. For this reason, this thesis work aims at the design and implementation of novel integrated evaluation techniques with the overall goal to improve the efficiency of the formalism in general, as well as for specific classes of programs. Challenges regarding efficient evaluation of HEX-programs comprise the lack of a tight integration of the solving process with the evaluation of external sources and with the grounding procedure. Accordingly, the main focus of this thesis is the design of advanced reasoning techniques that improve the evaluation of HEX-programs by tightly integrating processes which have so far been treated as mostly independent sub-problems. Moreover, all newly developed HEX-algorithms have been implemented in the state-of-the-art HEX-solver, and their performance has been empirically evaluated using a rich benchmark suite.

Another focus of this thesis is on new applications, which leverage the expressiveness of the HEX-formalism and capabilities of its solvers, and in turn, push the advancement of the formalism. In this context, two innovative applications of HEX-programs that utilize external atoms for integrating as well as realizing methods from the area of *machine learning* are developed.

Kurzfassung

Aufgrund aktueller Trends in Bezug auf verteilte Systeme und Informationsintegration besteht ein zunehmender Bedarf an Zugriff auf externe Informationsquellen innerhalb von Wissensrepräsentationsformalismen wie *Antwortmengenprogrammierung* (*ASP*). Beispielsweise kann es erforderlich sein, Informationen, die aus einer (möglicherweise entfernten) *Description Logics* (*DL*)-Ontologie stammen, in die Berechnung einer Antwortmenge zu integrieren. Wenn die Ableitung in der Ontologie von Informationen im ASP-Teil abhängt, ist ein bidirektionaler Austausch zwischen einem DL-Reasoner und einem ASP-Solver erforderlich. Diese Art der Interaktion wird von gewöhnlichem ASP nicht bereitgestellt, und es besteht oft nicht die Möglichkeit, alle potenziellen Ableitungen aus der Ontologie vorab zu berechnen und sie dem Antwortmengenprogramm hinzuzufügen. Aus diesem Grund wurde der HEX-Formalismus entwickelt, bei dem externe Quellen in einem Programm referenziert und bei der Berechnung einer Antwortmenge ausgewertet werden können. Der Ansatz ist mit *SAT-Modulo-Theories* (*SMT*) verwandt, der Schwerpunkt liegt jedoch eher auf Techniken zur Auswertung allgemeiner externer Quellen, die beliebige Berechnungen ausführen können, d.h. er ermöglicht einen API-ähnlichen Zugang zu externen Berechnungsquellen, so dass ein Benutzer Plugins erstellen kann ohne über Expertenwissen im Bereich der Solver-Konstruktion zu verfügen.

HEX-Programme sind sehr ausdrucksstark, da der bidirektionale Informationsaustausch zwischen einem Logikprogramm und externen Quellen die Formalisierung nichtmonotoner und rekursiver Aggregate umfasst. Folglich ist HEX für eine Vielzahl von Anwendungen geeignet, erfordert jedoch auch ausgefeilte Auswertungsalgorithmen, um die Komplexität zu bewältigen, die mit der hohen Ausdrucksstärke einhergeht. Aus diesem Grund zielt diese Doktorarbeit auf die Konzeption und Implementierung neuartiger integrierter Berechnungstechniken mit dem Ziel der Verbesserung der Effizienz des HEX-Formalismus im Allgemeinen sowie für bestimmte Klassen von Programmen. Zu den Herausforderungen bei der effizienten Auswertung von HEX-Programmen gehört das Fehlen einer engen Integration des Berechnungsprozesses mit der Auswertung externer Quellen sowie mit dem Grundierungsverfahren. Dementsprechend liegt der Schwerpunkt dieser Arbeit auf dem Entwurf fortschrittlicher Berechnungstechniken, die die Auswertung von HEX-Programmen verbessern, indem Prozesse eng integriert werden, die bisher als größtenteils unabhängige Unterprobleme behandelt wurden. Darüber hinaus wurden alle neu entwickelten HEX-Algorithmen in den HEX-Solver DLVHEX integriert und ihre Leistung mithilfe von umfangreichen Experimenten empirisch ausgewertet.

Ein weiterer Schwerpunkt dieser Arbeit liegt auf neuen Anwendungen, die die Ausdruckskraft des HEX-Formalismus und die Fähigkeiten seiner Berechnungssysteme nutzen und wiederum die Weiterentwicklung des Formalismus vorantreiben. In diesem Zusammenhang werden zwei innovative Anwendungen von HEX-Programmen entwickelt, welche externe Atome verwenden um Methoden aus dem Bereich *maschinelles Lernen* zu integrieren bzw. zu realisieren.

Contents

CHAPTER 1

Introduction

Answer set programming (*ASP*) is a well-known declarative programming approach based on the *stable-model semantics* (Gelfond & Lifschitz, 1991). Thanks to efficient and expressive systems like CLASP (Gebser, Kaufmann, et al., 2011), SMODELS (Simons et al., 2002), DLV (Leone et al., 2006), and WASP (Alviano, Dodaro, et al., 2015), it has been successfully applied to a wide range of applications in *artificial intelligence* and beyond (Brewka et al., 2011; Erdem et al., 2016). In a nutshell, a problem at hand is represented by a set of rules (an ASP-program) such that its models, called *answer sets*, encode the solutions to the problem; an answer set solver is used to compute models, from which the solutions are then extracted. The approach is a relative of SAT-solving, but in contrast, starts from a relational language where variables range over a (finite) set of constants, which allows for more compact formalization than in propositional logic. Furthermore, the support of *negation as failure* makes ASP inherently nonmonotonic, which allows one for instance to easily express transitive closure. Finally, a number of language extensions that include, among others, optimization constructs, aggregates, disjunctions, and choice rules (cf. Gebser & Schaub, 2016) have turned ASP into a very expressive and powerful problem solving tool.

HEX-programs (Eiter et al., 2008; Eiter, Kaminski, Redl, et al., 2017) are an extension of ASP-programs aimed at the integration of heterogeneous external information sources, such as XML/RDF data bases, SAT-solvers, route planners etc. So-called *external atoms* can be used in rules and provide a bidirectional interface between the logic program and the external sources in an API-style manner. To this end, an external atom states an input-output relationship; it passes information from the program, given by predicate extensions and constants, to an external source, which returns the output values for the respective input. The external atom then evaluates to either true or false for each output value. For example, an external atom $\&synonym[car](X)$ might find synonyms X of *car*, e.g. *automobile*, *bus*, *motorcar* etc., by accessing a thesaurus such as the one of Merriam-Webster (Merriam-Webster Website, 2018); that is, e.g. $\&synonym[car](automobile)$ evaluates to true. As seen from this example, external

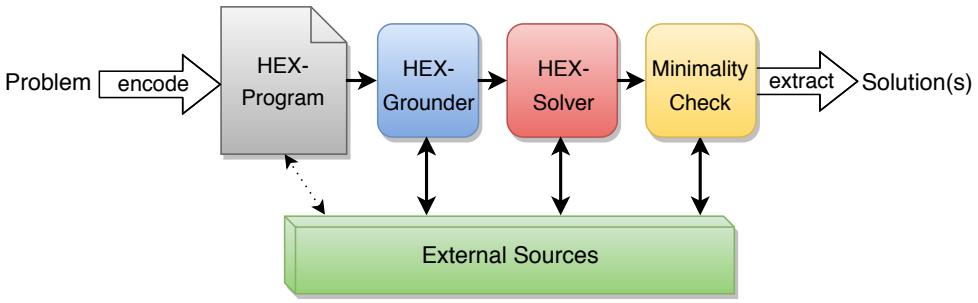

Figure 1.1: Traditional evaluation of HEX-programs

sources can be of non-logical nature, and without particular assumption about how the external source is evaluated. This is facilitated by an abstract modeling of external atoms that can exhibit nonmonotonic behavior, be used in recursive and cyclic definitions, and introduce new constants which do not appear in the original program (known as *value invention*). This rich expressiveness empowers HEX-programs to subsume many other ASP-extensions such as programs with (nonmonotonic) aggregates (Alviano, Faber, & Gebser, 2015), constraint ASP (Ostrowski & Schaub, 2012), and DL-programs (Eiter, Lukasiewicz, Schindlauer, & Tompits, 2004), to mention a few; furthermore, the versatility and genericity of external atoms has been exploited for different purposes and application domains (cf. Erdem, Gelfond, and Leone (2016) and Eiter, Kaminski, Redl, et al. (2017)).

1.1 Motivation

Due to the complexity that goes along with the high expressiveness of HEX-programs, advanced reasoning algorithms are required for HEX-evaluation. These algorithms need to take external sources into account during all phases of the solving process. HEX-evaluation is, like ordinary ASP-solving, performed in different phases, where besides the common separation into a *grounding phase*, i.e. the instantiation of variables in a program, and a *solving phase*, i.e. the search for answer sets, additional procedures are required to cater for the integration of external sources. These comprise the evaluation of external sources and a special minimality check which is necessary due to the presence of external atoms. The traditional approach of HEX-evaluation is illustrated in Figure 1.1, where a problem is encoded by means of a HEX-program, which is sequentially processed by the grounder, the solver, and a module that eliminates non-minimal solutions. The solutions of the original problem can then be extracted from the resulting answer sets. At this, diverse external sources can be referenced in the HEX-program, and need to be queried at each evaluation stage.

In recent years, many sophisticated methods have been developed that aim at different sub-processes of HEX-evaluation, e.g. elaborate approaches for grounding (Eiter, Fink, Krennwallner, & Redl, 2016) and modular program decomposition (Eiter, Fink, Ianni, et al., 2016), as well as the integration of modern solving techniques (Eiter et al., 2012). However, these techniques focus mainly on specific sub-tasks that have, until now, only

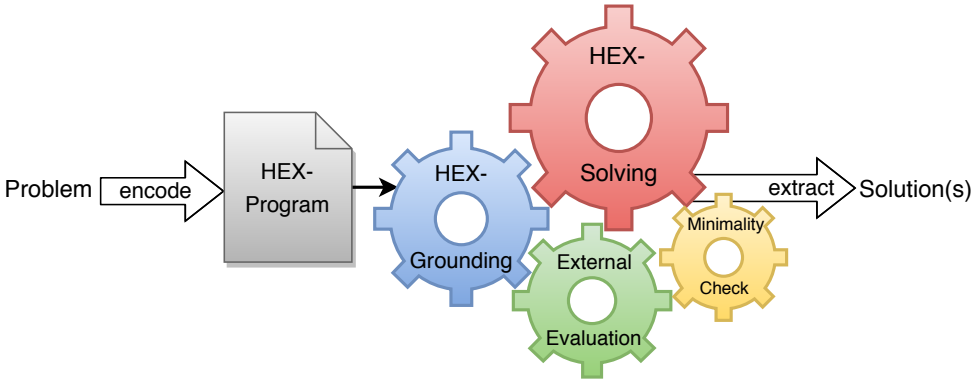

Figure 1.2: Integrated evaluation of HEX-programs

been remotely interleaved. Accordingly, previous HEX-algorithms lack a global view of the overall problem and do not achieve an optimal coupling of sub-processes, which so far have been mostly considered independently. In practice, this leads to unsatisfactory performance for different classes of programs and concrete use cases. Moreover, previous approaches considered external atoms to be largely *black boxes*, such that new interfaces to external sources that realize a *clear box* approach are essential to accomplish a tight coupling of the different evaluation processes. For the latter, it is e.g. necessary to query external sources with input information that is only partially available and to dispose of approximations of the external source semantics that are as precise as possible. At the same time, representing the exact semantics of external sources directly in the solver would defy the goal of computation outsourcing.

Accordingly, the main motivation for our work is the fact that there is still much room for unleashing untapped potential of the HEX-formalism by devising integrated HEX-algorithms. Figure 1.2 depicts the envisaged picture of HEX-evaluation, where all sub-processes are tightly coupled and their execution is interleaved with the evaluation of external sources; this way, external computations can effectively steer evaluation. Considerable improvements w.r.t. the efficiency of state-of-the-art HEX-solvers are expected by overcoming previous limitations emerging from the separation of evaluation processes. This is particularly important as the development of ASP has already shown that the ASP-paradigm became popular in practice only with the emergence of efficient solvers. Consequently, efficient algorithms and implementations are crucial for the success of formalisms such as HEX.

A further motivation for improving the core mechanisms of HEX-solvers is that newly developed techniques employed internally can to a large extent be hidden from users; and the simple yet flexible syntax of HEX can be maintained. As a result, performance improvements can directly be leveraged by users without expert knowledge on solver construction or advanced encoding techniques. This is in line with the goal of HEX to realize an API-like approach and to provide convenient means for incorporating general external sources, such that a user can easily define new plugins.

Besides efficient algorithms, a powerful formalism requires innovative applications that

leverage the provided techniques, and in turn push its advancement. For this reason, we additionally identify novel application areas for the HEX-formalism, which can drive the development of new evaluation methods. Even though HEX-programs are well-suited for combining diverse forms of reasoning, and many HEX-applications have been developed in the past (cf. Section 1.2.3), mostly use cases from the area of *knowledge representation and reasoning* (*KRR*) have been considered. Hence, there is a lack of applications that fully leverage the flexibility of HEX to combine KRR-related use cases with formalisms from other areas such as *machine learning*. In this thesis, we investigate such new types of applications since new limits of the HEX-formalism that can be addressed by future research are likely to be recognized during their design.

Finally, many techniques which are developed in the context of HEX are also relevant for related areas such as *SAT modulo theories* and ASP-solving and therefore, new techniques developed for HEX are potentially useful for these approaches as well.

1.2 State of the Art

In this section, which contains parts from (Eiter, Kaminski, Redl, Schüller, & Weinzierl, 2017), we discuss the state of the art regarding formalisms for integrating external sources into declarative problem solving, program evaluation inside the HEX-formalism and HEX-applications.

1.2.1 External Sources in Declarative Problem Solving

Because there are many scenarios where it is more natural, and often more efficient, to outsource some information or computation in the context of declarative problem solving, a number of approaches have been developed for this purpose, realizing different degrees of integration.

Motivated by the need for the integration of data in commercial relational databases, extensions of DLV have been developed that allow to access external data. The dlv^{DB} system (Terracina, Leone, Lio, & Panetta, 2008; Terracina, Francesco, Panetta, & Leone, 2008) offers via an ODBC-interface access to dispersed relational databases, where both direct (remote) execution of possibly recursive queries on databases and main memory execution (after loading the databases) are supported. The ontodlv system (Ricca et al., 2009), allows the user to retrieve information from OWL-ontologies, which can be utilized in a genuine ontology representation language that extends ASP with features such as classes, inheritance, relations and axioms.

DLV-EX programs (Calimeri, Cozza, & Ianni, 2007) represent an early generic integration approach, which enables bidirectional communication with an external source, and allows the introduction of new terms by value invention into an answer set program. However, the interaction is more restricted than in the case of HEX since only terms can be used as inputs to external sources and thus, e.g., recursive aggregates cannot be expressed in this formalism.

The CLINGO-system also provides a mechanism for importing the extension of user-defined predicates (Gebser, Kaminski, Kaufmann, & Schaub, 2014) via special atoms similar to DLV-EX, but they are different from external atoms in HEX in that their evaluation is not interleaved with the solving process. For this, GRINGO supports custom functions (implemented in the scripting languages Lua or Python) which are evaluated during the program grounding and thus compiled away prior to the solving step. They are intended to be used as customizable built-in atoms, but no cyclic dependencies are possible.

Recently, CLINGO 5 has been released (Gebser et al., 2016), which provides rich generic interfaces for integrating theory solving into ASP. A main difference between *ASP modulo theories* solving in CLINGO 5 and the HEX-framework consists in the fact that unfounded support over theory atoms is allowed by the semantics defined for CLINGO, which would violate the minimality criterion of HEX. Consequently, a more sophisticated minimality check has to be applied during the evaluation of HEX-programs, lifting the computational complexity of the formalism.[1]

Moreover, even though the CLINGO-system moves into a similar direction as HEX by facilitating the integration of external reasoners, the perspectives taken by the two systems are different, and their roles can be viewed as somewhat orthogonal.

While theory atoms are interrelated via an external theory in CLINGO, where the consistency of their truth assignments is usually checked during theory propagation, the truth values of external atoms in HEX depend on the evaluation of ordinary atoms representing their input. Thus, the focus of the HEX-approach is more on input-output relations over external atoms, which are easy to understand from a user's perspective and can be used to call external sources in an API-like fashion.

As a result, external atoms have a number of distinguishing features, which are tailored to their specific role in the HEX-framework. For instance, the possibility to declare additional properties of external source that can be exploited for solving (cf. Redl, 2016) constitutes a user-friendly high-level interface for steering the external evaluation process, which has to be implemented manually for each theory in CLINGO's propagation methods.

The input-output structure of external atoms facilitates the introduction of constants by value invention relying on the *liberal safety* condition for HEX-programs (Eiter, Fink, Krennwallner, & Redl, 2016), which is of special interest for applications in the area of the *semantic web*. There is no comparable mechanism for value invention in CLINGO 5 as new values cannot be imported based on a respective answer set, and theory solving is performed w.r.t. the pre-grounded program.

On the other side, CLINGO 5 is well-suited for system development and offers a powerful framework for solver building, by providing a comprehensive and rich infrastructure at the low technical levels for integrating theory reasoning into CLINGO, which is accessible through an interface. This novel interface will be exploited in future versions of the

[1] Deciding the existence of an answer set of a ground HEX program in the presence of nonmonotonic external atoms that are decidable in polynomial time is Σ_2^p-complete already for Horn programs (Eiter, Fink, Krennwallner, Redl, & Schüller, 2014).

main HEX-solver DLVHEX, which benefits a lot from the CLINGO advances, and CLINGO 5 constitutes the foundation of the more recent and lightweight HEX-solver *hexlite* (Schüller, 2019).

Besides CLINGO, the WASP-solver has recently been extended with support for general external Python propagators (Dodaro, Ricca, & Schüller, 2016). Furthermore there are extensions of ASP towards the integration of specific external sources. Examples are constraint ASP as an integration of ASP with constraint programming as realized e.g. in CLINGCON (Ostrowski & Schaub, 2012) LC2CASP (Cabalar, Kaminski, Ostrowski, & Schaub, 2016), EZCSP (Balduccini, 2009), and EZSMT (Susman & Lierler, 2016). The latter is like MINGO (Liu, Janhunen, & Niemelä, 2012) an SMT-based solver for constraint ASP; other formalisms that extend ASP with SMT are DINGO (Janhunen, Liu, & Niemelä, 2011), which uses difference logic, and ASPMT (Lee & Meng, 2013). For an overview of systems that combine ASP with constraint solving and other theories, we refer to (Lierler, Maratea, & Ricca, 2016).

Similar to SMT (Nieuwenhuis, Oliveras, & Tinelli, 2006), where usually only specific theories are considered, the mentioned approaches rely on a tailored integration of an external solver and hence, can easily leverage the propagation capabilities of the respective solver. The aim of the HEX-formalism differs in that it strives to enable a broad range of users to implement custom external sources and to harness efficient solving techniques for HEX-programs. Moreover, CLINGCON and approaches in SMT usually only consider monotonic external theories, which facilitates the integration of their evaluation into the respective solving algorithm. In contrast, HEX allows for the integration of arbitrary external sources through a general interface and their flexible combination; the other use cases correspond to special cases thereof.

1.2.2 Evaluation Techniques of HEX-Solvers

There are two state-of-the-art systems for evaluating HEX-programs, the DLVHEX-system[2] (Redl, 2016), and the more recent lightweight *hexlite*-solver[3] (Schüller, 2019). Both solvers are available for *Linux*, *macOS* and *Windows*, and external sources are realized by *Python*-plugins, whereby DLVHEX also supports sources implemented in *C++*. While *hexlite* handles a fragment of the HEX-language and delegates as much work as possible to its backend solver CLINGO for reasons of efficiency and simplicity, DLVHEX is fully featured and implements all techniques for HEX-solving that have been developed in the literature, e.g. in (Eiter, Fink, Ianni, et al., 2016; Eiter et al., 2012; Eiter, Fink, Krennwallner, et al., 2014; Eiter, Fink, Krennwallner, & Redl, 2016). Since *hexlite* only implements part of the methods available for HEX-program evaluation, in this section, we focus on the state-of-the-art evaluation techniques that have been incorporated into DLVHEX and discuss how they are integrated into the solver architecture, following Redl (2014).

[2] http://www.kr.tuwien.ac.at/research/systems/dlvhex
[3] https://github.com/hexhex/hexlite

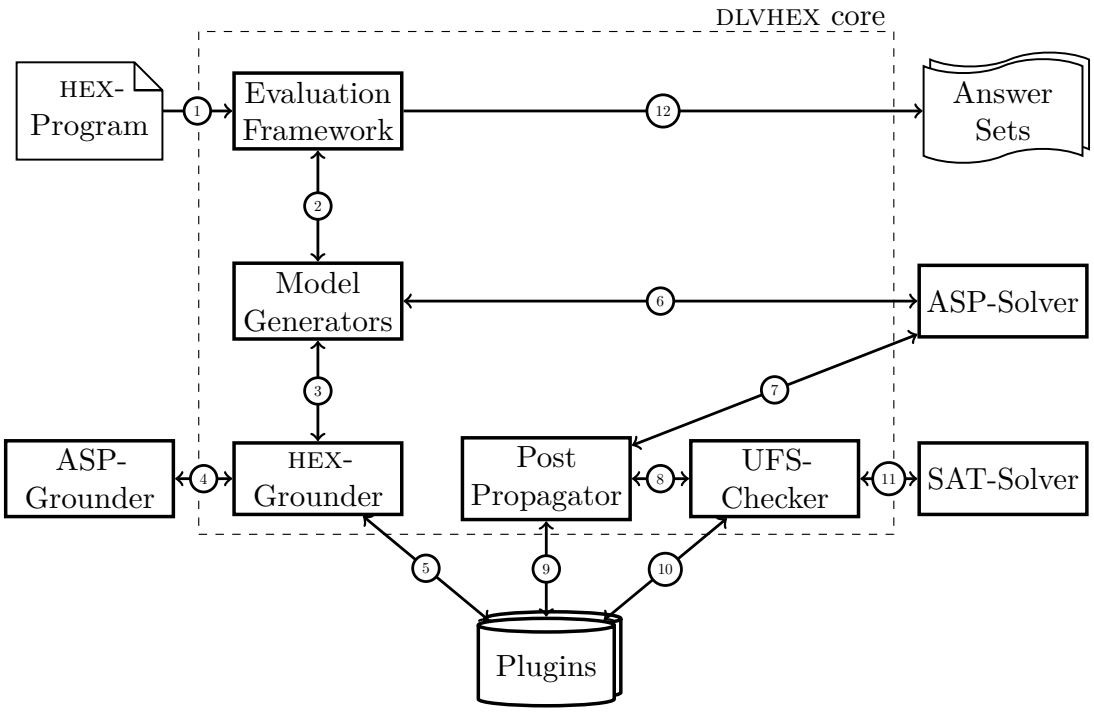

Figure 1.3: Architecture of the DLVHEX-solver (adopted from Redl (2014))

Initially, the DLVHEX-system focused on *semantic web* applications (Eiter et al., 2008). Early versions of DLVHEX were based on DLV (Leone et al., 2006) and extended it with higher-order and external atoms, where the name HEX stems from. Higher-order atoms allow for using variables in place of a predicate symbol, such as in the rule $C(X) \leftarrow subClassOf(D, C), D(X)$, to model a general subclass relation; while they are still supported, they were less emphasized in later versions as they can be compiled away.

In a nutshell, the traditional HEX-algorithm based on DLV translates the HEX-program into an ordinary ASP-program which guesses the values of external atoms (disregarding the actual semantics), evaluates this ASP-program using DLV, and performs for each answer set a post-check to ensure that the guesses are correct. As this approach did not scale to real applications, the evaluation algorithms were improved over time, which required a tighter integration with the backend (such as separate access to the grounding and the solving component of the backend, a callback interface, etc). In context of these improvements, the default backend was replaced by GRINGO and CLASP from the Potassco suite (Gebser, Kaufmann, et al., 2011); the original system name DLVHEX was kept as it stands for *Datalog with disjunctions, higher-order and external atoms.*

The system architecture as presented by Redl (2014) is shown in Figure 1.3, where arcs depict control flow (with the numbers showing the execution stages) as well as data flow. The sub-processes executed during HEX-solving can roughly be divided into four groups, which we discuss in turn now.

Program Decomposition

After the input HEX-program has been read, it is passed to an *evaluation framework* (stage ①), which can decompose the non-ground program into *evaluation units*, using an acyclic evaluation graph based on dependencies between program rules. The main advantage of the program decomposition technique developed by Eiter, Fink, Ianni, et al. (2016) consists in the fact that the resulting smaller program units can be grounded and evaluated independently. This often significantly decreases the size of the grounding, and since components are solved separately, guesses for the evaluation of external atoms can be checked already after computing a model candidate for a single unit. The performance improvements achieved by modular decomposition have, for instance, been shown to be essential for the realization of an application for Multi-Context Systems (Schüller, 2012) in the HEX-formalism.

Subsequent to decomposing the input program, each individual unit is processed by a *model generator* instance (stage ②). The answer sets of leaf components, i.e. units without predecessors, are computed first. The model generators for successor units then receive as *input interpretations* the outputs from predecessor units, which are constituted by the answer sets of the respective sub-programs. All output interpretations generated by the model generators are sent back to the evaluation framework (cf. Figure 1.3), which integrates them into the final answer sets (stage ⑫).

Grounding

When processing an evaluation unit, the model generator forwards the corresponding non-ground HEX-program to the HEX-grounder, from which it receives a ground program back (stage ③). The HEX-grounder utilizes an ordinary ASP-grounder such as GRINGO(stage ④), which is called repeatedly to handle value invention and interfaces the external sources (stage ⑤) in order to retrieve new terms that need to be considered for producing the ground HEX-program. Finally, the ground HEX-program is handed back to the model generator.

Instead of importing the full domain from external sources, i.e. all possible output values of external atoms, the ground HEX-program is enlarged incrementally until all relevant terms have been considered (Eiter, Fink, Krennwallner, & Redl, 2016). In general, this process may not terminate for HEX-programs that do not have a finite grounding, such that some safety notion needs to be imposed that ensures finite groundability. While *strong domain-expansion safety* (Eiter et al., 2006) has been employed traditionally, which does not allow the introduction of new values by external atoms that possibly depend on their own outputs, new safety criteria that combine syntactic and semantic conditions have been introduced by Eiter, Fink, Krennwallner, and Redl (2016). The according criterion of *liberal domain-expansion safety* can be checked efficiently and relaxes the syntactic restrictions that need to be observed when writing a HEX-program. This has, for the first time, enabled applications that use existential quantification to introduce new values (Eiter et al., 2013) or recursively process data structures (e.g. a map for route planning).

Solving

Following the grounding step, the obtained propositional HEX-program is transformed into an ordinary ASP-program by replacing external atoms by fresh ordinary atoms, and by adding guesses for their evaluation. The resulting ordinary ASP-program is then sent to an ordinary ASP-solver (stage ⑥). A crucial step for improving the efficiency of HEX-solving consisted in its integration with *conflict-driven nogood learning* (*CDNL*) (Eiter et al., 2012), as implemented e.g. by CLASP (Gebser, Kaufmann, & Schaub, 2012). For this, the input program is translated into a set of *nogoods*, i.e., a set of literals that must not all be true simultaneously in a solution. Based on this representation, techniques from SAT-solving, such as *unit propagation* and *conflict learning* (Marques-Silva, Lynce, & Malik, 2009), are applied to find an assignment which satisfies all nogoods.

By integrating CDNL-search into the HEX-algorithm, the input-output relations learned from external source evaluations regarding the guesses for external atoms can be learned in form of nogoods to avoid wrong guesses in the future search. Learning of these input-output nogoods significantly reduces the number of model candidates that need to be checked. Moreover, known properties of external sources can be exploited for learning, e.g. more general nogoods can be obtained for sources of which the output *monotonically* depends on the input since information about false input atoms is redundant in this case. Using such properties can have a large effect, e.g. when interfacing a DL-ontology in which reasoning is monotonic (Eiter et al., 2004).

In the DLVHEX-system, this is realized by means of solver callbacks to the *post-propagator* (stage ⑦) w.r.t. complete as well as partial models. The post propagator performs checks to eliminate spurious answer set candidates, which requires calls to the external sources. At this, the truth value assignments of input atoms are provided to the associated plugin (stage ⑨), which returns the truth value of the respective ground external atom (i.e. for a particular output value); nogoods that encode the information gained from these external evaluations are returned to the post-propagator and sent to the solver.

Minimality Checking

In addition, the post-propagator calls the *unfounded set checker* (*UFS-checker*) with the respective (complete or partial) model. The UFS-checker ensures that models satisfy the usual minimality condition of ASP extended to HEX-programs, by ensuring that they do not contain atoms that only circularly support each other (i.e. do not contain a so-called *unfounded set*). At this, the ordinary ASP-solver already performs a minimality check w.r.t. the ordinary ASP-program obtained by replacing external atoms. However, since the semantics of external sources are hidden from the ASP-solver, it cannot ensure minimality of answer sets in all cases, i.e. it cannot detect cyclic justifications involving external atoms.

The search for unfounded sets can be encoded as a SAT-problem and handed to a SAT-solver (stage ⑪), which is significantly more efficient than a direct check and constitutes the state-of-the-art procedure for checking minimality of answer sets in HEX

(Eiter, Fink, Krennwallner, et al., 2014). For this, a SAT-instance w.r.t. a candidate answer set and the according values of external atoms is constructed, such that its models represent unfounded sets. During model search, the SAT-solver needs to access the plugins (stage ⑩) in order to take the external source semantics into account; and nogoods can be returned that are learned from detected unfounded sets (stage ⑧). This way, it can be assured that correct answer sets are returned to the model generators (stage ⑥).

Moreover, a decision criterion has been developed on the basis of cyclic dependencies over external atoms that allows skipping the final minimality check in many cases. Eiter, Fink, Krennwallner, et al. (2014) showed that by constructing a particular dependency graph for a program, the check can also be skipped for subcomponents of a program, allowing a more targeted use of the costly search for unfounded sets.

1.2.3 Applications of HEX-Programs

The HEX-formalism has been applied to a wide range of use cases. Here, we provide an overview over a number of state-of-the-art HEX-applications, covering concrete application scenarios where external atoms are used in a problem encoding, as well as additional language features required by advanced applications which cannot be realized easily in ordinary ASP. HEX-programs can also be used as a backend for realizing formalisms that do not resemble HEX, by using appropriate translations.

One of the early applications of the HEX-formalism consisted in combining *description logics* (*DL*)-ontologies and ASP in the form of so-called *DL-programs*, developed by Eiter et al. (2008). DL-ontologies constitute a logical formalism that is used to model classes of objects and their relations, and enables reasoning tasks such as retrieving all objects that belong to a specific class from a data store while taking class relations into account. DL-ontologies are widely used in the area of the *semantic web* (Heflin & Munoz-Avila, 2002), and they have also been fruitfully employed for medical applications (Hoehndorf et al., 2007). Special *DL-atoms* can be utilized in DL-programs, which are based on external atoms of HEX and enable a bi-directional interaction with a DL-ontology. This way, *default reasoning* can be performed on top of DL-ontologies (Dao-Tran et al., 2009). DL-programs have been used, e.g., for complaint management in an e-government application (Zirtiloglu & Yolum, 2008).

HEX-programs with functions symbols have been devised by Calimeri et al. (2007) to facilitate the usage of uninterpreted functions in HEX. At this, external atoms are used for the composition and decomposition of function terms, exploiting the capability of HEX-programs to introduce new invented values from an external source. This way, function terms can be emulated while, for instance, their nesting depths can be controlled by the external source.

HEX$^{\exists}$-programs (Eiter et al., 2013) also leverage the possibility of value invention in HEX-programs, in this case to realize existential quantification in rule heads, which is not provided by standard ASP. While this is related to the formalism Datalog$^{\pm}$ (Calì, Gottlob, & Pieris, 2012), HEX$^{\exists}$-programs also allow *domain-specific existential quantification* where external atoms can be utilized to control the structure of invented values.

Nested HEX-programs (Redl, Eiter, & Krennwallner, 2011; Eiter, Krennwallner, & Redl, 2011) are able to query other HEX-programs for their answer sets, for which dedicated external atoms have been implemented. As a result, e.g. a library of HEX-programs for problems that are common in ASP such as graph problems can be created, which can be utilized by other HEX-programs.

An application of HEX in the area of *route planning* has been considered by Eiter, Fink, Krennwallner, and Redl (2016), where HEX-programs are utilized to integrate *side constraints* into route planning tasks with multiple stops. An external atom is used to compute the shortest connections between locations. Side constraints constitute additional semantic conditions, e.g. that some pharmacy should be on the route. The more complex task of *pair route planning* has also been encoded by means of a HEX-program, where routes for two persons are computed simultaneously, with the further constraint that the two routes need to intersect at some point (Eiter, Fink, Krennwallner, & Redl, 2016).

Constraint HEX-programs (Rosis, Eiter, Redl, & Ricca, n.d.) integrate *constraint ASP* (*CASP*) (Mellarkod, Gelfond, & Zhang, 2008; Lierler, 2014) and HEX-programs. In contrast to implementing constraints as used in constraint programming (Apt, 2003) directly in an ASP-encoding, the CASP-approach has the advantage that grounding issues due to large constraint domains can be avoided by leveraging a dedicated constraint solver. Unlike other CASP-systems such as CLINGCON (Ostrowski & Schaub, 2012), constraint HEX-programs also allow to combine an external constraint solver with other background theories. At this, constraints are represented by special *constraint atoms* (usually inequalities over arithmetic expressions) in a HEX-program; and are handed via an external atom to an external constraint solver, which checks consistency.

The ACTHEX-framework developed by Basol et al. (2010) allows to execute scheduled actions in an external environment declared by so-called *action atoms* in rule heads. For this, an ACTHEX-program is called repeatedly with evolving sensor information from the environment in which it is executed. By this, ACTHEX-programs are able to connect decisions made in an ASP-program to actual effects in (an abstraction of) the real world outside the program. The framework has e.g. been used to implement the action language $\mathcal{C}$ (Giunchiglia et al., 2004), and to enable the interaction of an ACTHEX-program with an IMAP-server for executing operations on emails in a mailbox.

Finally, the *AngryHEX* agent (Calimeri et al., 2016) is able to play the physics-based computer game *Angry Birds* and participates in the annual *AIBirds Competition*[4]. As the agent is implemented in the HEX-formalism, it is able to combine logic-based reasoning and planning with geometric computations and simulations of action effects, e.g. for simulating the trajectories of shots and the resulting damage to obstacles. Due to the need for applying statistics and physics (e.g. for taking gravity into account) during simulation, ordinary ASP alone is ill-suited for realizing this combination. While planning is preformed by the ASP-part, low-level computations involving floating point numbers can conveniently be outsourced by employing different external atoms of the HEX-formalism for different types of low-level computations.

[4] https://aibirds.org

1.3 Goals of the Research

The overall goal of this thesis is to increase the efficiency of HEX-program evaluation by developing integrated HEX-algorithms in order to promote the practical applicability of HEX. The main focus regarding efficient evaluation in the HEX-formalism concerns interleaving the grounding of programs that interface external sources (which may extend the Herbrand universe of the program by value invention) and the solving process; as well as integrating the search procedures applied during solving and minimality checking with the evaluation of external sources. Moreover, we strive to develop innovative applications of HEX-programs that utilize external atoms for integrating as well as realizing methods from the area of machine learning.

Accordingly, and in more detail, we aim to investigate and answer the following research questions:

(RQ1) How can the search employed during HEX-evaluation be interleaved with the evaluation of external sources, and does a tighter integration of the respective processes lead to more effective search space pruning in practice?

Our goal is to enable external evaluations based on partial input assignments to enable theory propagation, using ideas from SAT modulo theories *(Barrett et al., 2009) and nogood minimization similar to Ostrowski and Schaub (2012).*

(RQ2) In which manner can a closer approximation of the external source behavior be integrated into existing HEX-algorithms to improve the efficiency of the expensive external minimality check?

The idea is to exploit additional semantic information declared for external sources to allow skipping of the minimality check more frequently than previously possible (Eiter, Fink, Krennwallner, et al., 2014).

(RQ3) Is it possible to avoid the well-known *grounding bottleneck* of ASP during HEX-evaluation by interleaving the grounding with the solving process as well as with external source evaluation?

To investigate this question, our aim is to design a novel HEX-algorithm based on techniques from lazy grounding *(Taupe et al., 2019), and to integrate a lazy-grounding ASP-solver as backend into the DLVHEX-system.*

(RQ4) Can the HEX-formalism be profitably applied to problems that require the integration of sub-symbolic methods, and in general for use cases from the field of machine learning?

To address this research question, two novel HEX-applications are developed that leverage the expressiveness of HEX and employ the formalism in machine learning in different ways.

Finally, an additional goal of this research is to implement all newly developed evaluation algorithms in the HEX-program solver DLVHEX, and to investigate their

performance using benchmark problems. This is also essential in order to be able to adequately answer the above research questions.

1.4 Contributions and Outline

After introducing preliminaries on ASP, HEX-programs, and HEX-evaluation in Chapter 2, which provide the formal context for subsequent chapters, we present our work on novel algorithms for HEX-evaluation in Part I of this thesis. As the main goal of this work is to integrate different parts of HEX-evaluation, we consider each of the three main sub-processes of HEX-evaluation in turn, and develop new HEX-algorithms that tightly integrate them with external evaluation as well as with other sub-processes.

- In Chapter 3, we start by considering the integration of the main search procedure employed during HEX-solving with the evaluation of external sources. A drawback of the state-of-the-art approach is that external atoms are only evaluated under complete assignments (i.e., input to the external source) while in practice, their values often can be determined already based on partial assignments alone (i.e., from incomplete input to the external source). This prevents early backtracking in case of conflicts, and hinders more efficient evaluation of HEX-programs. We thus extend the notion of external atoms to allow for three-valued evaluation under partial assignments, while the two-valued semantics of the overall HEX-formalism remains unchanged. This paves the way for two enhancements: first, to evaluate external sources at any point during model search, which can trigger learning knowledge about the source behavior and/or early backtracking in the spirit of theory propagation in SAT modulo theories (SMT). Second, to optimize the knowledge learned in terms of nogoods. Shrinking nogoods to their relevant input part leads to more effective search space pruning. We further present an experimental evaluation of an implementation of a novel HEX-algorithm that incorporates these enhancements using a benchmark suite. Our results demonstrate a clear efficiency gain over the state-of-the-art HEX-algorithm for the benchmarks, and provide insights regarding the most effective combinations of solver configurations.
- In Chapter 4, we develop techniques for tightly integrating evaluation of external sources and the external minimality check of HEX, which is required to prevent cyclic justifications via external sources, in order to improve its efficiency. As this check often accounts for a large share of the total runtime, optimization is here particularly important. For this, we first extend methods for partial evaluation introduced in the previous chapter to the search employed during the minimality check of HEX. Moreover, syntactic information about atom dependencies has been used previously to detect when the check can be avoided. However, the approach largely overapproximates the real dependencies due to the black-box nature of external sources. In the second part of the chapter, we show how the dependencies can be approximated more closely by also exploiting semantic information, which significantly increases pruning of external minimality checking. Moreover, we

analyze checking and optimization of semantic dependency information. In the end of the chapter, we report results of an empirical evaluation, which exhibit a clear benefit of the new methods.

- In Chapter 5, our goal is to mitigate the well-known grounding bottleneck of ASP during HEX-evaluation by interleaving the grounding and the solving process, also taking external evaluations into account. For achieving this goal, we exploit recent advances in lazy-grounding ASP-solving. While ASP-solving is traditionally based on grounding the input program first, lazy grounding generates new rule instances on-the-fly only when they are needed. We explore this approach in the context of HEX and present a novel evaluation algorithm for HEX-programs based on lazy-grounding solving for ASP. Nonmonotonic dependencies and the import of new constants from external sources make an efficient solution nontrivial. Accordingly, a novel interface for evaluating external sources and special safety criteria had to be designed for the integration. However, illustrative benchmarks show a clear advantage of the new algorithm for grounding-intense programs, which is a new perspective to make HEX more suitable for real-world application needs.

As HEX allows to integrate different formalisms, it is well-suited for combining diverse forms of reasoning. In Part II of this thesis, the main goal is to exploit this strength for two new applications in the area of machine learning. The first application encodes an existing approach for logic-based machine learning in HEX, while the second application integrates an external statistical classifier and a spatial reasoner by means of external atoms into a HEX-encoding.

- In Chapter 6, we apply the HEX-formalism for meta-interpretive learning (MIL), which learns logic programs from examples by instantiating meta-rules and has been implemented before in the Metagol-system based on Prolog. Viewing MIL-problems as combinatorial search problems, they can alternatively be solved by ASP, which can result in performance gains as a result of efficient conflict propagation. However, a straightforward ASP-encoding of MIL results in a huge search space due to a lack of procedural bias and the need for grounding. To address these challenging issues, we encode MIL in the HEX-formalism, which allows us to outsource the background knowledge, and we restrict the search space to compensate for a procedural bias in ASP. This way, the import of constants from the background knowledge can for a given type of meta-rules be limited to relevant ones. Moreover, by abstracting from term manipulations in the encoding and by exploiting the interface mechanism of HEX, the import of such constants can be entirely avoided in order to mitigate the grounding bottleneck. To empirically evaluate the new MIL-approach, we conducted a number of experiments, which show promising results.

- In Chapter 7, we consider the problem of classifying visual objects in scene images by exploiting their semantic context. For this task, we define hybrid classifiers (HC) that combine local statistical classifiers with context constraints, and can be

applied to collective classification problems (CCPs) in general. Context constraints are represented by weighted ASP-constraints using object relations. To integrate probabilistic information provided by the classifier and the context, we embed our encoding in the formalism LP^{MLN}, and show that an optimal labeling can be efficiently obtained from the corresponding LP^{MLN}-program via a back-translation from LP^{MLN} into HEX-programs and by exploiting existing HEX-solvers. Moreover, we describe a methodology for constructing an HC for a CCP, and present experimental results of applying an HC for object classification in indoor and outdoor scenes, which exhibit significant improvements in terms of accuracy compared to using only a local classifier.

We conclude in Chapter 8 by summarizing our main contributions, and we discuss in which regards the work developed during the doctoral research has improved the state of HEX-evaluation. Moreover, we give an overview over remaining open issues and possible directions for future research.

1.5 Evolution of This Work and Relevant Publications

In the beginning of this doctoral project, which started in August 2015 and was mainly conducted within the context of the research project "Integrated Evaluation of Answer Set Programs and Extensions"[5] funded by the *Austrian Science Fund* (project number P27730), we considered the tighter integration of HEX-solving and external evaluations. Techniques devised for this purpose served as a basis for all subsequently developed HEX-algorithms. The results obtained during the first phase of the project have been presented at the "25$^{\text{th}}$ International Joint Conference on Artificial Intelligence" in July 2016 (Eiter, Kaminski, Redl, & Weinzierl, 2016), and have subsequently been extended and published in the "Journal of Artificial Intelligence Research" (Eiter, Kaminski, Redl, & Weinzierl, 2018). The presentation of new techniques for tightly integrating solving and external evaluation in Chapter 3 uses material from both publications; and parts of Chapters 1 and 2 appear in the journal paper.

In parallel to evolving the integration of the solving and the external evaluation process, we developed our first application of HEX-programs in the area of machine learning, which integrates a statistical classifier and ASP-constraints for image classification. The approach has been presented at the "15$^{\text{th}}$ European Conference On Logics In Artificial Intelligence" in November 2016 (Eiter & Kaminski, 2016). Chapter 7 discusses a further developed version of the approach and is based on the corresponding material. While the first version of this approach did not leverage the expressiveness of external atoms, they have been integrated afterwards and their usage is discussed in this thesis.

In the second project phase, we exploited the previously developed techniques and the novel lazy-grounding ASP-solver ALPHA for integrating the grounding process of HEX with the solving process (as well as with external evaluations) in order to avoid the usual grounding bottleneck of ASP during HEX-evaluation. The resulting approach encompasses

[5] http://www.kr.tuwien.ac.at/research/projects/inthex/

a novel lazy-grounding HEX-algorithm and has been published in the proceedings of the "26th International Joint Conference on Artificial Intelligence" (Eiter, Kaminski, & Weinzierl, 2017). The content of the paper has been extended with detailed proofs for all theoretical results and is used in Chapter 5, which discusses the integration of HEX-grounding and solving.

In spring of 2017, we also created a tutorial article for the "13th Reasoning Web Summer School" (Eiter, Kaminski, Redl, Schüller, & Weinzierl, 2017), which served as the basis for a lecture given at the summer school in July 2017 and provides a broad overview over the HEX-formalism as well as many of its use cases. Parts of this publication are used in Chapter 1.

The second application of HEX-programs in the area of machine learning has been developed during a research stay at the *National Institute of Informatics* in Tokyo in collaboration with Prof. Katsumi Inoue. The approach differs from the firstly developed HEX-application in that external atoms are not only used to interface an external machine learning method, but an approach for logic-based machine learning, *meta-interpretive learning*, has itself been implemented by means of HEX-encodings. The according results have been published in the proceedings of the "34th International Conference on Logic Programming" in July 2018 (Kaminski, Eiter, & Inoue, 2018b); the contribution won the *best paper award* of the conference. The new HEX-application is presented in Chapter 6, which uses material from the corresponding publication. The chapter also discusses an extension of the approach, which has been presented in the work-in-progress track of the "28th International Conference on Inductive Logic Programming" in September 2018 (Kaminski, Eiter, & Inoue, 2018a).

In the final phase of this project, we devoted particular attention to the third main subprocess of HEX-evaluation, the external minimality check, which is special to the HEX-formalism and the reason that problems of higher computational complexity compared to related formalisms can be encoded by HEX using Horn-style programs (i.e. without using disjunction or negation). In this regard, we integrated additional information about the semantics of external sources into a check for the necessity of the expensive external minimality check. A paper that introduces the new method has been presented at the "15th International Conference on Logic Programming and Nonmonotonic Reasoning" in June 2019 (Eiter & Kaminski, 2019), where it was awarded with the *best student paper award*. Chapter 4 discusses the integration of external minimality checking and external evaluation; besides material on extending partial evaluation to the external minimality check published in (Eiter, Kaminski, Redl, & Weinzierl, 2018), it incorporates the material on our new method for minimality check skipping.

In order to avoid fragmentation of the text in this thesis and according to common practice, usage of material from the above mentioned publications is not additionally indicated for each passage; and the results presented in this thesis correspond to the results from the respective publications if not stated otherwise.

CHAPTER 2

Preliminaries

In this chapter, we start by introducing the necessary background regarding ASP and HEX-programs, which subsequent chapters will build on. We follow Eiter et al. (2018) for preliminaries.

Our vocabulary consists of a set $\mathcal{P}$ of *predicate symbols*, where each predicate symbol $p \in \mathcal{P}$ has a fixed arity $ar(p)$, a set $\mathcal{C}$ of *constants symbols*, and a set $\mathcal{X}$ of (*first-order*) *variables symbols*, where $\mathcal{X}$ is disjoint from the sets $\mathcal{P}$ and $\mathcal{C}$.

An *atom* is of the form $p(t_1, \ldots, t_\ell)$, abbreviated as $p(\vec{t})$, with predicate $p \in \mathcal{P}$ of arity ℓ and *terms* $t_1, \ldots, t_\ell \in \mathcal{C} \cup \mathcal{X}$.[1] An atom $p(t_1, \ldots, t_\ell)$ is *ground* if $t_1, \ldots, t_\ell \in \mathcal{C}$. For a vector $\vec{t} = t_1, \ldots, t_\ell$ we write $t \in \vec{t}$ if $t = t_i$ for some $1 \leq i \leq \ell$.

2.1 Answer Set Programs

In ASP, problems are encoded by sets of nonmonotonic rules, which can be read as *if-then* expressions, i.e. if the assertions on the right-hand side of a rule hold, then at least one of the elements on the left-hand side must hold as well.

Definition 2.1 (Answer Set Program)**.** *An* answer set program P *is a finite set of (disjunctive) rules* r *of the form*

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n, \tag{2.1}$$

where all a_i, $1 \leq i \leq k$, *and* b_j, $1 \leq j \leq n$, *are atoms. The* head *of a rule* r *is* $H(r) = \{a_1, \ldots, a_k\}$, *its* body *is* $B(r) = \{b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n\}$, *and its* positive *resp.* negative body *is* $B^+(r) = \{b_1, \ldots, b_m\}$ *resp.* $B^-(r) = \{b_{m+1}, \ldots, b_n\}$.

A rule r is called a (*disjunctive*) *fact* if $B(r) = \emptyset$, and an *integrity constraint* if $H(r) = \emptyset$. An answer set program P is called *normal logic program* if $k = 1$ for all $r \in P$;

[1] Terms containing function symbols are not considered in this thesis.

and *definite program* if $k = 1$ and $m = n$ for all $r \in P$. As usual, an atom (rule, program etc.) is *ground*, if no variable occurs in it.

For a program P we let $X(P) = \bigcup_{r \in P} X(r)$ for each $X \in \{H, B, B^+, B^-\}$ to denote the sets of literals that occur in rule heads (H) and rule bodies (B), and the sets of atoms that occur in the positive (B^+) and the negative rule bodies (B^-), respectively.

In the context of ASP, interpretations are usually *Herbrand interpretations*. In this setting, programs with variables can be reduced to programs without variables, by instantiating the variables in rules in all possible ways with constants from $\mathcal{C}$. This process, which is known as *grounding* (Kaufmann et al., 2016), is also adopted commonly by solvers in practice, where in addition optimization steps are made in order to avoid useless rules. Suitable syntactic and/or semantic safety conditions guarantee that a finite number of rule instances suffices for answer set computation. After grounding, in a second *solving phase* the answer sets of the program are then computed.

A rule r of the form (2.1) is called *safe* if all variables that occur in r occur in $B^+(r)$ as well, and we assume in the following that all rules are safe. Since under this condition, a ground program over a finite vocabulary can always be obtained from an answer set program with variables s.t. their answer sets are identical, we can assume in the sequel that the vocabulary (and in particular the sets of constant and predicate symbols $\mathcal{C}$ and $\mathcal{P}$) is finite, and that it suffices to consider ground programs for defining program semantics; in examples, we may use rules containing variables standing for all ground instances with respect to this set of constants. Moreover, if not stated otherwise, all definitions are implicitly parameterized with the according finite vocabulary. By $\mathcal{HB}_{\mathcal{P},\mathcal{C}}$ we denote the finite *Herbrand base* that contains all ground atoms constructible from $\mathcal{P}$ and $\mathcal{C}$; we write $\mathcal{HB}$ if $\mathcal{P}$ and $\mathcal{C}$ are clear from the context.

Herbrand interpretations are usually represented by the sets of ground atoms that are true in them. However, when discussing the evaluation of answer set programs, it is often more convenient to explicitly represent which atoms are assigned to *true* resp. *false*. Following Drescher et al. (2008), a (*signed*) *literal* is either a positive or a negated ground atom $\mathbf{T}a$ (intuitively, a is true) or $\mathbf{F}a$ (a is false), where a is a ground atom. For $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, we let $\overline{\sigma} = \mathbf{T}$ if $\sigma = \mathbf{F}$ and $\overline{\sigma} = \mathbf{F}$ if $\sigma = \mathbf{T}$, and for a literal $L = \sigma a$, we let $\overline{L} = \overline{\sigma} a$. A *complete assignment*[2] over a (finite) set A of atoms is a set $\mathbf{A}$ of literals such that for all $a \in A$, $\mathbf{T}a \in \mathbf{A}$ iff $\mathbf{F}a \notin \mathbf{A}$; here $\mathbf{T}a \in \mathbf{A}$ expresses that a is true and $\mathbf{F}a \in \mathbf{A}$ that a is false. Moreover, we define that $\mathbf{A}(a) = \mathbf{T}$ if $\mathbf{T}a \in \mathbf{A}$, and $\mathbf{A}(a) = \mathbf{F}$ otherwise. The Herbrand interpretation I corresponding to a complete assignment $\mathbf{A}$ is $I = \{a \mid \mathbf{T}a \in \mathbf{A}\}$. For simplicity, we will not switch between Herbrand interpretations and complete assignments, which can be used interchangeably when the Herbrand base is finite; instead we only talk about assignments in the rest of this thesis.

Let $\mathbf{A}$ be a complete assignment. Then $\mathbf{A}$ satisfies a ground atom a, denoted $\mathbf{A} \models a$, if $\mathbf{T}a \in \mathbf{A}$, and it does not satisfy it, denoted $\mathbf{A} \not\models a$, if $\mathbf{F}a \in \mathbf{A}$. Furthermore, $\mathbf{A}$ satisfies a default-negated atom $\mathrm{not}\, a$, denoted $\mathbf{A} \models \mathrm{not}\, a$, if $\mathbf{A} \not\models a$, and it does not satisfy it,

[2] Here, *complete* refers to the fact that the complete assignment defines for each atom $a \in A$ whether it is true or false. We explicitly say *complete* in this section in order to distinguish it from the more general concept of partial assignments we introduce in Chapter 3.

denoted $\mathbf{A} \not\models \text{not}\, a$, if $\mathbf{A} \models a$. A ground rule r is satisfied by $\mathbf{A}$, denoted $\mathbf{A} \models r$, if either $\mathbf{A} \models a$ for some $a \in H(r)$ or $\mathbf{A} \not\models a$ for some $a \in B(r)$. A ground answer set program P is satisfied by $\mathbf{A}$, denoted $\mathbf{A} \models P$, if $\mathbf{A} \models r$ for all $r \in P$.

Answer set programs are interpreted under the *answer set semantics* based on the well-known *Gelfond–Lifschitz* (*GL-*)*reduct* by Gelfond and Lifschitz (1991). Given a ground answer set program P and a complete assignment $\mathbf{A}$, the GL-reduct of P w.r.t. $\mathbf{A}$ is the program

$$P^{\mathbf{A}} = \{H(r) \leftarrow B^+(r) \mid r \in P, \mathbf{A} \not\models b \text{ for all } b \in B^-(r)\}.$$

For complete assignments $\mathbf{A}_1$ and $\mathbf{A}_2$, let $\mathbf{A}_1 \leq \mathbf{A}_2$ denote that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A}_1\} \subseteq \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A}_2\}$ holds. A complete assignment $\mathbf{A}$ is an *answer set* of an answer set program P if $\mathbf{A}$ is a $\leq$-minimal model of $P^{\mathbf{A}}$. The general idea of ASP is to encode a problem by means of an answer set program and to extract corresponding solutions from the respective answer sets.

Example 2.1. Consider the answer set program $P = \{\, a \leftarrow \text{not}\, b.;\ b \leftarrow \text{not}\, a.;\ \leftarrow a. \,\}$ and the complete assignment $\mathbf{A} = \{\mathbf{F}a, \mathbf{F}b\}$. It is easy to see that $\mathbf{A}$ is not a $\leq$-minimal model of $P^{\mathbf{A}} = \{\, a \leftarrow .;\ b \leftarrow .;\ \leftarrow a. \,\}$ and thus, not an answer set of P. On the other hand, $\mathbf{A}' = \{\mathbf{F}a, \mathbf{T}b\}$ is a $\leq$-minimal model of $P^{\mathbf{A}'} = \{\, b \leftarrow .;\ \leftarrow a. \,\}$, and it is the only answer set of P as $\mathbf{A}'' = \{\mathbf{T}a, \mathbf{F}b\}$ is not a model of $P^{\mathbf{A}''} = \{\, a \leftarrow .;\ \leftarrow a. \,\}$ due to the constraint "$\leftarrow a.$". △

Sets of signed literals are also utilized to formulate constraints w.r.t. assignments, i.e. to specify combinations of signed literals that are not permitted to be part of a complete assignment. A *nogood* is a set $\{L_1, \ldots, L_n\}$ of literals; and a complete assignment $\mathbf{A}$ is a *solution* to a nogood δ resp. a set of nogoods Δ, if $\delta \not\subseteq \mathbf{A}$ resp. $\delta \not\subseteq \mathbf{A}$ holds for all $\delta \in \Delta$.

Example 2.2. The complete assignment $\mathbf{A} = \{\mathbf{T}p(a), \mathbf{T}p(b), \mathbf{T}p(c)\}$ is a solution to the nogood $\{\mathbf{T}p(a), \mathbf{F}p(b)\}$, but not to the nogood $\{\mathbf{T}p(a), \mathbf{T}p(b)\}$. △

Nogoods correspond to clauses as known from SAT-solving, and are utilized by ASP-solvers that are based on *conflict-driven nogood learning* (*CDNL*) (Drescher et al., 2008) for representing the input program and for guiding the search by learning additional nogoods from conflicts.

2.2 HEX-Programs

In this section, we introduce HEX-programs, which generalize (disjunctive) logic programs under the answer set semantics by integrating external sources of computation; for more details and background, cf. (Eiter et al., 2005b; Eiter, Fink, Krennwallner, et al., 2014).

As in the case of answer set programs, we can restrict our theoretical investigation of HEX-programs to ground programs because safety conditions allow for applying an advanced grounding algorithm to compute finite groundings, cf. (Eiter, Fink, Krennwallner, & Redl, 2016). However, our examples will also use variables as shortcuts for instantiations with all possible values.

2.2.1 Syntax

HEX-programs extend ordinary ASP-programs by *external atoms*, which enable a bidirectional interaction between a program and external sources of computation. In addition to the sets $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{X}$ introduced above, we assume a further finite set $\mathcal{G}$ of external predicate symbols in our vocabulary, which is disjoint from $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{X}$. External predicate symbols in $\mathcal{G}$ are prefixed with '&' to distinguish them from ordinary predicate symbols, and each $\&g \in \mathcal{G}$ has fixed input and output arity $ar_I(\&g)$ and $ar_O(\&g)$, respectively. Again, due to our restriction of the formal discussion to ground programs, it is sufficient to consider only a finite vocabulary.

Informally, external atoms are associated with input and output values, where constants and the extensions of predicates in the input are provided to an external source which computes whether the respective output values are correct.

More formally, a *ground external atom* is of the form $\&g[\vec{p}](\vec{c})$, where $\&g \in \mathcal{G}$, $\vec{p} = p_1, \ldots, p_k$, with $k = ar_I(\&g)$, is a list of input parameters (predicate names or object constants), called *input list*, and $\vec{c} = c_1, \ldots, c_l$, with $l = ar_O(\&g)$, are constant output terms. More generally, a *non-ground external atom* is of the form $\&g[\vec{Y}](\vec{X})$, where $\vec{Y} = Y_1, \ldots, Y_k$ is a list of input terms (variables, predicate names or object constants), and $\vec{X} = X_1, \ldots, X_l$ are output terms, i.e, variables or object constants. Given a ground external atom $\&g[\vec{p}](\vec{c})$, we call $\&g[\vec{p}]$ a *ground external (ge-)predicate*.

In contrast to answer set programs, in HEX-programs external atoms can be used in the bodies of rules to specify dependencies on external sources. Formally:

Definition 2.2 (HEX-Program)**.** *A HEX-program* Π *consists of rules* r *of the form (2.1), where each* a_i*,* $1 \leq i \leq k$*, is an ordinary atom and each* b_j*,* $1 \leq i \leq n$*, is either an ordinary atom or an external atom.*

In the following, we call a program *ordinary* if it does not contain external atoms, i.e., if it is a standard ASP-program; as usual, an (ordinary or external) atom, rule, program etc. is *ground*, if it is variable-free. The head $H(r)$, the body $B(r)$, the positive body $B^+(r)$ and the negative body $B^-(r)$ of a rule r in a HEX-program are defined as before for ordinary programs. We let $B_o^+(r)$ resp. $B_o^-(r)$ be the set of ordinary atoms in $B^+(r)$ resp. $B^-(r)$. Moreover, we denote by $A(\Pi)$ the set of ordinary atoms that occur in a HEX-program Π.

As for ASP-programs, a rule r in a HEX-program is *safe*, if each variable occurring in r also occurs in $B^+(r)$, and every rule r in a HEX-program Π must be safe. Moreover, to ensure external atoms introduce only finitely many new constants, we assume Π is *liberal domain-expansion* (*lde*)-safe (cf. (Eiter, Fink, Krennwallner, & Redl, 2016) for more details). The notion of lde-safety allows to modularly combine syntactic and/or semantic safety criteria to guarantee that a HEX-program is finitely groundable, and it is the most liberal safety notion that has been considered for the HEX-formalism.

Like ordinary answer set programs, HEX-programs can be reduced by grounding to variable-free programs, where in a non-ground external atom $\&g[\vec{Y}](\vec{X})$ each variable Y_i in $\vec{Y} = Y_1, \ldots, Y_k$ is instantiated with a predicate name or an object constant, and each variable X_i in $\vec{X} = X_1, \ldots, X_l$ with an object constant. The grounding $grnd(r)$ of a rule

r is the set of all possible rules $r\sigma$ that result from r by applying a *(ground) substitution* $\sigma : \mathcal{V} \to \mathcal{C}$; the grounding of program Π is $grnd(\Pi) = \bigcup_{r \in \Pi} grnd(r)$.

According to Definition 2.2, the usage of external atoms is restricted to the rule bodies in a HEX-program as they can only be queried for information. A common use case of external atoms consists in eliminating answer sets based on external constraints as illustrated by the following example.

Example 2.3. Consider the HEX-program Π that consists of the following facts and rules:

$$\begin{aligned} node(a).\ \ node(b). & \\ edge(X,Y) \vee n\_edge(X,Y) \leftarrow\ & node(X), node(Y), X \neq Y. \\ \leftarrow\ & \&geq[edge, 2](). \end{aligned}$$

Informally, the rule on the second line guesses edges (arcs) of a self-loop-free directed graph whose vertices are given as facts on the first line. The constraint on the third line uses an external atom $\&geq[edge, 2]()$ to check whether the number of edges is at most one, by eliminating the guess if at least two edges exist. △

2.2.2 Semantics

Next, we discuss the semantics of ground HEX-programs Π, which generalizes the answer set semantics of Gelfond and Lifschitz (1991). In the following, if not stated otherwise, assignments are over the set $A = A(\Pi)$ of ordinary atoms that occur in the ground HEX-program Π at hand. Furthermore, we let $\mathbb{A}_{\mathcal{P},\mathcal{C}}$ be the set of all possible complete assignments over predicates $\mathcal{P}$ and constants $\mathcal{C}$; in the following we will drop $\mathcal{P}, \mathcal{C}$ from the index and denote this set just as $\mathbb{A}$ since the vocabulary is assumed to be fixed.

The semantics of a ground external atom $\&g[\vec{p}](\vec{c})$ w.r.t. a complete assignment $\mathbf{A}$ is given by the value of a $1+ar_I(\&g)+ar_O(\&g)$-ary decidable *two-valued (Boolean) oracle function*

$$f_{\&g} \colon \mathbb{A} \times (\mathcal{P} \cup \mathcal{C})^k \times \mathcal{C}^l \to \{\mathbf{T}, \mathbf{F}\}$$

that is defined for all possible complete assignments $\mathbf{A} \in \mathbb{A}$, and tuples $\vec{p}$ and $\vec{c}$, where k and l are the lengths of $\vec{p}$ and $\vec{c}$, respectively. Thus, $\&g[\vec{p}](\vec{c})$ is true relative to $\mathbf{A}$ (informally, $\vec{c}$ is an output of $\&g$ for input $\vec{p}$), denoted $\mathbf{A} \models \&g[\vec{p}](\vec{c})$, if $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ and false, denoted $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$, otherwise.

Example 2.4. Consider the external atom $\&synonym[words](X)$, where *words* is an input predicate. Its semantics is given by the oracle function $f_{\&synonym}$, where e.g. $f_{\&synonym}(\mathbf{A}, words, automobile) = \mathbf{T}$ and $f_{\&synonym}(\mathbf{A}, words, motorcar) = \mathbf{T}$ in case the word *car* is in the extension of the predicate *words*, i.e. if $\mathbf{T}words(car) \in \mathbf{A}$. △

Importantly, external oracles support *value invention* such that they can be true for output values that do not occur in a respective (non-ground) program. However, all relevant constants are imported by available grounding algorithms (Eiter, Fink, Krennwallner, & Redl, 2016) invoked during the grounding phase of HEX-program evaluation. In practice, oracle functions are realized as solver-plugins, which are usually implemented in *C++* or *Python*.

While in general, the value of an external atom $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$ may depend on any literal in $\mathbf{A}$, we assume in the following that its value depends only on literals over predicates that appear in $\vec{p}$; formally: $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$ for all complete assignments $\mathbf{A}$ and $\mathbf{A}'$ that assign the same truth values to all atoms of the form $p(\vec{c}')$ where $p \in \vec{p}$.

A complete assignment $\mathbf{A}$ satisfies (or models) a ground atom a, denoted $\mathbf{A} \models a$, if $\mathbf{T}a \in \mathbf{A}$; and it models a ground external atom $\&g[\vec{p}](\vec{c})$ if $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$. Satisfaction of ordinary rules and ASP-programs is then extended to HEX-rules and HEX-programs in the obvious way by also taking the satisfaction of external atoms w.r.t. a complete assignment $\mathbf{A}$ into account. The semantics of HEX-programs is defined in terms of a variant of the GL-reduct, which has originally been introduced by Faber, Pfeifer, and Leone (2011) to define a semantics for programs containing arbitrary aggregates.

Definition 2.3 (FLP-Reduct)**.** *The* FLP-reduct *of a ground HEX-program* Π *w.r.t. a complete assignment* $\mathbf{A}$ *is the set* $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models b,\ \text{for all } b \in B(r)\}$ *of all rules whose body is satisfied by* $\mathbf{A}$*.*

The answer sets of a ground HEX-program Π are then defined as follows.

Definition 2.4 (Answer Set of a HEX-Program)**.** *A complete assignment* $\mathbf{A}$ *is an answer set of a ground HEX-program* Π*, if* $\mathbf{A}$ *is a* $\leq$*-minimal model of* $f\Pi^{\mathbf{A}}$*.*

The answer sets of a non-ground HEX-program Π are those of $grnd(\Pi)$. For a given ground HEX-program Π we let $\mathcal{AS}(\Pi)$ denote the set of all answer sets of Π.

For ordinary ASP-programs (i.e., HEX-programs without external atoms), the above definition of answer sets based on the FLP-reduct $f\Pi^{\mathbf{A}}$ is equivalent to the original definition of answer sets by Gelfond and Lifschitz (1991) based on the GL-reduct. However, for HEX-programs, the FLP-reduct is more attractive than the GL-reduct as it prevents unintuitive answer sets involving cyclic justifications. Furthermore, the stronger notion of well-justified answer set by Shen et al. (2014) excludes any cyclic justification whatsoever, as the whole answer must be obtained in a constructive process that involves classical provability.

We illustrate the notion of answer set in the case of HEX-programs on two simple examples.

Example 2.5. The external atom in Example 2.3 is associated with an oracle function $f_{\&geq}(\mathbf{A}, p, n)$ defined as follows:

$$f_{\&geq}(\mathbf{A}, p, n) = \begin{cases} \mathbf{T} & \text{if } |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}\}| \geq n, \\ \mathbf{F} & \text{otherwise.} \end{cases}$$

The complete assignment $\mathbf{A}_1 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{F}edge(b,b)\}$ is an answer set of Π if we add $\mathbf{T}n\_edge(c,c')$ if $\mathbf{F}edge(c,c') \in \mathbf{A}_1$, and we add $\mathbf{F}n\_edge(c,c')$ if $\mathbf{T}edge(c,c') \in \mathbf{A}_1$, where $c, c' \in \{a,b\}$. On the other hand, the assignment $\mathbf{A}_2 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}edge(b,b)\}$ is not an answer set of Π for an analogous addition. As easily seen, Π has three answer sets that correspond to the self-loop-free directed graphs on a, b with less than two edges. △

Example 2.6. Consider as another example the program $\Pi = \{p \leftarrow \&id[p]().\}$, where $\&id[p]()$ is true iff p is true. Then Π has the answer set $\mathbf{A}_1 = \{\mathbf{F}p\}$; indeed $\mathbf{A}_1$ is a $\leq$-minimal model of the reduct $f\Pi^{\mathbf{A}_1} = \emptyset$. We remark that using the traditional GL-reduct, adapted to HEX-programs instead of the FLP-reduct, would admit another answer set $\mathbf{A}_2 = \{\mathbf{T}p\}$; constructing the latter would however involve cyclic justification, which is intuitively not acceptable. △

2.3 Evaluation of HEX-Programs

The basic evaluation procedure for ground HEX-programs uses a guess-and-check rewriting to ordinary ASP (Eiter, Fink, Ianni, et al., 2016) and leverages available solvers such as CLASP (Gebser, Kaufmann, et al., 2011) for HEX-evaluation. To this end, HEX-programs Π are transformed to ordinary programs by replacing each external atom $\&g[\vec{p}](\vec{c})$ in Π by an ordinary *replacement atom* $e_{\&g[\vec{p}]}(\vec{c})$, and by adding a disjunctive fact $e_{\&g[\vec{p}]}(\vec{c}) \vee ne_{\&g[\vec{p}]}(\vec{c}) \leftarrow$ that represents a guess for the truth value of the respective external atom. The answer sets of the resulting *guessing program* $\hat{\Pi}$ are then computed by an ASP-solver. The assignment encoded by such an answer set may not satisfy Π, as $f_{\&g}$ may yield for $\&g[\vec{p}](\vec{c})$ a value that is different from the guess for $e_{\&g[\vec{p}]}(\vec{c})$. Thus, the answer set is merely a *model candidate*; if a check against the external sources finds no discrepancy, it is a *compatible set*. Formally:

Definition 2.5 (Compatible Set)**.** *A* compatible set *of a program* Π *is an answer set* $\hat{\mathbf{A}}$ *of the guessing program* $\hat{\Pi}$ *such that* $f_{\&g}(\hat{\mathbf{A}}, \vec{p}, \vec{c}) = \mathbf{T}$ *iff* $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \hat{\mathbf{A}}$ *for all external atoms* $\&g[\vec{p}](\vec{c})$ *in* Π.

As mentioned in Section 1.2.2 and described by Eiter, Fink, Krennwallner, et al. (2014), nogoods can be learned from the external source evaluations which are performed to ensure that a model candidate is also a compatible set. These nogoods can additionally be provided to an ASP-solver to guide the search for model candidates and to avoid the reoccurrence of wrong guesses for external atoms; up to exponentially many guesses can be excluded by the learned nogoods (Eiter, Fink, Krennwallner, et al., 2014). More precisely, a nogood is learned from the evaluation of an external atom $\&g[\vec{p}](\vec{c})$ w.r.t. a complete assignment $\mathbf{A}$ that represents that $e_{\&g[\vec{p}]}(\vec{c})$ must be true under $\mathbf{A}$ if $f_{\&g}(\hat{\mathbf{A}}, \vec{p}, \vec{c}) = \mathbf{T}$, respectively false under $\mathbf{A}$ if $f_{\&g}(\hat{\mathbf{A}}, \vec{p}, \vec{c}) = \mathbf{F}$; only the part of $\mathbf{A}$ relevant for evaluating $f_{\&g}(\hat{\mathbf{A}}, \vec{p}, \vec{c})$ must be contained in the nogood. Accordingly, a nogood can be learned independent from the correctness of the truth value guessed for $e_{\&g[\vec{p}]}(\vec{c})$, but for incorrect guesses the learned nogood can trigger backtracking directly.

Example 2.7. Assume we are given for the graph guessing program Π from Example 2.3 the complete assignment $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}n\_edge(a,b), \mathbf{F}n\_edge(b,a)\}$. After evaluating the oracle function associated with $\&geq[edge, 2]()$ under $\mathbf{A}$, the nogood $\{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}e_{\&geq[edge,2]}()\}$ can be generated in order to learn that $\mathbf{A} \models \&geq[edge, 2]()$; the nogood encodes that whenever $edge(a,b)$, $edge(b,a)$ are true, the external atom $\&geq[edge, 2]()$ must not be assigned the truth value $\mathbf{F}$.

In addition, for $\mathbf{A}' = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{T}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{F}n\_edge(b,a)\}$, it can be learned that $\mathbf{A}' \not\models \&geq[edge,2]()$ by adding the nogood $\{\mathbf{F}edge(a,b), \mathbf{T}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\}$. △

Each answer set of Π is the projection $\mathbf{A}$ of a compatible set $\hat{\mathbf{A}}$ to the atoms $A(\Pi)$ in Π, but not vice versa. To discard the non-answer sets, the evaluation algorithm calls an FLP-check to check minimality w.r.t. $f\Pi^{\mathbf{A}}$ (Eiter, Fink, Krennwallner, et al., 2014). This check constitutes a second (external) minimality check which intuitively is needed in addition to the usual check that ensures minimality w.r.t. $\hat{\Pi}^{\hat{\mathbf{A}}}$ to prevent cyclic justifications involving external atoms.

Example 2.8 (cont'd). For the program Π from Example 2.3, the guessing program $\hat{\Pi}$ is as follows:

$$\begin{aligned} node(a).\ \ node(b). \\ edge(X,Y) \vee n\_edge(X,Y) &\leftarrow node(X), node(Y), X \neq Y. \\ &\leftarrow e_{\&geq[edge,2]}(). \\ e_{\&geq[edge,2]}() \vee ne_{\&geq[edge,2]}() &\leftarrow . \end{aligned}$$

The answer sets of $\hat{\Pi}$ comprise the sets $\hat{\mathbf{A}}_1 = \mathbf{A}_1 \cup \{\mathbf{F}e_{\&geq[edge,2]}()\}$ where $\mathbf{A}_1 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{F}edge(b,b)\}$, and $\hat{\mathbf{A}}_2 = \mathbf{A}_2 \cup \{\mathbf{F}e_{\&geq[edge,2]}()\}$ where $\mathbf{A}_2 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}edge(b,b)\}$. While $\hat{\mathbf{A}}_1$ is a compatible set of $\hat{\Pi}$, $\hat{\mathbf{A}}_2$ is not. Thus, the latter cannot give rise to some answer set of Π. Regarding $\hat{\mathbf{A}}_1$, it is easy to see that $\mathbf{A}_1$ is a minimal model of the FLP-reduct $f\Pi^{\mathbf{A}_1} = \{node(a).;\ node(b).;\ edge(a,b) \vee n\_edge(a,b) \leftarrow node(a), node(b), a \neq b.;\ edge(b,a) \vee n\_edge(b,a) \leftarrow node(b), node(a), b \neq a.\}$. Hence, $\mathbf{A}_1$ is an answer set of Π. △

Example 2.9. Reconsider $\Pi = \{\, p \leftarrow \&id[p]().\,\}$ from Example 2.6. Then the guessing program $\hat{\Pi} = \{\, p \leftarrow e_{\&id[p]}().;\ e_{\&id[p]} \vee ne_{\&id[p]} \leftarrow .\,\}$ has the answer sets $\hat{\mathbf{A}}_1 = \{\mathbf{F}p, \mathbf{F}e_{\&id[p]}\}$ and $\hat{\mathbf{A}}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$; as easily seen, both are compatible sets of $\hat{\Pi}$. Here the projection $\mathbf{A}_1$ is a $\leq$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$, and thus $\mathbf{A}_1$ is an answer set of $\hat{\Pi}$; on the other hand, $\mathbf{A}_2$ is not a minimal model of $f\Pi^{\mathbf{A}_2} = \Pi$, and thus $\mathbf{A}_2$ is not an answer set of $\hat{\Pi}$. △

As illustrated by these examples, an additional procedure for checking external minimality is required for finding answer sets of HEX-programs, which is described in more detail in the section below.

2.4 External Minimality Check

The basic approach for ensuring *external* (*e-*)*minimality* of the projection $\mathbf{A}$ of a compatible set $\hat{\mathbf{A}}$ for a HEX-program Π w.r.t. the FLP-reduct $f\Pi^{\mathbf{A}}$ (called *explicit FLP-check* by Eiter, Fink, Krennwallner, et al. (2014)) consists in explicitly constructing the FLP-reduct $f\Pi^{\mathbf{A}}$ and checking that it has no model $\mathbf{A}'$ s.t. $\mathbf{A}' \subseteq \mathbf{A}$. In general, performing the

e-minimality check efficiently is highly non-trivial as it is co-NP-complete already for ground *Horn* programs with external atoms that can be evaluated in polynomial time (Eiter, Fink, Krennwallner, et al., 2014).

However, a variant of the e-minimality check based on *unfounded sets*, which is a semantics-based characterization of minimality for answer sets (Leone, Rullo, & Scarcello, 1997) that has been lifted to HEX-programs (Eiter, Fink, Krennwallner, et al., 2014), can be significantly faster than an explicit FLP-check as it avoids the explicit generation of models of $f\Pi^{\mathbf{A}}$. An unfounded set of a HEX-program Π w.r.t. an assignment $\mathbf{A}$ is a set of atoms that can be jointly set to false without violating any rule in Π because they only circularly support each other w.r.t. $\mathbf{A}$. Formally:

Definition 2.6 (Unfounded Set)**.** *Let Π be a ground HEX-program and let $\mathbf{A}$ and U be complete assignments over $A(\Pi)$. Then, U is an* unfounded set *for Π w.r.t. $\mathbf{A}$ if, for each rule r with $H(r) \cap \{a \mid \mathbf{T}a \in U\} \neq \emptyset$, at least one of the following holds, where $\mathbf{A} \dot\cup \neg.U = (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in U\}) \cup \{\mathbf{F}a \mid \mathbf{T}a \in U\}$:*

(1) some literal of $B(r)$ is false w.r.t. $\mathbf{A}$,

(2) some literal of $B(r)$ is false w.r.t. $\mathbf{A} \dot\cup \neg.U$, or

(3) some atom of $H(r) \setminus \{a \mid \mathbf{T}a \in U\}$ is true w.r.t. $\mathbf{A}$.

Note that different from the literature, here we define unfounded sets as complete assignments rather than sets of atoms to make the operator $\dot\cup$ reusable for subsequent results. However, conceptually an unfounded set U still represents a set of atoms, given by the true atoms $\mathbf{T}a \in U$.

Example 2.10. Consider again the HEX-program Π from Example 2.6. The complete assignment $U = \{\mathbf{T}p\}$ is an unfounded set for Π w.r.t. $\mathbf{A} = \{\mathbf{T}p\}$, while no unfounded set for Π w.r.t. $\mathbf{A} = \{\mathbf{F}p\}$ exists. △

The answer sets of a HEX-program Π correspond exactly to those complete assignments $\mathbf{A}$, with $\mathbf{A} \models \Pi$, where the true part of $\mathbf{A}$ does not intersect with any unfounded set for Π w.r.t. $\mathbf{A}$, i.e. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} = \emptyset$ for every unfounded set U for Π w.r.t. $\mathbf{A}$ (Faber, 2005; Eiter, Fink, Krennwallner, et al., 2014).

Part I

Integrated Algorithms for HEX-Program Evaluation

CHAPTER 3

Integration of Solving and External Evaluation

In this chapter, we start by considering the tight integration of the first two sub-processes which are at the core of HEX-evaluation: the solving and the external evaluation process. The chapter is based on the conference paper (Eiter, Kaminski, Redl, & Weinzierl, 2016) and the journal paper (Eiter, Kaminski, Redl, & Weinzierl, 2018).

As discussed in Section 1.2.2, previous evaluation algorithms for (ground) HEX-programs first compute a complete truth-assignment by guessing the truth values of all external atoms and by evaluating an accordingly rewritten program using existing ASP-solvers such as CLASP. Only when the assignment is complete, the correctness of the guess can be verified by calls to the external sources. Despite the enhancement of this basic approach with conflict-driven learning techniques (Eiter et al., 2012), which learn parts of the external source semantics while the search space is traversed, the evaluation of external atoms over complete assignments is an obstacle to good performance in general.

Intuitively, evaluating external sources under yet partial assignments (i.e., assignments in which only some input atoms are set to true or false, while others remain unassigned) may in some cases allow to decide the eventual truth value of an external atom, regardless of how the assignment will be completed. For example, suppose an external atom $\&planar[node, edge]()$ interfaces an external source for checking whether a graph whose nodes and edges are captured by the unary predicate *node* and the binary predicate *edge*, respectively, is a planar graph. If a rule $edge(X, Y) \vee not\_edge(X) \leftarrow connected(X, Y)$ guesses the edges of a graph from a pool of connections, the external checker might detect non-planarity even if the guess is not yet complete (i.e., some edges are missing). Early external evaluation has the potential for significant performance gains, as wrong guesses may be detected early on or avoided entirely. In this way, the external sources can guide the answer set search proactively.

This idea is in the spirit of *theory propagation* in *SAT modulo theories (SMT)* (Barrett et al., 2009). However, adopting evaluations under partial assignments for HEX-programs is non-trivial, because – unlike in SMT, which considers only fixed theories – external sources are largely black boxes, without much information about their structure (as in case of privacy and data hiding, or of a wrapped web service) let alone is a propagation machinery available. Moreover, their heterogeneity and (possibly) nonmonotonic nature, e.g. if the external source access is to ASP engines or argumentation solvers, adds further conceptual and computational complexity.

In this chapter, we address the issue of partial evaluation by extending external source access via external atoms from a Boolean semantics, which is defined only under complete assignments, to a three-valued evaluation semantics that is defined under partial assignments. This extension is exploited for novel evaluation techniques to achieve the main goal of efficiency improvements. In particular, learning about the behavior of external sources during evaluation under partial assignments allows us to acquire additional knowledge that aids in guiding the search, similar as theory propagation in SMT. Moreover, the possibility of such early evaluation enables identifying the part of the input that is relevant for the final value of an external atom. This allows for minimizing the learned nogoods in order to approximate the external source semantics more closely. Importantly, the semantics of the overall formalism remains unchanged, i.e., the three-valued semantics of external sources is only exploited for performance improvements during the search, while the final answer sets are still two-valued.

The rest of this chapter is organized as follows:

- In Section 3.1, we extend the notion of external atoms in two dimensions: first, that they can be evaluated under partial assignments, which set each atom to either true, false, or unassigned. Second, that the output of the evaluation can be either true, false or unknown; this is because the truth value of the external atom might be definitely known (true or false), or it is yet unknown under the current partial input.

- In Section 3.2, we present a novel evaluation algorithm which exploits three-valued evaluation of external sources for early detection of conflicts due to wrong guesses for the truth values of external atoms; and by this, allows for earlier backjumping and improved search space pruning during answer set search.

- In Section 3.3, we consider learning of *input-output nogoods* based on additional knowledge about external sources using abstractly defined learning functions, similar as done by theory propagation in SMT solving. As well-known, learning can be much more effective if structural properties of the underlying domain are known (cf. Valiant (1984)). We thus present also a concrete learning function for monotonic external sources (relative to the input assignment).

- In Section 3.4, we devote particular attention to minimizing learned input-output nogoods, given the interface for evaluation under partial assignments; as already mentioned, small (non-redundant) nogoods are important for pruning the search

space of candidate answer sets effectively. Furthermore, we mitigate the associated minimization costs by exploiting the divide-and-conquer strategy that was introduced by Junker (2004) for conflict set minimization in constraint programming.

- In Section 3.5, we perform an experimental evaluation of the new techniques on a rich benchmark suite using a prototype implementation of our approach in the DLVHEX-system. It appears that each of them can yield significant performance gains, yet the picture of their combination is more complex; in particular, heuristics may lead to diverging (though explainable) behavior. In any case, our experimental results show a speedup of up to two orders of magnitude in performance (in theory, even exponential gains are possible).

- In Section 3.6, we discuss related work, and conclude the chapter in Section 3.7 with a discussion of further issues and future work.

Our techniques are related to *theory propagation* in SMT (Barrett et al., 2009) and minimization techniques in constraint ASP-solvers such as CLINGCON (Ostrowski & Schaub, 2012). These, however, usually rely on a tailored integration of theory solvers crafted by experts, whereas our approach allows a broad range of users, without prior knowledge on solver construction, to harness performance gained by the new learning techniques. In addition, full backward compatibility with existing two-valued source descriptions makes exploiting the new features an option but not a requirement for the use of DLVHEX on legacy and new applications.

3.1 Extension to Partial Assignments

In this section, we start by generalizing complete assignments and oracle functions, as defined in Section 2.1, to partial assignments, which provide a means for explicitly representing that some atom is yet unassigned. To this end, we introduce signed literals $\mathbf{U}a$ to represent that an atom a is yet unassigned in an assignment. Also oracle functions may be extended to deal with unassigned input atoms and in turn, may also evaluate to $\mathbf{U}$ to represent that the value of the corresponding external atom is yet unknown under the given input. This allows us, in the next step, to enhance the existing evaluation algorithm in such a way that external sources are already evaluated early during search, which potentially allows for earlier backtracking. We note that the extended concepts are only used by the algorithm during solving, while the semantics of the formalism remains unchanged. That is, answer sets define the truth values of all atoms, and are thus still two-valued.

We start with a formal definition of the required concepts:

Definition 3.1 (Partial Assignment)**.** *A* partial assignment *over a set A of atoms is a set $\mathbf{A}$ of signed literals of the form $\mathbf{T}a$, $\mathbf{F}a$ and $\mathbf{U}a$ such that for every $a \in A$ it holds that $|\mathbf{A} \cap \{\mathbf{T}a, \mathbf{F}a, \mathbf{U}a\}| = 1$.*

Then, a complete assignment as defined in Section 2.1 corresponds to the special case of a partial assignment which contains no signed literal $\mathbf{U}a$. Since the rest of this chapter will use the more general concept of partial assignment only (with complete assignments as special case thereof), we will drop 'partial' in the rest of the chapter and say only *assignment*.

To avoid the introduction of further symbols and heavy notation, we let $\mathbb{A}_{\mathcal{P},\mathcal{C}}$ denote the set of all three-valued assignments over the given vocabulary from now on; as before we drop $\mathcal{P},\mathcal{C}$ from the index since the vocabulary is fixed. As we use only three-valued assignments in the remaining part of the chapter, this is unambiguous.

For assignments $\mathbf{A}, \mathbf{A}'$ we call $\mathbf{A}'$ an *extension* of $\mathbf{A}$, denoted $\mathbf{A}' \succeq \mathbf{A}$, if it holds that $\mathbf{A} \setminus \{\mathbf{U}a \in \mathbf{A} \mid a \in A\} \subseteq \mathbf{A}'$ (i.e., some unassigned atoms in $\mathbf{A}$ may be flipped to true resp. false to obtain $\mathbf{A}'$). Oracle functions are then extended as follows in order to define the semantics of an external atom $\&g[\vec{p}](\vec{c})$ w.r.t. partial assignments.

Definition 3.2 (Three-Valued Oracle Function)**.** *A* three-valued oracle function $f_{\&g}$ *for a ground external atom* $\&g[\vec{p}](\vec{c})$ *with* k *input and* l *output parameters is a* $1{+}k{+}l$*-ary function*

$$f_{\&g} : \mathbb{A}{\times}(\mathcal{P} \cup \mathcal{C})^k{\times}\mathcal{C}^l \to \{\mathbf{T}, \mathbf{F}, \mathbf{U}\},$$

where $\mathbb{A}$ *is the set of all possible assignments* $\mathbf{A}$*, such that* $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq \mathbf{U}$ *whenever* $\mathbf{A}$ *is a complete assignment.*

Thus, $\&g[\vec{p}](\vec{c})$ is true, false or unassigned relative to $\mathbf{A}$, if the value of $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$ is $\mathbf{T}$, $\mathbf{F}$ or $\mathbf{U}$, respectively. As in the case of two-valued oracle functions, we assume that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$ for all partial assignments $\mathbf{A}$ and $\mathbf{A}'$ that assign the same truth values to all atoms of the form $p(\vec{c}')$ where $p \in \vec{p}$; i.e., the function value only depends on the predicates in $\vec{p}$.

We require that once the output of $f_{\&g}$ is assigned to true or false for some $\mathbf{A}$, the value stays the same for all extensions.

Definition 3.3 (Assignment-Monotonicity)**.** *A three-valued oracle function* $f_{\&g}$ *is* assignment-monotonic *if* $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = X$*,* $X \in \{\mathbf{T}, \mathbf{F}\}$*, implies* $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = X$ *for all assignments* $\mathbf{A}' \succeq \mathbf{A}$*.*

Assignment-monotonicity guarantees that no compatible set is lost when querying external sources on partial assignments.

Example 3.1. Reconsider the HEX-program Π from Example 2.3 in Section 2.2.1 and recall that the (two-valued) oracle function $f_{\&geq}(\mathbf{A}, edge, 2)$ for a complete assignment $\mathbf{A}$ evaluates to true if $\mathbf{A}$ contains at least two literals $\mathbf{T}edge(x, y)$, and to false otherwise.

We can extend the oracle to partial assignments $\mathbf{A}'$ by defining an assignment-monotonic three-valued oracle function $f'_{\&geq}(\mathbf{A}', p, n)$ as follows:

$$f'_{\&geq}(\mathbf{A}', p, n) = \begin{cases} \mathbf{T} & \text{if } |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}'\}| \geq n, \\ \mathbf{U} & \text{if } |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}'\}| < n \text{ and} \\ & \quad |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}' \text{ or } \mathbf{U}p(x,y) \in \mathbf{A}'\}| \geq n, \\ \mathbf{F} & \text{otherwise.} \end{cases}$$

such that $\&geq[edge, 2]()$ can also be evaluated under partial assignments, where the three-valued oracle function $f'_{\&geq}(\mathbf{A}', edge, 2)$ yields true if $|\{edge(x, y) \mid \mathbf{T}edge(x, y) \in \mathbf{A}'\}| \geq 2$, unassigned if $|\{edge(x, y) \mid \mathbf{T}edge(x, y) \in \mathbf{A}'\}| < 2$ and $|\{edge(x, y) \mid \mathbf{T}edge(x, y) \in \mathbf{A}' \text{ or } \mathbf{U}edge(x, y) \in \mathbf{A}'\}| \geq 2$ (i.e. when two edges can still potentially be mapped to true by an extension of $\mathbf{A}'$), and false otherwise. △

Note that the definition of answer sets carries immediately over to programs with external atoms that use three-valued oracle functions. This is because answer sets are complete assignments and thus, the oracle function call for an external atom $\&g[\vec{p}](\vec{c})$ evaluates to either $\mathbf{T}$ or $\mathbf{F}$ according to Definition 3.2. It is therefore not necessary to extend the definitions of satisfaction of ordinary atoms, rules, and programs, and the definition of answer sets, to partial assignments.

A two-valued oracle function, however, cannot handle partial assignments and is thus *not* a special case of a three-valued oracle function that can be passed to an algorithm expecting the latter. However, we can always obtain a three-valued from a two-valued oracle function such that answer sets remain invariant.

Proposition 3.1. *For every* HEX*-program* Π *and external predicate* $\&g$ *defined by a two-valued oracle function, one can redefine* $\&g$ *by an assignment-monotonic three-valued oracle function without changing the answer sets of* Π.

Proof. For each external predicate $\&g$ we introduce a new external predicate $\&g'$, construct program Π' by replacing all occurrences of $\&g$ in Π by $\&g'$, and define $f_{\&g'}(\mathbf{A}, \cdot, \cdot) = f_{\&g}(\mathbf{A}, \cdot, \cdot)$ if $\mathbf{A}$ is complete over Π and $f_{\&g'}(\mathbf{A}, \cdot, \cdot) = \mathbf{U}$ otherwise. Since under complete assignments all external atoms in Π have the same truth values as the corresponding external atoms in Π', and answer sets are complete assignments by definition, it follows immediately that $\mathcal{AS}(\Pi) = \mathcal{AS}(\Pi')$. □

Intuitively, we construct a three-valued oracle function which coincides with the two-valued one for complete assignments, and returns $\mathbf{U}$ otherwise. Hence, Proposition 3.1 allows us to "wrap" two-valued oracle functions for use by our algorithms below; in the implementation this is the basis for backwards compatibility with existing external sources.

We exploit partial assignments by extending previous evaluation algorithms. In the spirit of *theory propagation* in SMT solvers (Barrett et al., 2009), we use *external theory learning (ETL)*. It is related to *external behavior learning*, which encodes observed output of external sources as nogoods (Eiter et al., 2012), but our extension works over partial assignments such that external sources may drive early propagation of truth values implied by the current partial assignment.

As for external behavior learning, we can associate with each external source a *learning-function* Λ that yields a set of nogoods $\Lambda(\&g[\vec{p}], \mathbf{A})$ learned from the evaluation of $\&g[\vec{p}]$ under an assignment $\mathbf{A}$. Learned nogoods have to be correct, i.e., they must not eliminate compatible sets. Formally, a nogood δ is *correct w.r.t. a program* Π, if all compatible sets of Π are solutions to δ.

Algorithm 3.1: HEX-CDNL with Partial Evaluation

Input: A HEX-program Π
Output: An answer set of Π if one exists, and $\bot$ otherwise

Let $\hat{\Pi}$ be the guessing program of Π
$\hat{\mathbf{A}} \leftarrow \{\mathbf{U}a \mid a \in A(\Pi)\}$ `// all atoms unassigned`
$\nabla \leftarrow \emptyset$ `// set of dynamic nogoods`
$dl \leftarrow 0$ `// decision level`
while *true* **do**
(a) $(\hat{\mathbf{A}}, \nabla) \leftarrow \mathsf{Propagation}(\hat{\Pi}, \nabla, \hat{\mathbf{A}})$
(b) **if** *some nogood* δ *violated by* $\hat{\mathbf{A}}$ **then**
  if $dl = 0$ **then return** $\bot$
  analyze conflict, add learned nogood to ∇, set dl to backjump level
(c) **else if** $\hat{\mathbf{A}}$ *is complete* **then**
  $\mathbf{A} \leftarrow \hat{\mathbf{A}} \cap \{\mathbf{T}a, \mathbf{F}a \mid a \in A(\hat{\Pi})\}$
  if *there is an unfounded set* U *of* $\hat{\Pi}$ *w.r.t.* $\hat{\mathbf{A}}$ *s.t.* $U \cap \{\mathbf{T}a \mid \mathbf{T}a \in \hat{\mathbf{A}}\} \neq \emptyset$ **then**
    construct violated nogood for U and add it to ∇
    analyze conflict, add learned nogood to ∇, set dl to backjump level
(d) **else if** $\hat{\mathbf{A}}$ *is not compatible for* $\hat{\Pi}$ *or* $\mathbf{A}$ *is not a minimal model of* $f\Pi^{\mathbf{A}}$ **then**
    $\nabla \leftarrow \nabla \cup \{\hat{\mathbf{A}}\}$
  else
    return $\mathbf{A}$
  end
(e) **else if** *heuristics evaluates* $\&g[\vec{y}]$ *and* $\Lambda(\&g[\vec{y}], \hat{\mathbf{A}}) \not\subseteq \nabla$ **then**
  $\nabla \leftarrow \nabla \cup \Lambda(\&g[\vec{y}], \hat{\mathbf{A}})$
(f) **else**
  Guess $\sigma a \in \{\mathbf{T}a, \mathbf{F}a\}$ for some atom a with $\mathbf{U}a \in \hat{\mathbf{A}}$
  $dl \leftarrow dl + 1$
  $\hat{\mathbf{A}} \leftarrow (\hat{\mathbf{A}} \setminus \{\mathbf{U}a\}) \cup \{\sigma a\}$
 end
end

3.2 HEX-Algorithm Based on Partial Assignments

We extend learning functions for partial assignments as follows. Let $\mathcal{E}(\Pi)$ contain all ge-predicates $\&g[\vec{p}]$ that occur in Π, and let $\mathcal{L}(\hat{\Pi}) = \{\mathbf{T}a, \mathbf{F}a, \mathbf{U}a \mid a \in A(\hat{\Pi})\}$ denote the set of all signed literals on atoms that occur in $\hat{\Pi}$.

Definition 3.4 (Three-Valued Learning Function)**.** *A* (three-valued) learning function *for a* HEX*-program* Π *is a mapping* $\Lambda\colon \mathcal{E}(\Pi) \times \mathbb{A} \to 2^{2^{\mathcal{L}(\hat{\Pi})}}$ *that assigns each ge-predicate* $\&g[\vec{p}]$ *and partial assignment* $\mathbf{A}$ *a set* $\Lambda(\&g[\vec{p}], \mathbf{A})$ *of nogoods. We call* Λ correct *for* Π*, if for all arguments* $\&g[\vec{p}] \in \mathcal{E}$ *and* $\mathbf{A} \in \mathbb{A}$*, every nogood* $\delta \in \Lambda(\&g[\vec{p}], \mathbf{A})$ *is correct for* Π*.*

Throughout the rest, we assume that learning functions are always correct for the programs at hand.

We now present a procedure for computing an answer set of a HEX-program, shown in Algorithm 3.1 and illustrated by Figure 3.1. To compute multiple answer sets, we can naively add previous answer sets as constraints and call the algorithm again (cf. Gebser et al. (2007) for more elaborated techniques). The basic structure of Algorithm 3.1 resembles an ordinary ASP-solver, but has additional checks in Part (c) and external calls to learn further nogoods in Part (e), which is based on partial assignments. Without the extensions, it computes an answer set $\hat{\mathbf{A}}$ of the guessing program $\hat{\Pi}$ and returns the projection of $\hat{\mathbf{A}}$ to the atoms in Π (cf. Drescher et al. (2008)). To this end, it starts from a void assignment and performs unit propagation in Part (a) to derive further truth values. Part (b) backtracks and learns nogoods from conflicts. Part (c) checks compatibility and minimality of the model candidate. To this end, the (more efficient) check in the **if**-block checks minimality from the perspective of an ordinary ASP-solver without respecting the semantics of external sources (i.e., minimality of $\hat{\mathbf{A}}$ w.r.t. $\hat{\Pi}$); the minimality check is realized using *unfounded sets* introduced in Section 2.4, which are atoms that support each other only cyclically. If this check fails, the algorithm learns a nogood and backtracks. Only if this check is passed, the **else if**-block in Part (d) checks compatibility and e-minimality under consideration of external sources (i.e., e-minimality of $\mathbf{A}$ w.r.t. Π), cf. Definition 2.4. We will discuss the e-minimality check in detail in Chapter 4. If this check is also passed, an answer set has been found. Finally, without Part (e), the algorithm makes a guess in Part (f) if no further truth values can be derived and the assignment is incomplete.

The additional calls to external sources and nogood learning in Part (e) are not mandatory but prune the search space, and may be executed more or less frequently according to different heuristics; they may eliminate assignments violating known behavior of external sources already early during the search, while the correctness of the learning function Λ guarantees that no compatible set of $\hat{\Pi}$ (and hence no answer set of Π) is eliminated. Notably and in contrast to previous algorithms (Eiter et al., 2012), external atoms are evaluated under partial assignments and use a three-valued oracle function.

We can show that this algorithm is sound and complete:

Theorem 3.1. *If Algorithm 3.1 returns for an input program* Π *(i) an assignment* $\mathbf{A}$*, then* $\mathbf{A}$ *is an answer set of* Π*; (ii) the symbol* $\bot$*, then* Π *is inconsistent.*

Proof. The algorithm extends the conflict-driven algorithm for ordinary ASP as follows:

- The check for compatibility of $\hat{\mathbf{A}}$ and for minimality w.r.t. $f\Pi^{\mathbf{A}}$ in the **if**-block of Part (c) is added.
- The evaluation of external atoms and addition of nogoods in Part (e).

Without these changes, soundness and completeness of the algorithm for ordinary ASP (as presented by Drescher et al. (2008)) implies that the algorithm returns the projection of some answer set $\hat{\mathbf{A}}$ of $\hat{\Pi}$ to $A(\Pi)$ if $\hat{\Pi}$ is consistent, and $\bot$ otherwise. We now show that the changes adopt the behavior as desired.

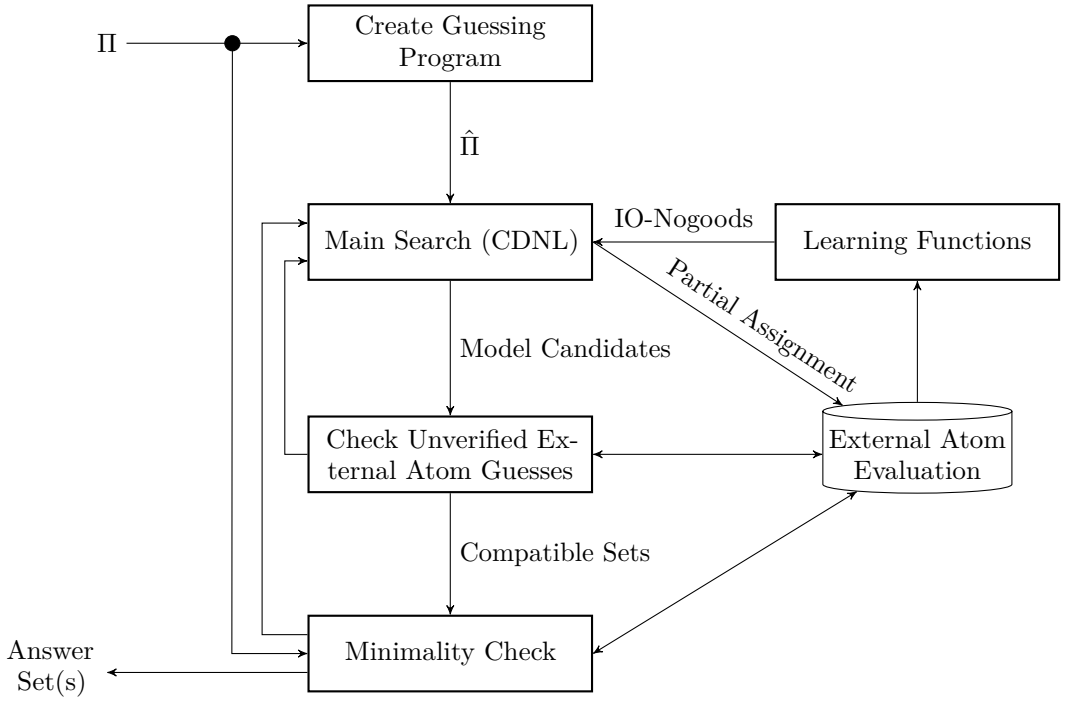

Figure 3.1: Illustration of the workflow of Algorithm 3.1 (adapted from (Redl, 2014))

First, the added **if**-block in Part (c) eliminates those answer sets of $\hat{\Pi}$ which are either not compatible sets of Π, or not minimal models w.r.t. $f\Pi^{\mathbf{A}}$. The remaining answer sets of $\hat{\Pi}$ projected to the atoms in Π are exactly the answer sets of Π (cf. Definition 2.4). Thus, the algorithm with the added **if**-block in Part (c) but without the addition of Part (e) has exactly the desired behavior.

Second, the addition of Part (e) is only an optimization and we need to justify that it does not eliminate answer sets of Π. But this follows from the correctness of $\Lambda(\cdot,\cdot)$, which implies that assignments forbidden by such nogoods would be incompatible with the external sources anyway; therefore they cannot be compatible sets and also not answer sets. □

Algorithm HEX-CDNL describes the schematic backbone of concrete incarnations that are obtained by choosing particular learning functions and heuristics for driving the learning process under partial assignment evaluation. Furthermore, different procedures for the unfounded set check might be used; we shall deal with these aspects in the next Section as well as Chapter 4.

3.3 Nogood Learning with Partial Assignments

In this section, we discuss the generation of nogoods which partially encode the semantics of external atoms. In contrast to previous work on external behavior learning, this

generation however will work for partial assignments in general, and not only for complete assignments. When certain ground instances of an external atom can already be decided, nogoods can be learned early on, and incompatible assignments can be identified; thus, they can guide the solver. Intuitively, nogoods learned based on partial assignments are preferable as they are usually smaller and cut incomplete assignments.

3.3.1 Three-Valued Learning Functions

Let us first assume that we have no further knowledge about external sources and can only observe their (partial) output under a given (possibly partial) input. We introduce a three-valued learning function for the general case, which is a lifting of the respective two-valued learning function defined by Eiter et al. (2012).

Definition 3.5 (Faithful Input-Output Nogood)**.** *An* input-output *(*io-*)*nogood *is any nogood of the form*

$$N = \{\sigma_1 a_1, \ldots, \sigma_n a_n\} \cup \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\} \text{ where } \sigma_1, \ldots, \sigma_{n+1} \in \{\mathbf{T}, \mathbf{F}\};$$

we let $N_I = \{\sigma_1 a_1, \ldots, \sigma_n a_n\}$ *be the literals over ordinary atoms (called the* input part*),* $N_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ *be the replacement atom (called the* output part*) of* N*, and* $\sigma(N_O) = \sigma_{n+1}$*. We call* N faithful*, if* $f_{\&e}(\mathbf{A}, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ *for all partial assignments* $\mathbf{A} \supseteq N_I$*, i.e., it resembles the semantics of the external source.*

We note the following property.

Proposition 3.2. *If* N *is a faithful io-nogood such that* $N_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$*, then* N *is correct w.r.t. all programs* Π *that use* $e_{\&g[\vec{p}]}(\vec{c})$*.*

Proof. Consider a faithful io-nogood $N = \{\sigma_1 a_1, \ldots, \sigma_n a_n\} \cup \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$. Then faithfulness implies $f_{\&e}(N_I, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$. Suppose an assignment $\mathbf{A}$ violates N. Then $\mathbf{A} \supseteq N_I$ and thus $f_{\&e}(\mathbf{A}, \vec{p}, \vec{c}) = \overline{\sigma(N_O)} = \overline{\sigma_{n+1}}$. However, since $\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}$, it follows that $\mathbf{A}$ cannot be a compatible set of any program. □

As for the converse, correct nogoods w.r.t. a given program Π may be io-nogoods that are not faithful, or simply even no io-nogoods. In particular, for inconsistent ordinary ASP-programs, any io-nogood is trivially correct as there are no compatible sets which could be wrongly eliminated, but e.g. the empty nogood is not an io-nogood.

When the oracle of an external atom is evaluated, the solver can create a new nogood for the observed input-output relationship. That is, evaluating $\&g[\vec{p}]$ for a partial assignment $\mathbf{A}$, the solver learns, given all true and false literals of input predicates, whether the output contains $\vec{c}$, where $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq \mathbf{U}$. Note that, since we only consider ground HEX-programs Π, for any partial assignment $\mathbf{A}$ and input list $\vec{p}$ there can only be finitely many tuples $\vec{c}$ where $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ such that $\vec{c}$ occurs in a given program Π. Hence, we only need to consider a fixed number of potential output tuples, which we call a *scope* $\mathcal{S}$ of output tuples. In general, the scope may simply contain all output tuples over the given finite vocabulary.

Definition 3.6 (Input-Output Learning Function)**.** *The learning function for a ge-predicate* $\&g[\vec{p}]$ *under partial assignment* $\mathbf{A}$ *and scope* $\mathcal{S}$ *is*

$$\Lambda_u(\&g[\vec{p}], \mathbf{A}) = \{\mathbf{A}' \cup \{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\} \mid f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma \neq \mathbf{U}, \vec{c} \in \mathcal{S}\},$$

where $\mathbf{A}' = \{\sigma' p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p}, \sigma' \neq \mathbf{U}\}$ *is the relevant part of the external atom input.*

Each respective nogood is an io-nogood by construction and as we have that the oracle is assignment-monotonic, also faithful. Hence:

Proposition 3.3. *Let* $\&g[\vec{p}](\cdot)$ *be an external atom in a* HEX*-program* Π*. Then for all assignments* $\mathbf{A}$*, the nogoods* $\Lambda_u(\&g[\vec{p}], \mathbf{A})$ *in Definition 3.6 are correct w.r.t.* Π*.*

Proof. The added nogood for an output tuple $\vec{c}$ such that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma$, $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, is $\{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\} \cup \{\sigma' p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p}, \sigma' \neq \mathbf{U}\}$. If the nogood is violated by an assignment $\mathbf{A}'$, then the guess for $e_{\&g[\vec{p}]}(\vec{c})$ w.r.t. $\mathbf{A}'$ was wrong as the replacement atom is guessed false (resp. true) but the tuple $\vec{c}$ is in the output (resp. not in the output). Hence, the assignment $\mathbf{A}'$ is not compatible and cannot be extended to a compatible set anyway. □

Example 3.2 (cont'd). Assume we are given for the graph guessing program Π from Example 2.3 in Section 2.2.1 the partial assignment $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{U}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{U}n\_edge(b,a)\}$. In this case, the nogood learning function $\Lambda_u(\&geq[edge, 2], \mathbf{A})$ yields the single io-nogood $\{\mathbf{F}edge(a,b), \mathbf{T}e_{\&geq[edge,2]}()\}$, which is indeed faithful. On the other side, for $\mathbf{A}' = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{T}edge(a,b), \mathbf{U}edge(b,a), \mathbf{F}n\_edge(a,b), \mathbf{U}n\_edge(b,a)\}$, we find that no io-nogood can be learned and $\Lambda_u(\&geq[edge, 2], \mathbf{A}')$ returns $\emptyset$ as $f'_{\&geq}(\mathbf{A}', edge, 2) = \mathbf{U}$, where $f'_{\&geq}$ is the three-valued assignment-monotonic oracle functions as in Example 3.1. △

3.3.2 Exploiting External Source Properties

According to Eiter et al. (2012), given a ge-predicate $\&g[\vec{p}]$ and a complete assignment $\mathbf{A}$, an input parameter $p_i \in \vec{p}$ is *monotonic*, if $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ implies that $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$ for every assignment $\mathbf{A}' \geq \mathbf{A}$ that augments $\mathbf{A}$ only by atoms with predicate p_i. We can refine a three-valued learning function Λ_u similar to two-valued learning functions, cf. Eiter et al. (2012), and tailor it to external sources with specific properties. Here, we show this for external atoms which are monotonic in an input predicate p_i, i.e., the value of the external atom cannot switch from true to false if more atoms over p_i become true and, conversely, it cannot switch from false to true if more atoms over p_i become false. Accordingly, literals of the form $\mathbf{F}p_i(\vec{c}')$ may be dropped from io-nogoods containing $\mathbf{F}e_{\&g[\vec{p}]}(\vec{c})$, and literals of the form $\mathbf{T}p_i(\vec{c}')$ may be dropped from io-nogoods containing $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$. Thus, by exploiting monotonic behavior of oracle functions we are able to obtain smaller, i.e. more general, io-nogoods than by using the general learning function Λ_u.

Definition 3.7 (Monotonic Learning Function)**.** *The learning function for a ge-predicate* $\&g[\vec{p}]$ *that is monotonic in* $\vec{p_m} \subseteq \vec{p}$*, under a partial assignment* $\mathbf{A}$ *and a scope* $\mathcal{S}$*, yields*

$$\Lambda_{mu}(\&g[\vec{p}], \mathbf{A}) = \{\mathbf{A}'_\sigma \cup \{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})} \mid f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma \neq \mathbf{U}, \vec{c} \in \mathcal{S}\},$$

where $\mathbf{A}'_\sigma = \{\sigma' p(\vec{c'}) \in \mathbf{A} \mid p \in \vec{p}, p \notin \vec{p_m}, \sigma' \neq \mathbf{U}\} \cup \{\sigma p(\vec{c'}) \in \mathbf{A} \mid p \in \vec{p_m}\}$.

As before, we can also show that nogoods learned by means of the learning function Λ_{mu} are not violated by any compatible set:

Proposition 3.4. *Let* $\&g[\vec{p}](\cdot)$ *be an external atom in a* HEX*-program* Π*. Then for all assignments* $\mathbf{A}$*, the nogoods* $\Lambda_{mu}(\&g[\vec{p}], \mathbf{A})$ *in Definition 3.7 are correct w.r.t.* Π.

Proof. The added nogood for an output tuple $\vec{c}$ such that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma$, $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, is $\{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\} \cup \{\sigma' p(\vec{c'}) \in \mathbf{A} \mid p \in \vec{p}, p \notin \vec{p_m}, \sigma' \neq \mathbf{U}\} \cup \{\sigma p(\vec{c'}) \in \mathbf{A} \mid p \in \vec{p_m}\}$. If the nogood is violated by an assignment $\mathbf{A}'$, then the guess for $e_{\&g[\vec{p}]}(\vec{c})$ w.r.t. $\mathbf{A}'$ was wrong as the replacement atom is guessed false (resp. true) but the tuple $\vec{c}$ is in the output (resp. not in the output). The previous holds despite the fact that literals of form $\mathbf{T}p(\vec{c})$ (resp. $\mathbf{F}p(\vec{c})$), where $p \in \vec{p_m}$, are omitted from io-nogoods implying a false (resp. true) evaluation of the oracle function because $\{\mathbf{T}p(\vec{c}) \mid \mathbf{T}p(\vec{c}) \in \mathbf{A}'\} \geq \{\mathbf{T}p(\vec{c}) \mid \mathbf{T}p(\vec{c}) \in \mathbf{A}\}$ (resp. $\{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}'\} \supseteq \{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}\}$) must hold for all $p \in \vec{p_m}$, and we have that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ implies that $f_{\&g}(\mathbf{A}'', \vec{p}, \vec{c}) = \mathbf{T}$ for every $\mathbf{A}'' \geq \mathbf{A}$ (resp. that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ implies that $f_{\&g}(\mathbf{A}'', \vec{p}, \vec{c}) = \mathbf{F}$ for every $\mathbf{A}''$ s.t. $\{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}''\} \supseteq \{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}\}$), due to the definition of monotonic input parameters. Hence, the assignment $\mathbf{A}'$ is not compatible and cannot be a compatible set anyway. □

Example 3.3 (cont'd). Consider again program Π from Example 2.3 in Section 2.2.1 and the partial assignment $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{T}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{U}n\_edge(b,a)\}$. When employing the function Λ_u, we obtain $\Lambda_u(\&geq[edge, 2], \mathbf{A}) = \{\{\mathbf{F}edge(a,b), \mathbf{T}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\}\}$. However, when employing Λ_{mu}, we obtain $\Lambda_{mu}(\&geq[edge, 2], \mathbf{A}) = \{\{\mathbf{F}edge(a,b), \mathbf{T}e_{\&geq[edge,2]}()\}\}$ by exploiting monotonicity of the input parameter *edge*. △

The learning functions Λ_u and Λ_{mu} generate nogoods depending on the oracle function given a certain input. However, an external source provider usually knows the source semantics better and can thus provide better nogoods. The latter might include only the necessary atoms in the input; they are thus smaller and prune more of the search space. In such cases, it makes sense to provide custom learning functions $\Lambda_l(\&g[\vec{p}], \mathbf{A})$ which generate for $\&g[\vec{p}]$ and a (possibly partial) assignment $\mathbf{A}$ a set of nogoods.

3.4 Nogood Minimization

In this section, we discuss a second way to exploit partial assignments for more effective learning of io-nogoods based on three-valued oracle functions. Instead of calling three-valued oracle functions with partial input assignments that are generated during solving, a new partial assignment $\mathbf{A}'$ can be obtained from a given assignment $\mathbf{A}$, where $\mathbf{A} \succeq \mathbf{A}'$, by

changing part of the truth values of literals in $\mathbf{A}$ from $\mathbf{T}$ or $\mathbf{F}$ to $\mathbf{U}$. Then, a three-valued oracle function can be called with the resulting assignment $\mathbf{A}'$ in order to detect truth assignments in $\mathbf{A}$ which are irrelevant for the evaluation of the respective external source.

By employing this strategy, we eliminate redundant (input) literals from the nogoods in Λ_u and Λ_{mu}, while faithfulness of io-nogoods is retained (relying on assignment-monotonicity of three-valued oracle functions). Recall that io-nogoods do not contain any literals which are unassigned. Hence, we can obtain smaller and thus, more general io-nogoods, which potentially prune larger parts of the search space. For this purpose, we introduce two new algorithms for computing minimal io-nogoods, i.e, nogoods from which no literal in the input part can be removed without changing the output value of the respective oracle function to unassigned. Moreover, we show that minimization and theory-specific learning are in fact closely related.

Definition 3.8 (Io-Nogood Minimization)**.** *Given a faithful io-nogood N with $N_O = \{\sigma e_{\&g[\vec{p}]}(\vec{c})\}$, the set of minimized nogoods of N is*

$$minimize_{\&g[\vec{p}]}(N) = \{N' \subseteq N \mid N' \text{ is a faithful io-nogood}, f_{\&g}(N'', \vec{p}, \vec{c}){=}\mathbf{U} \text{ for all } N'' \subsetneq N'_I\}.$$

This extends to sets S of nogoods by $minimize_{\&g[\vec{p}]}(S) = \bigcup_{N \in S} minimize_{\&g[\vec{p}]}(N)$. Note that exponentially many minimal nogoods in the size of N are possible. In the following, we omit the subscript $\&g[\vec{p}]$ if the ge-predicate is clear from the context and just write $minimize(N)$.

Example 3.4 (cont'd). Consider the assignment $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{T}n\_edge(b,a)\}$ together with the learned faithful io-nogood $N = \{\mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\} \in \Lambda_u(\&geq[edge,2], \mathbf{A})$. According to Definition 3.8, we obtain $minimize(N) = \{\{\mathbf{F}edge(a,b), \mathbf{T}e_{\&geq[edge,2]}()\}, \{\mathbf{F}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\}\}$. △

The minimized nogoods subsume all faithful io-nogoods.

Proposition 3.5. *Let $\mathbf{A}$ be a partial assignment and N be a faithful io-nogood for $\&g[\vec{p}]$ over the atoms in $\mathbf{A}$. Then some $N' \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ exists such that $N' \subseteq N$.*

Proof. The nogood N can be reduced to a subset-minimal set M such that M is a faithful io-nogood but $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all N'' with $N''_I \subsetneq M_I$, $N''_O = M_O$. We observe that $M \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ and $M \subseteq N$. □

As a subset of each faithful io-nogood occurs among all minimized nogoods, no further faithful io-nogoods prune the search space more effectively. Still, there might be further correct nogoods (non-io ones and/or depending on the program).

Definition 3.9 (Io-Complete and Partial Learning Functions)**.** *A theory-specific learning function $\Lambda_l(\cdot,\cdot)$ is* io-complete *for an external source $\&g$, if for every partial assignment $\mathbf{A}' \subseteq \mathbf{A}$ and input list $\vec{p}$, it holds that $\Lambda_l(\&g[\vec{p}], \mathbf{A})$ is the least set that contains $\mathbf{A}' \cup \{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\}$ for every output list $\vec{c}$ such that $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \sigma \in \{\mathbf{T}, \mathbf{F}\}$; otherwise, the learning function $\Lambda_l(\cdot,\cdot)$ is* partial.

That is, an io-complete theory-specific learning function Λ_l learns *all and only* io-nogoods with a premise over the current partial assignment which resemble the semantics of $\&g$.

As it turns out, learning using io-complete theory-specific learning functions and nogood minimization are closely related. Let $min_{\subseteq}(S) = \{N \in S \mid \nexists N' \in S \text{ s.t. } N' \subsetneq N\}$ be the restriction of S to subset-minimal nogoods.[1] Then:

Proposition 3.6. *Let Λ_l be an io-complete theory-specific learning function for an external source $\&g$. Then, for all partial assignments $\mathbf{A}$ and input lists $\vec{p}$ we have $minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A})) = min_{\subseteq}(\Lambda_l(\&g[\vec{p}], \mathbf{A}))$.*

Proof. Let $\mathbf{A}$ be a partial assignment and let $\vec{p}$ be an input list.

($\Rightarrow$) Let $N \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ be an io-nogood learned from Λ_u after minimization. Since $f_{\&g}(N_I, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ by faithfulness, it follows from completeness of Λ_l that $N \in \Lambda_l(\&g[\vec{p}], \mathbf{A})$. Moreover, since N is minimal, it follows that $f_{\&g}(N'_I, \vec{p}, \vec{c}) = \mathbf{U}$ for all $N' = N'_I \cup N_O$ with $N'_I \subsetneq N_I$. Therefore, there can be no $N' \subsetneq N$ with $N' \in \Lambda_l(\&g[\vec{p}], \mathbf{A})$, thus $N \in min_{\subseteq}(\Lambda_l(\&g[\vec{p}]))$.

($\Leftarrow$) Let $N \in min_{\subseteq}(\Lambda_l(\&g[\vec{p}], \mathbf{A}))$ be a subset-minimal nogood learned from Λ_l. Since $f_{\&g}(N_I, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ (due to faithfulness) we have $N \in \Lambda_u(\&g[\vec{p}], \mathbf{A})$ by definition of Λ_u. Moreover, since N is subset-minimal among the nogoods $\Lambda_l(\&g[\vec{p}], \mathbf{A})$ and Λ_l is io-complete, we have that $f_{\&g}(N'_I, \vec{p}, \vec{c}) = \mathbf{U}$ for all $N' = N'_I \cup \{\sigma(N_O)e_{\&g[\vec{p}]}(\vec{c})\}$. But then no atom from N can be removed as by Definition 3.8, thus $N \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$. □

This proposition implies that we have alternative techniques to learn all io-nogoods that prune the search space in an optimal way (cf. Proposition 3.5). As above, it considers only *faithful io-nogoods* while further correct nogoods may exist. Notably, the equality holds only under the premises of *exhaustive* minimization in the first case and an *io-complete* theory-specific learning function in the second; otherwise, different sets of nogoods may be produced. As both operations are expensive and impractical, it makes sense to support both (incomplete) minimization and (incomplete) theory-specific learning functions.

3.4.1 Sequential Nogood Minimization

In practice, we use Algorithm 3.2 to compute only one minimal io-nogood for each learned io-nogood. Instead of minimizing each nogood separately and to avoid redundant queries, we proceed in parallel and use a cache for the external atom output of a set S of io-nogoods with identical input but different outputs. The algorithm works by sequentially removing the same literal simultaneously from the premises of all N in S in Part (a), and checking whether the output for the resulting premises is already in the cache, in Part (b). If not, all outputs $\vec{c}'$ s.t. $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}') \neq \mathbf{U}$ are computed (this is a

[1] Despite similar names, *minimize* differs from $min_{\subseteq}$ as it minimizes w.r.t. oracle results while $min_{\subseteq}$ just selects the minimal sets.

Algorithm 3.2: Simultaneous Nogood Minimization

Input: A set S of faithful io-nogoods N with $N_I = \{\sigma_1 a_1, \ldots, \sigma_n a_n\}$
Output: A set of minimal faithful io-nogoods

$ch \leftarrow \emptyset$ `// cache for oracle calls`
for *each signed literal* $\sigma_i a_i \in N_I$ **do**
(a) **for** *each io-nogood* $N' \in S$ *with* $N'_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ **do**
$N^s \leftarrow N'_I \setminus \{\sigma_i a_i\}$ `// smaller oracle input`
(b) **if** $\langle N^s, \cdot \rangle \notin ch$ **then**
$ch \leftarrow ch \cup \{\langle N^s, \{\sigma e_{\&g[\vec{p}]}(\vec{c'}) \mid f_{\&g}(N^s, \vec{p}, \vec{c'}) = \sigma \neq \mathbf{U}\}\rangle\}$
end
(c) **if** $\overline{\sigma_{n+1}} e_{\&g[\vec{p}]}(\vec{c}) \in$ *output for* $\langle N^s, output\rangle \in ch$ **then**
Replace N' by $N^s \cup \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ in S
end
end
end
return S

single call in the implementation) and stored in the cache. Otherwise, no external source call is needed. It is then checked if the resulting nogood is still faithful in Part (c), and N is replaced by its reduced equivalent in S in this case. Formally:

Proposition 3.7. *For a set S of faithful io-nogoods with equal input parts and distinct output parts, Algorithm 3.2 yields exactly one faithful io-nogood $N' \in minimize(N)$ for each $N \in S$.*

Proof. Let S be a set of faithful io-nogoods with identical input parts and distinct output parts. To distinguish between the input and the output of Algorithm 3.2, we denote by S^o the altered set which is returned by the algorithm given input S.

First of all, S is only manipulated in Part (c) by replacing the input part of nogoods in S. Hence, it holds that $|S| = |S^o|$. Moreover, all $N \in S$ have different output parts which are not changed in Part (c). As a result, after every replacement exactly one element in the resulting set S' can be associated with each nogood in the initial set S. This proves that each input has a corresponding output nogood. It remains to show that these are minimal faithful io-nogoods.

Let $N \in S$ and $N^o \in S^o$ be the corresponding output nogood, i.e. $N_O = N^o_O = \sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})$. We have to show that $N^o \in minimize(N)$. According to Definition 3.8, this means we have to show that $N^o \subseteq N$, that N^o is a faithful io-nogood, and that $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all $N'' \subsetneq N^o_I$. Clearly, it holds that $N^o \subseteq N$ since elements are only *removed* from N_I by Algorithm 3.2.

Next, we prove that N^o is a faithful io-nogood by showing that faithful io-nogoods in S are only replaced by faithful io-nogoods, in Algorithm 3.2. Let S' be an arbitrary state of S during the execution of Algorithm 3.2, and $N' \in S'$ a faithful io-nogood. We need to show that $N^s \cup N'_O$ is also a faithful io-nogood for $N^s = N'_I \setminus \sigma_i a_i$ where $\sigma_i a_i \in N'_I$. Since $N^s \subset N'_I$, it holds that $N^s \cup N'_O$ is an io-nogood. In Part (c), N'_I

is replaced by N^s in S only if $\overline{N'_O} \in \mathit{output}$ for $\langle N^s, \mathit{output}\rangle \in \mathit{ch}$, which is the case only if $f_{\&g}(N^s, \vec{p}, \vec{c}) = \overline{\sigma(N'_O)} \neq \mathbf{U}$, due to Part (b) of the algorithm. Note that it is ensured in Part (b) that for N^s there is exactly one $\langle N^s, \mathit{output}\rangle \in \mathit{ch}$. Further, we know that $\overline{\sigma(N'_O)} \neq \mathbf{U}$ as N' is an io-nogood. Due to assignment-monotonicity of $f_{\&g}$, we have that $f_{\&g}(N^s, \vec{p}, \vec{c}) = X$, $X \in \{\mathbf{T}, \mathbf{F}\}$, implies $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = X$ for all partial assignments $\mathbf{A} \succeq N^s$, by Definition 3.3. We derive that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \overline{\sigma(N'_O)}$ for all partial assignments $\mathbf{A} \supseteq N^s$ and thus, that $N^s \cup N'_O$ is in fact a faithful io-nogood. Since we know that N is a faithful io-nogood, we conclude that N^o is a faithful io-nogood as well.

Finally, we prove that $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all $N'' \subsetneq N^o_I$. Assume to the contrary that we have $f_{\&g}(N'', \vec{p}, \vec{c}) \neq \mathbf{U}$ for some $N'' \subsetneq N^o_I$, and let $\sigma_i a_i \in N^o_I \setminus N''$. Since $\sigma_i a_i \in N^o_I$, we have that $N^s = N'_I \setminus \sigma_i a_i$ holds for some N' that is chosen during the execution of Algorithm 3.2 in Part (a) (i.e. in the iteration when it is tried to obtain smaller nogoods by removing $\sigma_i a_i$), with $N'_O = \sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})$ and $N'_I \subseteq N^o_I$. As we have that $\sigma_i a_i \in N^o_I$, we derive that $f_{\&g}(N^s, \vec{p}, \vec{c}) = \mathbf{U}$. Otherwise N' would be replaced by $N^s \cup N'_O$ in Part (c), and we would obtain $\sigma_i a_i \notin N^o_I$. However, we obtain that $f_{\&g}(N'', \vec{p}, \vec{c}) \neq \mathbf{U}$, $f_{\&g}(N^s, \vec{p}, \vec{c}) = \mathbf{U}$ and $N^s \supset N''$, which together contradicts that $f_{\&g}$ is assignment-monotonic. This proves that indeed $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all $N'' \subsetneq N^o_I$ and thus, $N^o \in \mathit{minimize}(N)$. □

3.4.2 Divide-and-Conquer Strategy for Nogood Minimization

Even when io-nogoods with the same input parts are minimized simultaneously, removing each literal from the respective input separately and checking the output of the corresponding oracle function may result in a large number of external calls, which directly depends on the length of the input part. In cases where io-nogoods are large or the external evaluation requires a lot of time, the additional computational effort required for nogood minimization may outweigh the positive effect of obtaining smaller nogoods, or even make minimization infeasible. While in the worst case, i.e. when an io-nogood is already minimal, this situation cannot be improved, it can be more efficient to remove several literals from a nogood at once before evaluating the external source when the input part contains many irrelevant literals.

The QuickXplain algorithm, which has been introduced by Junker (2004) for efficiently computing minimal conflict sets in the context of constraint programming, can be used for this purpose as an alternative algorithm for minimizing io-nogoods. It implements a divide-and-conquer strategy producing a binary search tree, and can be employed for computing a minimal nogood from a given io-nogood more efficiently if the nogood contains many irrelevant literals, especially when the input part of the io-nogood is large. In this way, given an io-nogood with input part of size n, instead of n calls to the oracle function, only $O(log_2 n)$ external calls are required. However, in the worst case, i.e. when no literal can be removed from a given nogood, $O(n)$ calls are necessary. Consequently, the algorithm can behave either better or worse than a sequential algorithm, depending on the properties of the io-nogoods that are minimized.

Algorithm 3.3: QuickXplain Nogood Minimization

Input: A faithful io-nogood $N = \{\sigma_1 a_1, \ldots, \sigma_n a_n, \sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$
Output: A minimal faithful io-nogood $N' \in \mathit{minimize}(N)$

(a) **if** $f_{\&g}(\emptyset, \vec{p}, \vec{c}) = \overline{\sigma_{n+1}}$ **then return** $\{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$
(b) **return** $\mathit{quickXplain}(\emptyset, \emptyset, N_I) \cup \{\sigma_{n+1}\, e_{\&g[\vec{p}]}(\vec{c})\}$

function $\mathit{quickXplain}(B, D, N')$
(c) **if** $D \neq \emptyset$ *and* $f_{\&g}(B, \vec{p}, \vec{c}) = \overline{\sigma_{n+1}}$ **then return** $\emptyset$
(d) **if** $|N'| = 1$ **then return** N'
(e) Partition N' into two non-empty sets N_1 and N_2
$D_1 \leftarrow \mathit{quickXplain}(B \cup N_2, N_2, N_1)$
$D_2 \leftarrow \mathit{quickXplain}(B \cup D_1, D_1, N_2)$
return $D_1 \cup D_2$

Algorithm 3.3 is a variant of the QuickXplain-algorithm by Shchekotykhin et al. (2015), adapted to our specific setting of io-nogood minimization. The algorithm receives a faithful io-nogood $N = \{\sigma_1 a_1, \ldots, \sigma_n a_n, \sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ and first checks whether the literal in the output part N_O depends on a non-empty input part N_I, in Part (a). Subsequently, a recursive function is called in Part (b), which during its execution checks if different subsets of N_I imply the same external replacement literal in N_O as N_I.

The first argument B of the function $\mathit{quickXplain}(B, D, N')$ contains the current subset of the input part N_I w.r.t. which the oracle function $f_{\&g}(B, \vec{p}, \vec{c})$ is evaluated in Part (c). The second argument D indicates if the oracle function needs to be evaluated for a given B, which is only the case if D is non-empty as only then B has changed since the last external evaluation. If B is determined to imply the same truth value for $e_{\&g[\vec{p}]}(\vec{c})$ as N_I in Part (c), no further literals from N' need to be added to B and thus, the empty set is returned. In case the subset N' of the input part N_I of literals that can still be added to B to obtain the correct value for $f_{\&g}(B, \vec{p}, \vec{c})$ is a singleton, it is returned in Part (d). Finally, in Part (e), the provided subset N' of the input part N_I is partitioned into two nonempty sets N_1 and N_2, and the function is called recursively, once for each partition N_1 and N_2 of N'. The result D_1 of the first call, where N_1 is provided as new subset of the input part N_I, contains all literals from N_1 that need to be added to $B \cup N_2$ such that the oracle function still evaluates to σ_{n+1}. Similarly, all literals from N_I that need to be added to $B \cup D_1$ such that the oracle function still evaluates to σ_{n+1} are stored in D_2. As a result, no literal can be removed from D_1 or D_2 such that $f_{\&g}(B \cup D_1 \cup D_2, \vec{p}, \vec{c}) = \sigma$ still holds, and the input part $D_1 \cup D_2$, which together with $\{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ yields a minimal io-nogood, is returned.

The computation of a minimal io-nogood by Algorithm 3.3 is illustrated by the following example.

Example 3.5. Reconsider $f'_{\&geq}$ from Example 3.1 and the faithful io-nogood $N = \{\mathbf{F}edge(a,a),\ \mathbf{T}edge(b,b), \mathbf{T}edge(a,b),\ \mathbf{T}edge(b,a),\ \mathbf{F}e_{\&geq[edge,2]}()\}$. When Algorithm 3.3 is executed with input N, the function call $\mathit{quickXplain}(\emptyset, \emptyset, N_I)$ is performed with $N_I = \{\mathbf{F}edge(a,a),\ \mathbf{T}edge(b,b), \mathbf{T}edge(a,b),\ \mathbf{T}edge(b,a)\}$. Since $D = \emptyset$ and

$|N| \neq 1$ hold w.r.t. the first call, $N' = N_I$ is partitioned into two sets, for example $N_1 = \{\mathbf{F}edge(a,a), \mathbf{T}edge(b,b)\}$ and $N_2 = \{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a)\}$.

Subsequently, the first recursive call of the function *quickXplain* returns $\emptyset$, which is assigned to D_1, as $f'_{\&geq}(N_2, edge, 2) = \mathbf{T}$, i.e. $\{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}e_{\&geq[edge,2]}()\} \subset N$ is still a faithful io-nogood. Accordingly, the second recursive call in Part (e) corresponds to the call $quickXplain(\emptyset, \emptyset, N_2)$, in which N_2 is partitioned again into $\{\mathbf{T}edge(a,b)\}$ and $\{\mathbf{T}edge(b,a)\}$. Because each of the sets has cardinality 1 but none of them suffices to derive $\&geq[edge,2]()$, N_1 and N_2 are returned from the two recursive calls in Part (e), respectively. Thus, $N_1 \cup N_2$ is returned by the second recursive call in the outer function call, which is assigned to D_2. Consequently, $\emptyset \cup \{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a)\}$ is returned by the function called in Part (b). Finally, the minimal io-nogood $\{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}e_{\&geq[edge,2]}()\}$ is returned by Algorithm 3.3. △

Algorithm 3.3 always finds a minimal faithful io-nogood:

Proposition 3.8. *Given a faithful io-nogood N, Algorithm 3.3 terminates and returns exactly one faithful io-nogood $N' \in minimize(N)$.*

Proof. The statement follows directly from Proposition 3.6 and Theorem 1 by Junker (2004). □

Like Algorithm 3.2, Algorithm 3.3 returns exactly one minimal io-nogood for a given input. A straightforward way to obtain multiple minimal io-nogoods consists in re-running Algorithm 3.3 with different partition heuristics in Part (e); and every minimal io-nogood can be obtained in this way.

3.5 Empirical Evaluation

In this section, we present the results of an experimental evaluation of our techniques.

3.5.1 Experimental Setup

We integrated the techniques for partial evaluation during HEX-solving that have been developed in this chapter into DLVHEX 2.5.0 with GRINGO 4.4.0 and CLASP 3.1.1 as backends.

We remark that although CLINGO 5 is known for its theory solving capabilities (Gebser et al., 2016), also previous versions of GRINGO resp. CLASP had similar features, which are exploited by DLVHEX; CLINGO 5 makes these features more accessible.

Note that an upgrade of our backend to CLINGO 5, which is currently available in version 5.4[2] and supports a large range of modern ASP solving techniques such as *multi-shot ASP solving* (Gebser, Kaminski, Kaufmann, & Schaub, 2019) and custom heuristics, will mainly simplify the interfaces, but will not directly enable algorithmic improvements.

[2] https://github.com/potassco/clingo

Hence, there is no interference with the techniques presented in this chapter. For a more detailed discussion of the differences to clingo 5 we refer to Section 3.6.

In the following we first describe the platform used for carrying out our benchmarks and the configurations we are going to compare. We then describe the benchmark suite used for the evaluation. All instances and log files of the experiments can be found at `http://www.kr.tuwien.ac.at/research/projects/inthex/partialeval`.

Evaluation Platform

All benchmarks were run on a Linux machine with two 12-core AMD Opteron 6238 SE CPUs and 512 GB RAM; the timeout was 300 seconds and the memout was 8 GB per instance. We used the *HTCondor* load distribution system (HTCondor Website, 2018) to ensure robust running times (i.e., deviations of runs on the same instance are negligible). The average running time of 50 instances per problem size is reported (in seconds) for computing all answer sets respectively the first answer set; the number of timeouts is shown in parentheses and furthermore, the average number of solutions of the instances is given in the tables, where '$\geq$' respects timeouts.

Benchmark Configurations

Naturally, there is a tradeoff between the information that can be gained from additional external evaluations under partial assignments during solving, and the running time that has to be invested for the respective external calls. For this reason, we used different heuristics for controlling the number of external evaluations, and investigated the impact of 8 different solver configurations.

Initially, we tested three heuristics for additional external source calls during the main search for compatible sets (cf. Algorithm 3.1, Part (e)) without nogood minimization, namely

- **never**: external atoms are only evaluated w.r.t. *candidate models* (representing the standard configuration of dlvhex without the new techniques);
- **periodic**: external atoms are evaluated w.r.t. *partial assignments* only after every 10^{th} solver guess during the model search; and
- **always**: external atoms are evaluated w.r.t. *partial assignments* after every solver guess during the model search.

We then tested nogood minimization instead of additional calls (i.e., only for complete assignments), where we used the algorithm for simultaneous nogood minimization (cf. Algorithm 3.2) and the QuickXplain algorithm (cf. Algorithm 3.3), respectively, for minimizing either

- *all* nogoods in conditions **ngm** and **qxp**, respectively, or
- the currently *conflicting* ones, i.e. those which violate the current solver assignment and trigger backjumping, in conditions **ngm-c** and **qxp-c**.

For benchmarks where external atoms have output values, we also compared simultaneous minimization with sequential minimization (**ngm-sq**), i.e. minimizing each io-nogood separately. We omit results for minimization combined with **periodic** or **always**, as this was always significantly slower than some other configuration (due to many more external calls with little gain).

Benchmark Problems

We considered encodings of three different problems in the evaluation:

- *Pseudo-boolean (PB-)problems*, also known as *0-1 integer linear programs*, representing linear constraints over Boolean variables, which are among Karp's famous 21 NP-complete problems (Karp, 1972).
- Assignment of taxi drivers to customers under constraints, where queries to an external ontology, expressed in the lightweight *description logic* (*DL*) *DL-Lite*, are made via external atoms to find out locations and classify customers and drivers (*taxi assignment with ontology access*) (Eiter, Fink, Redl, & Stepanova, 2014; Eiter, Fink, & Stepanova, 2016). Note that despite a similar scenario, our benchmark is different from the one used by Eiter, Fink, Redl, and Stepanova (2014), as it admits multiple solutions due to nondeterministic guessing of customer assignments.
- The well-known *strategic companies* problem (Cadoli, Eiter, & Gottlob, 1997), which is popular with ASP competitions, extended with externally stored conflicts among companies (*conflicting strategic companies*).

The problems have different characteristics with regard to the computational complexity and the external atoms and their usage. While query answering w.r.t. the DL-Lite ontology used in our taxi assignment benchmark is tractable (Calvanese, De Giacomo, Lembo, Lenzerini, & Rosati, 2007), and solving PB-problems is NP-complete, computing strategic companies is located at the second level of the polynomial hierarchy. Moreover, the general learning function Λ_u is used for the PB-problems benchmark. Due to monotonicity of external sources, the learning function Λ_{mu} can be utilized in all other benchmarks. A further difference consists in the fact that external atoms are used to formulate integrity constraints in the PB-problems and the conflicting strategic companies benchmark, while output values are derived in the taxi assignment benchmark.

3.5.2 Hypotheses

We started our investigation with the following hypotheses regarding the employment of partial evaluation in the answer set search:

(H3.1) The heuristics **periodic** and **always** decrease the runtime over **never** if useful information is obtainable by early evaluation with little runtime overhead, and increase it otherwise, whereby the effect is stronger for **always**.

$$trueAt(X) \vee falseAt(X) \leftarrow atom(X).$$
$$\leftarrow \&pbCheck[trueAt, PBInst]().$$

Figure 3.2: Pseudo-Boolean Problems Rules

(H3.2) The heuristics **periodic** performs better than **always** if more running time needs to be invested for each external call, mitigating the tradeoff between information gain and running time invested in additional calls.

(H3.3) The tradeoff between information gain and runtime invested in additional calls can be mitigated even more effectively by just minimizing io-nogoods on complete assignments using **ngm** or **ngm-c** instead of evaluating early.

(H3.4) Using **qxp** or **qxp-c** instead of **ngm** or **ngm-c** decreases the running time when io-nogoods contain many irrelevant literals, but does not increase it significantly otherwise.

3.5.3 Experiments on Partial Evaluation and Nogood Minimization

In this section, we use the three benchmark problems to investigate the effect of partial evaluations during HEX-search using different heuristics. In addition, we compare the results to the running times achieved by employing our new algorithms for nogood minimization.

Pseudo-Boolean Problems

Pseudo-boolean (*PB-*)problems constitute sets of *pseudo-boolean constraints* of the form $C_0p_0 + ... + C_{n-1}p_{n-1} \geq C_n$, where all p_i are literals and all C_i are integers (Roussel & Manquinho, 2009). A solution to a PB-problem P is a truth assignment to the Boolean variables occurring in P such that all inequalities in P are satisfied, where a true literal is interpreted as the value 1 and a false literal as the value 0. Several dedicated PB-solvers have been developed (cf. Manquinho & Silva, 2005), and CLASP can also be employed for efficient PB-problem solving.

Here, however, our goal is not to implement a reasoner for solving PB-problems that can compete with tailored solvers, but to specify external constraints of a HEX-program in the form of PB-problems such that answer sets are restricted to those assignments that also represent solutions to the respective PB-problem. This strict separation of the guess and the check part results in benchmark instances that are well-suited for investigating the effect of a tighter integration of the solving algorithm and the evaluation of external constraints.[3] Moreover, applying an analogous pattern for outsourcing constraints in HEX-programs is a common strategy to avoid the explicit generation of all forbidden combinations of atoms during grounding (Eiter, Redl, & Schüller, 2016).

[3]Note that for the purpose of solving PB-problems as part of a HEX-program (possibly in combination with other external sources), the external source could directly interface a dedicated PB solver.

#	All Answer Sets							av. solutions
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**	
4	**0.13** (0)	**0.13** (0)	0.14 (0)	0.14 (0)	**0.13** (0)	0.14 (0)	0.14 (0)	2.06
8	0.34 (0)	0.33 (0)	**0.22** (0)	0.24 (0)	**0.22** (0)	0.26 (0)	0.23 (0)	4.82
12	4.82 (0)	3.95 (0)	0.80 (0)	0.59 (0)	0.50 (0)	0.60 (0)	**0.46** (0)	10.96
16	280.02 (1)	71.99 (0)	3.28 (0)	1.29 (0)	1.11 (0)	1.23 (0)	**0.94** (0)	12.66
20	300.00 (50)	300.00 (50)	18.87 (0)	2.63 (0)	2.16 (0)	2.41 (0)	**1.62** (0)	24.56
24	300.00 (50)	300.00 (50)	76.75 (1)	4.28 (0)	3.58 (0)	3.69 (0)	**2.42** (0)	32.00
28	300.00 (50)	300.00 (50)	247.06 (32)	9.92 (0)	7.25 (0)	9.48 (0)	**4.61** (0)	82.28
32	300.00 (50)	300.00 (50)	294.05 (47)	20.49 (0)	11.18 (0)	22.03 (1)	**6.94** (0)	269.24
36	300.00 (50)	300.00 (50)	300.00 (50)	36.44 (1)	17.31 (0)	39.27 (3)	**10.28** (0)	519.20
38	300.00 (50)	300.00 (50)	298.99 (49)	38.66 (1)	19.48 (0)	40.86 (2)	**10.90** (0)	451.78
40	300.00 (50)	300.00 (50)	300.00 (50)	37.13 (0)	23.89 (0)	36.04 (0)	**12.70** (0)	233.50

Table 3.1: Results for random PB-problems with 4 to 40 variables (all answer sets)

In our benchmark implementation, we search for solutions to a PB-problem P by guessing an interpretation of the atoms occurring in P utilizing a disjunctive rule, and we restrict the answer sets of the program to solutions of P by employing the external atom $\&pbCheck[trueAt, PBInst]()$ in a program constraint. This results in a simple encoding shown in Figure 3.2, where a fact $atom(a)$ is added for each atom a occurring in P. At this, the variable $PBInst$ is instantiated by a string containing the path to a file encoding the instance P, and the true extension of the predicate $trueAt$ w.r.t. an assignment $\mathbf{A}$ represents those atoms occurring in P that are mapped to true by $\mathbf{A}$. The external atom $\&pbCheck[trueAt, PBInst]()$ evaluates to true w.r.t. a complete assignment $\mathbf{A}$ iff the interpretation of the atoms occurring in P represented by $\mathbf{A}$ constitutes a solution for P. We extend the semantics of the associated evaluation function to partial assignments $\mathbf{A}$ as follows:

$$f_{\&pbCheck'}(\mathbf{A}, trueAt, PBInst) = \begin{cases} \mathbf{T} & \text{if every } C_0p_0 + \ldots + C_{n-1}p_{n-1} \geq C_n \\ & \text{in } P \text{ fulfills } \sum_{\{c \mid \mathbf{T}trueAt(c) \in \mathbf{A}\} \models p_i} C_i \geq C_n; \\ \mathbf{F} & \text{if some } C_0p_0 + \ldots + C_{n-1}p_{n-1} \geq C_n \\ & \text{in } P \text{ fulfills } \sum_{\{c \mid \mathbf{F}trueAt(c) \notin \mathbf{A}\} \models p_i} C_i < C_n; \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

By exploiting this three-valued semantics, inconsistent partial assignments to the atoms occurring in P can be detected earlier. As a result, potentially large parts of the search space can be pruned and the inconsistent partial assignments can be learned in form of io-nogoods to avoid revisiting the same partial assignments subsequently.

First, we tested randomly generated problems with $N \in [4, 40]$ variables and $4 \times N$ PB-constraints with $n = 6$ and $C_i \in [1, 5]$ for $0 \leq i \leq n$. The results are shown in Tables 3.1 and 3.2.

The specific ratio between the number of variables and the number of constraints ensures that only a small fraction of all assignments are answer sets. A clear improvement over **never** is observed whenever external atoms are evaluated early. The configuration **always** shows the best performance, with **periodic** falling in-between **always** and **never**; hence learning the io-behavior of the external source as early as possible outweighs the

#	First Answer Set						
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
4	**0.12** (0)	0.13 (0)	0.13 (0)	0.13 (0)	0.13 (0)	0.13 (0)	0.13 (0)
8	0.23 (0)	0.22 (0)	**0.16** (0)	0.18 (0)	0.18 (0)	0.18 (0)	0.18 (0)
12	2.23 (0)	1.82 (0)	0.35 (0)	0.34 (0)	0.33 (0)	0.31 (0)	**0.30** (0)
16	123.32 (0)	38.42 (0)	1.32 (0)	0.83 (0)	0.81 (0)	0.70 (0)	**0.67** (0)
20	259.72 (42)	237.47 (32)	7.92 (0)	1.59 (0)	1.58 (0)	1.22 (0)	**1.17** (0)
24	294.30 (49)	286.30 (46)	31.13 (0)	2.90 (0)	2.84 (0)	1.96 (0)	**1.89** (0)
28	300.00 (50)	300.00 (50)	96.86 (7)	5.30 (0)	5.26 (0)	3.32 (0)	**3.20** (0)
32	300.00 (50)	300.00 (50)	179.38 (21)	8.05 (0)	8.00 (0)	4.71 (0)	**4.60** (0)
36	300.00 (50)	300.00 (50)	272.92 (42)	12.29 (0)	12.30 (0)	6.65 (0)	**6.58** (0)
38	300.00 (50)	300.00 (50)	264.80 (40)	13.66 (0)	13.76 (0)	7.23 (0)	**7.10** (0)
40	300.00 (50)	300.00 (50)	289.35 (46)	17.26 (0)	17.11 (0)	8.74 (0)	**8.68** (0)

Table 3.2: Results for random PB-problems with 4 to 40 variables (first answer set)

#	All Answer Sets							av. solutions
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**	
2	51.59 (0)	23.31 (0)	**0.13** (0)	0.14 (0)	0.14 (0)	**0.13** (0)	**0.13** (0)	0.00
4	61.01 (0)	29.80 (0)	0.42 (0)	0.32 (0)	0.31 (0)	**0.24** (0)	**0.24** (0)	0.04
6	70.69 (0)	40.36 (0)	9.87 (0)	5.32 (0)	2.21 (0)	7.20 (0)	**2.18** (0)	286.58
8	75.15 (0)	58.03 (0)	66.40 (0)	72.73 (0)	**14.78** (0)	98.86 (0)	15.04 (0)	6178.00
10	78.48 (0)	78.48 (0)	150.24 (0)	191.57 (0)	**43.80** (0)	242.41 (0)	44.13 (0)	18297.76
12	87.54 (0)	98.84 (0)	222.71 (0)	258.52 (1)	**72.83** (0)	282.77 (16)	73.49 (0)	26785.20
14	95.24 (0)	111.57 (0)	267.78 (0)	275.09 (3)	**90.38** (0)	269.99 (7)	91.07 (0)	30629.80
16	103.85 (0)	123.68 (0)	299.38 (37)	281.36 (6)	**103.35** (0)	245.74 (2)	103.38 (0)	32141.44
18	**113.89** (0)	135.02 (0)	300.00 (50)	285.74 (5)	114.60 (0)	221.97 (0)	114.31 (0)	32538.18
20	**122.84** (0)	146.10 (0)	300.00 (50)	294.55 (12)	123.51 (0)	205.68 (0)	123.65 (0)	32685.16

Table 3.3: Results for random PB-problems with PB-constraint length of 2 to 20 (all answer sets)

running time overhead for querying it additionally. When minimizing io-nogoods only after a complete assignment has been generated in condition **ngm**, the overhead of many external calls can be reduced, while similar information can be obtained from them, resulting in much lower running times. Nogood minimization is even more effective when only conflicting nogoods are minimized in condition **ngm-c**. The reason is that in this benchmark, the external atom is only used in a program constraint such that it must evaluate to false w.r.t. any answer set of the program. Accordingly, the truth value of the external atom is never guessed to be true and non-conflicting io-nogoods, i.e. those which imply a false evaluation of the external atom, cannot prune the search space. The conditions **qxp** and **qxp-c** perform better than **ngm** and **ngm-c**, respectively, which is explained by the fact that in this benchmark io-nogoods typically contain many irrelevant literals. Overall, **qxp-c** shows the best performance w.r.t. all instance sizes. Regarding computing the first answer set we observe a similar pattern.

Second, to investigate the behavior when large parts of the search space contain solutions, i.e. when there is less room for pruning it, we fixed the number of variables and PB-constraints to 15 and 60, respectively, and tested different lengths $N \in [2, 20]$. The results are shown in Tables 3.3 and 3.4.

The solution count increases with length, and for $N > 14$ nearly all assignments are

#	First Answer Set						
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
2	51.91 (0)	23.21 (0)	**0.12** (0)	0.14 (0)	0.14 (0)	0.13 (0)	0.13 (0)
4	60.33 (0)	29.36 (0)	0.42 (0)	0.32 (0)	0.31 (0)	**0.24** (0)	**0.24** (0)
6	2.48 (0)	1.87 (0)	**0.31** (0)	0.50 (0)	0.48 (0)	0.48 (0)	0.45 (0)
8	0.20 (0)	**0.18** (0)	**0.18** (0)	0.25 (0)	0.22 (0)	0.28 (0)	0.23 (0)
10	**0.14** (0)	**0.14** (0)	0.16 (0)	0.19 (0)	0.16 (0)	0.22 (0)	0.17 (0)
12	**0.13** (0)	**0.13** (0)	0.17 (0)	0.18 (0)	0.14 (0)	0.20 (0)	0.14 (0)
14	0.13 (0)	0.13 (0)	0.18 (0)	0.17 (0)	0.13 (0)	0.19 (0)	**0.12** (0)
16	**0.13** (0)	**0.13** (0)	0.19 (0)	0.18 (0)	**0.13** (0)	0.19 (0)	**0.13** (0)
18	**0.13** (0)	**0.13** (0)	0.19 (0)	0.18 (0)	**0.13** (0)	0.19 (0)	**0.13** (0)
20	**0.12** (0)	0.14 (0)	0.20 (0)	0.18 (0)	0.13 (0)	0.19 (0)	0.13 (0)

Table 3.4: Results for random PB-problems with PB-constraint length of 2 to 20 (first answer set)

answer sets. As expected, **periodic** and **always** are slower than **never** if many (more than about half of) the candidates are solutions. Frequent evaluation is detrimental here, as running time investment has no pay-off in information gain or early search termination. Likewise, minimizing all io-nogoods in conditions **ngm** and **qxp** performs worse than **never** as identical nogoods are computed for many complete assignments. However, the configuration **ngm-c** is very efficient and finds valuable io-nogoods without investing much running time because it focuses on valuable (i.e. conflicting) io-nogoods. Hence, the overhead of **ngm-c** compared to **never** is also small for instance sizes 18 and 20, where hardly any useful information for pruning the search space is available. In contrast to Table 3.1, **qxp-c** performs slightly worse than **ngm-c** because conflicting io-nogoods now contain mostly relevant literals. As the search space contains a large number of solutions for instances with $N > 6$, the first answer set is always found very fast for such instances.

Taxi Assignment with Ontology Access

To facilitate query access for logic programs to external *description logics knowledge bases (DL-KBs)* was one of the early motivating applications of the HEX-formalism, which has been syntactically framed by so-called *DL-programs* (Eiter et al., 2008). Common reasoning tasks w.r.t. DL-ontologies are concept and role retrieval, i.e. deriving all individuals respectively pairs of individuals that are instances of a given concept respectively role relationship.

For integrating concept and role queries into ASP, *DL-programs* provide so-called *DL-atoms*, which can be represented by external atoms of the form $\&DL[c^+, c^-, r^+, r^-, q](\vec{X})$. Here the inputs c^+ and c^- are binary predicates that declare positive and negative assertions of ontology concept instances, respectively. More specifically an atom $c^+(\text{“}C\text{”}, a)$ (resp. $c^-(\text{“}C\text{”}, a)$) encodes that $C(a)$ (resp. $\neg C(a)$) should be asserted in the DL-KB. Similarly, r^+ and r^- are ternary predicates where $r^+(\text{“}R\text{”}, a, b)$ (resp. $r^-(\text{“}R\text{”}, a, b)$) encodes that $R(a, b)$ (resp. $\neg R(a, b)$) should be asserted in the DL-KB. Evaluating the DL-atom retrieves all instances of the query q, which is either a concept or a role name,

$$drives(X,Y) \leftarrow driver(X), customer(Y), \&DL[n,n,n,n,isIn](X,A), \quad \text{(r1)}$$
$$\&DL[n,n,n,n,isIn](Y,A), region(A), \text{not } ndrives(X,Y).$$
$$ndrives(X,Y) \leftarrow driver(X), customer(Y), \text{not } drives(X,Y). \quad \text{(r2)}$$
$$driven(Y) \leftarrow drives(\_,Y). \quad \text{(r3)}$$
$$\leftarrow \text{not } driven(Y), customer(Y). \quad \text{(r4)}$$
$$\leftarrow drives(X,Y), drives(X1,Y), X \neq X1. \quad \text{(r5)}$$
$$r^+(\text{“}drivesEC\text{”},X,Y) \leftarrow drives(X,Y), \&DL[n,n,r^+,n,ECust](Y). \quad \text{(r6)}$$
$$\leftarrow \#count\{Y : drives(X,Y)\} > 4, driver(X). \quad \text{(r7)}$$
$$\leftarrow drives(X,Y), \text{not } \&DL[n,n,r^+,n,ECust](Y), \&DL[n,n,r^+,n,EDrv](X). \quad \text{(r8)}$$
$$\leftarrow drives(X,Y), \&DL[n,n,r^+,n,ECust](Y), \text{not } \&DL[n,n,r^+,n,EDrv](X). \quad \text{(r9)}$$

Figure 3.3: Taxi Assignment Rules

relative to the modified ontology. In this way, a bidirectional interaction between the rules of a logic program and the DL-KB is enabled. Accordingly, DL-programs constitute a special type of HEX-programs; using the *DL-Lite* plug-in for DLVHEX (Eiter, Fink, Redl, & Stepanova, 2014), one can evaluate DL-programs with a DL-KB formulated in the DL-Lite language.

For our experiments, we employ the DL-program shown in Figure 3.3, which assigns taxi drivers to customers under constraints. Our encoding is similar to the one by Eiter, Fink, and Stepanova (2016), but guesses assignments of drivers to customers such that different combinations are possible, whereby non-permissible ones can possibly be detected early by partial evaluation. An external DL-KB formulated in DL-Lite holds part of the information, e.g. about locations of individuals, about e-customers (customers demanding electric cars), and about e-drivers (drivers of electric cars). Here, exactly one driver is assigned to each customer by the rules (r1)-(r5), where the respective driver must be located in the same region as the customer. The latter condition is enforced by using information regarding the regions in which drivers and customers are located that is imported via the DL-atom $\&DL[n,n,n,n,isIn](X,A)$ from the external DL-KB. Customers may share the driver, where a taxi fits at most four customers according to rule (r7). Based on information about which customers are e-customers and which drivers are e-drivers, which is imported via the DL-atoms $\&DL[n,n,r^+,n,ECust](Y)$ and $\&DL[n,n,r^+,n,EDrv](X)$, e-customers must be assigned to e-drivers and normal customers to normal drivers according to rules (r8) and (r9), respectively. Moreover, drivers of e-customers are positively asserted for the concept *drivesEC* by rule (r6), which affects subsequent inferences in the DL-KB.

The answer sets of the program with the rules in Figure 3.3 and further facts *driver(d)*, *customer(c)* and *region(r)* for drivers d, customers c and regions r encode legal assignments. For a complete assignment $\mathbf{A}$, a ground DL-atom $\&DL[c^+,c^-,r^+,r^-,q](\vec{c})$ evaluates as

#	All Answer Sets																av. solutions
	never		**periodic**		**always**		**ngm-sq**		**ngm**		**ngm-c**		**qxp**		**qxp-c**		
4	0.20	(0)	0.18	(0)	0.22	(0)	0.19	(0)	0.18	(0)	**0.17**	(0)	0.18	(0)	**0.17**	(0)	7.88
6	0.33	(0)	0.26	(0)	0.32	(0)	0.29	(0)	0.26	(0)	**0.22**	(0)	0.26	(0)	**0.22**	(0)	17.44
8	1.13	(0)	0.41	(0)	0.60	(0)	0.57	(0)	0.47	(0)	**0.33**	(0)	0.41	(0)	**0.33**	(0)	36.08
10	7.61	(0)	0.89	(0)	1.42	(0)	1.60	(0)	1.15	(0)	0.66	(0)	0.88	(0)	**0.65**	(0)	93.76
12	228.44	(18)	2.19	(0)	3.98	(0)	5.50	(0)	2.98	(0)	**1.49**	(0)	2.65	(0)	1.75	(0)	329.92
14	300.00	(50)	9.25	(0)	12.71	(0)	24.52	(1)	15.44	(1)	**5.15**	(0)	6.78	(0)	6.45	(0)	651.52
16	300.00	(50)	15.34	(1)	24.22	(1)	55.81	(2)	33.73	(1)	**12.31**	(1)	16.89	(1)	13.27	(1)	≥964.68
18	300.00	(50)	67.38	(5)	79.43	(4)	131.03	(12)	87.58	(9)	**47.30**	(3)	51.88	(4)	58.43	(3)	≥2767.34
20	300.00	(50)	79.94	(6)	108.65	(7)	186.26	(21)	139.49	(14)	**50.26**	(3)	76.36	(5)	67.77	(6)	≥3783.20
22	300.00	(50)	**146.88**	(15)	201.91	(23)	265.82	(42)	209.16	(27)	160.66	(17)	167.53	(17)	178.43	(18)	≥5665.76
24	300.00	(50)	**194.62**	(25)	243.07	(32)	286.70	(46)	249.06	(34)	216.56	(28)	212.44	(25)	209.65	(26)	≥5840.56
26	300.00	(50)	265.54	(41)	284.26	(45)	294.01	(49)	290.69	(47)	**261.73**	(39)	275.54	(43)	265.89	(40)	≥5743.16
28	300.00	(50)	248.42	(39)	253.66	(42)	258.08	(43)	254.46	(42)	**243.08**	(39)	252.00	(41)	247.76	(40)	≥5057.28
30	300.00	(50)	293.90	(48)	294.02	(49)	294.01	(49)	294.01	(49)	**292.78**	(48)	294.02	(49)	294.01	(49)	≥5322.62

Table 3.5: Results for taxi assignment with ontology access (all answer sets)

follows:

$$f_{\&DL}(\mathbf{A}, c^+, c^-, r^+, r^-, q) = \begin{cases} \mathbf{T} & \text{if } q(\vec{c}) \text{ is derivable from } KB \cup Assrt(\mathbf{A}), \\ \mathbf{F} & \text{otherwise,} \end{cases}$$

where $Assrt(\mathbf{A})$ consists of all assertions $c(i)$ such that $\mathbf{T}c^+(\text{“}C\text{”}, i) \in \mathbf{A}$, all assertions $\neg c(i)$ such that $\mathbf{T}c^-(\text{“}C\text{”}, i) \in \mathbf{A}$, all assertions $r(i_1, i_2)$ such that $\mathbf{T}r^+(\text{“}r\text{”}, i_1, i_2) \in \mathbf{A}$ and all assertions $\neg r(i_1, i_2)$ such that $\mathbf{T}r^-(\text{“}r\text{”}, i_1, i_2) \in \mathbf{A}$. Exploiting monotonicity of DLs, the evaluation of the associated three-valued oracle function is as follows:

$$f_{\&DL'}(\mathbf{A}, c^+, c^-, r^+, r^-, q) = \begin{cases} \mathbf{T} & \text{if } q(\vec{c}) \text{ is derivable from } KB \cup Assrt(\mathbf{A}), \\ \mathbf{F} & \text{if } q(\vec{c}) \text{ is not derivable from } KB \cup Assrt(\mathbf{A}_{\max}), \\ \mathbf{U} & \text{otherwise,} \end{cases}$$

where $\mathbf{A}_{\max} \supseteq \mathbf{A}$ is the (unique) assignment leading to the largest addition set of assertions.

For instance, the atom $\&DL[n, n, r^+, n, EDrv](d)$ is true w.r.t. a partial assignment $\mathbf{A}$ if $KB \cup \{r(i_1, i_2) \mid \mathbf{T}r^+(\text{“}r\text{”}, i_1, i_2) \in \mathbf{A}\} \models EDrv(d)$; it is false if $KB \cup \{r(i_1, i_2) \mid \mathbf{F}r^+(\text{“}r\text{”}, i_1, i_2) \notin \mathbf{A}\} \not\models EDrv(d)$; and it is unassigned otherwise. The input parameters n in Figure 3.3 are dummies that, as they do not occur in rule heads or in facts added, have empty extent in every answer set.

In our tests, we increased the number N of drivers and customers gradually from 4 to 30, which were put in $N/2$ regions randomly, where the drivers were balanced among regions. Furthermore, half of the customers were e-customers. The results are shown in Tables 3.5 and 3.6.

As DL-atoms have output constants, simultaneous minimization (**ngm**) and sequential minimization (**ngm-sq**) yield different results in this benchmark and we tested both configurations. All configurations that exploit partial evaluations are significantly faster than **never**. The configuration **periodic** now shows better results than **always** because the external DL calls are costly, and waiting a bit until issuing the next one can pay

#	First Answer Set							
	never	**periodic**	**always**	**ngm-sq**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
4	**0.15** (0)	**0.15** (0)	0.16 (0)	**0.15** (0)	0.16 (0)	**0.15** (0)	0.16 (0)	**0.15** (0)
6	0.18 (0)	**0.16** (0)	0.18 (0)	0.18 (0)	0.18 (0)	0.17 (0)	0.18 (0)	0.18 (0)
8	0.36 (0)	**0.17** (0)	0.20 (0)	0.22 (0)	0.21 (0)	0.22 (0)	0.21 (0)	0.22 (0)
10	1.24 (0)	**0.18** (0)	0.23 (0)	0.25 (0)	0.25 (0)	0.33 (0)	0.25 (0)	0.32 (0)
12	23.56 (0)	**0.22** (0)	0.27 (0)	0.33 (0)	0.33 (0)	0.44 (0)	0.30 (0)	0.43 (0)
14	139.70 (16)	**0.25** (0)	0.32 (0)	0.41 (0)	0.44 (0)	1.10 (0)	0.36 (0)	1.08 (0)
16	273.78 (40)	**0.29** (0)	0.37 (0)	0.49 (0)	0.63 (0)	7.39 (1)	0.42 (0)	7.50 (1)
18	300.00 (50)	**0.40** (0)	0.42 (0)	0.59 (0)	0.73 (0)	23.39 (2)	0.50 (0)	26.53 (3)
20	300.00 (50)	**0.34** (0)	0.46 (0)	0.66 (0)	2.00 (0)	2.12 (0)	0.56 (0)	9.62 (1)
22	300.00 (50)	**0.43** (0)	0.69 (0)	0.64 (0)	0.53 (0)	61.87 (4)	0.67 (0)	53.99 (3)
24	300.00 (50)	**0.46** (0)	0.76 (0)	0.72 (0)	0.60 (0)	113.78 (12)	0.77 (0)	88.03 (9)
26	300.00 (50)	**0.52** (0)	0.86 (0)	0.85 (0)	0.68 (0)	59.02 (6)	0.85 (0)	84.77 (10)
28	300.00 (50)	**0.56** (0)	1.00 (0)	0.90 (0)	0.74 (0)	76.59 (5)	0.88 (0)	95.74 (14)
30	300.00 (50)	**0.63** (0)	1.10 (0)	1.04 (0)	0.83 (0)	112.02 (11)	1.05 (0)	103.03 (11)

Table 3.6: Results for taxi assignment with ontology access (first answer set)

off. Since the premise of an io-nogood can be large but the output often depends only on a small part, minimization can drastically shrink io-nogoods. However, this comes at the price of many external calls due to the large size of the io-nogoods, such that **ngm-sq** is slower than **periodic** and **always**. The costs of minimization can be reduced by minimizing nogoods with the same premise simultaneously, or applying binary search in form of the QuickXplain algorithm. Accordingly, both **ngm** and **qxp** perform better than **ngm-sq**. Moreover, we observe that **qxp** is slightly faster than **ngm**, even though io-nogoods with identical input parts are not minimized simultaneously by the QuickXplain algorithm. As for the previous benchmark, minimizing only conflicting nogoods in conditions **ngm-c** and **qxp-c** yields the best results.

Notably, by employing partial evaluations, the first solution can be found rapidly and much faster than in condition **never**, except for the configurations **ngm-c** and **qxp-c**. In contrast to the PB-problems benchmark, here the use of external atoms is not limited to constraints such that minimal nogoods obtained from non-conflicting io-nogoods may contain valuable information. As a result, the missing information leads to timeouts for certain instances even before the first answer set is found, while for other instances the set of all answer sets can be computed faster by **ngm-c** and **qxp-c** than by other configurations.

Conflicting Strategic Companies

Strategic Companies is a business problem that is a popular benchmark for ASP competitions, located at the second level of the polynomial hierarchy (Cadoli et al., 1997; Leone et al., 2006). The scenario is that a set $C = \{c_1, ..., c_m\}$ of companies and a set $G = \{g_1, ..., g_n\}$ of goods are given, where each company $c_i \in C$ produces some goods $G_i \subseteq G$ and is possibly controlled by a consortium of owner companies $O_i \subseteq C$. A set of companies $C' \subseteq C$ constitutes a *strategic set* if (1) the companies in C' produce all the goods in G, (2) if $O_i \subseteq C'$ for some $1 \leq i \leq m$, then c_i is in C' as well, and (3) C' is subset-minimal w.r.t. conditions (1) and (2) (Leone et al., 2006). The knowledge

#	All Answer Sets														av. solutions
	never		**periodic**		**always**		**ngm**		**ngm-c**		**qxp**		**qxp-c**		
5	0.15	(0)	**0.14**	(0)	**0.14**	(0)	**0.14**	(0)	0.15	(0)	**0.14**	(0)	**0.14**	(0)	1.72
10	**0.13**	(0)	**0.13**	(0)	0.14	(0)	**0.13**	(0)	**0.13**	(0)	**0.13**	(0)	**0.13**	(0)	3.22
15	0.20	(0)	0.19	(0)	0.21	(0)	0.16	(0)	0.16	(0)	**0.15**	(0)	**0.15**	(0)	8.08
20	0.65	(0)	0.49	(0)	0.51	(0)	0.25	(0)	0.21	(0)	**0.20**	(0)	**0.20**	(0)	23.42
25	3.49	(0)	1.60	(0)	1.35	(0)	0.43	(0)	0.31	(0)	0.29	(0)	**0.27**	(0)	53.50
30	26.79	(0)	7.61	(0)	5.30	(0)	1.09	(0)	0.55	(0)	0.50	(0)	**0.44**	(0)	105.32
35	193.38	(0)	33.34	(0)	17.46	(0)	5.72	(0)	1.05	(0)	1.07	(0)	**0.87**	(0)	282.26
40	300.00	(50)	135.66	(0)	46.70	(0)	45.95	(3)	1.82	(0)	2.03	(0)	**1.53**	(0)	507.08
45	300.00	(50)	297.18	(49)	139.51	(5)	131.44	(15)	5.46	(0)	13.53	(1)	**4.97**	(0)	1794.60
50	300.00	(50)	300.00	(50)	267.27	(33)	227.98	(31)	10.25	(0)	27.46	(1)	**9.59**	(0)	3453.50
55	300.00	(50)	300.00	(50)	295.25	(46)	262.83	(42)	35.64	(0)	108.24	(14)	**35.15**	(0)	5419.98
60	300.00	(50)	300.00	(50)	300.00	(50)	297.71	(49)	56.68	(1)	147.00	(16)	**55.26**	(1)	≥6300.94
65	300.00	(50)	300.00	(50)	300.00	(50)	295.89	(49)	151.06	(11)	242.18	(35)	**150.19**	(11)	≥7531.40
70	300.00	(50)	300.00	(50)	300.00	(50)	293.64	(48)	194.48	(21)	267.46	(41)	**192.18**	(22)	≥7005.06

Table 3.7: Results for conflicting strategic companies (all answer sets)

$$s(C1) \vee s(C1) \vee s(C3) \vee s(C4) \leftarrow \mathit{producedBy}(\_, C1, C2, C3, C4).$$
$$s(C) \leftarrow \mathit{controlledBy}(C, C1, C2, C3, C4), s(C1), s(C2), s(C3), s(C4).$$
$$\leftarrow \&\mathit{stratConflict}[s]().$$

Figure 3.4: Conflicting Strategic Companies Rules

about which companies belong to a strategic set can be crucial for a holding owning the companies in C, e.g. if it has to sell some of its companies and does not want to suffer a loss in economic power. The problem can be encoded concisely in ASP by exploiting the minimality of answer sets, so that each answer set corresponds to one strategic set.

In our benchmark setting, we assume that each product is produced and each company is controlled by at most four companies in C. We further assume an additional conflict relation $R \subseteq C \times C$, such that companies which are related by R cannot occur together in a strategic set. This constraint makes sense when certain companies cannot be kept simultaneously, e.g. due to legislation. The program in Figure 3.4 encodes the strategic sets that satisfy the conflict relation in its answer sets. In this program, we check the conflict constraint on strategic sets via the external atom $\&\mathit{strategicConflict}[\mathit{strategic}]()$, where *strategic* contains all companies in the strategic set; on complete assignments, it evaluates to true if some companies c_i, c_j in *strategic* are in conflict, i.e., $(c_i, c_j) \in R$ holds (where R is externally stored).

Since finding a strategic set is computationally hard, excluding candidate strategic sets with a conflict early in the search by partial evaluations should noticeably decrease the running time. We use for such evaluations a three-valued oracle function $f_{\&\mathit{strategicConflict}'}(\mathbf{A}, s)$, defined as follows:

$$f_{\&\mathit{strategicConflict}'}(\mathbf{A}, s) = \begin{cases} \mathbf{T} & \text{if } \mathbf{T}s(c_i), \mathbf{T}s(c_j) \in \mathbf{A} \text{ holds for some } (c_i, c_j) \in R, \\ \mathbf{F} & \text{if } \mathbf{F}s(c_i) \in \mathbf{A} \text{ or } \mathbf{F}s(c_j) \in \mathbf{A} \text{ for every } (c_i, c_j) \in R, \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

We ran tests on instances with $N \in [5, 70]$ companies, at most N randomly assigned

#	First Answer Set						
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
5	**0.14** (0)	**0.14** (0)	0.15 (0)	**0.14** (0)	**0.14** (0)	0.15 (0)	0.15 (0)
10	**0.12** (0)	**0.12** (0)	0.13 (0)	**0.12** (0)	**0.12** (0)	**0.12** (0)	**0.12** (0)
15	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	**0.13** (0)
20	0.24 (0)	0.20 (0)	0.17 (0)	0.16 (0)	0.16 (0)	**0.15** (0)	**0.15** (0)
25	0.76 (0)	0.36 (0)	0.21 (0)	0.20 (0)	0.20 (0)	**0.18** (0)	**0.18** (0)
30	4.60 (0)	1.03 (0)	0.41 (0)	0.29 (0)	0.29 (0)	**0.23** (0)	**0.23** (0)
35	26.05 (0)	2.71 (0)	0.54 (0)	0.39 (0)	0.37 (0)	**0.27** (0)	**0.27** (0)
40	118.70 (12)	10.67 (0)	1.47 (0)	0.54 (0)	0.54 (0)	0.34 (0)	**0.33** (0)
45	158.24 (22)	21.35 (0)	1.38 (0)	0.56 (0)	0.55 (0)	**0.34** (0)	**0.34** (0)
50	230.00 (32)	43.29 (3)	6.19 (0)	0.76 (0)	0.75 (0)	**0.40** (0)	**0.40** (0)
55	287.99 (47)	131.69 (15)	5.67 (0)	1.05 (0)	1.04 (0)	**0.51** (0)	**0.51** (0)
60	291.70 (47)	187.90 (26)	10.10 (0)	1.47 (0)	1.45 (0)	0.64 (0)	**0.63** (0)
65	294.49 (49)	229.99 (35)	15.06 (0)	1.85 (0)	1.86 (0)	**0.76** (0)	0.77 (0)
70	300.00 (50)	244.27 (39)	52.26 (1)	2.32 (0)	2.26 (0)	0.85 (0)	**0.83** (0)

Table 3.8: Results for conflicting strategic companies (first answer set)

control relations, $5 \times N$ products with randomly assigned producers, and $N/2$ randomly created conflicts. The results are shown in Tables 3.7 and 3.8.

The external conflict constraint cuts more than 90% of the strategic sets (i.e., solution candidates). Thus, like in the first PB-problems benchmark, only a small part of the search space contains solutions. Accordingly, we observe a similar pattern as in Tables 3.1 and 3.2, where partial evaluation significantly decreases the running time in all conditions. The configuration **qxp-c** again exhibits the best results. Since strategic sets are minimal, io-nogoods learned on complete assignments do not provide any valuable information, such that we did not observe a difference when the learning function Λ_u is used instead of Λ_{mu} in this case. Notably, for computing strategic sets containing a specific company (which is Σ_2^{P}-hard in general) we obtain similar results.

Regarding the results for finding the first answer set, the larger difference between **never** and **always** in comparison to Table 3.2 is due to the higher computational effort required for finding compatible sets, where learning based on partial assignments is able to guide the search towards a compatible set which is also an answer set.

3.5.4 Discussion of Results

In our experiments, we found that early evaluation in conditions **always** and **periodic** increased the performance for all benchmarks, except for the case where nearly all candidate solutions correspond to answer sets such that no useful information is obtainable from additional oracle calls. This finding is in line with hypothesis (H3.1). Moreover, in the case of the taxi assignment benchmark, where external calls require more running time than in the other benchmark implementations, **periodic** performed better than **always** due to less runtime overhead, which supports our hypothesis (H3.2). Similar improvements could be achieved by minimizing all io-nogoods that are learned based on complete assignments with configuration **ngm**. However, regarding hypothesis (H3.3), the results are mixed because minimization performs worse when io-nogoods are large or contain many relevant literals (cf. Tables 3.3 and 3.5). This overhead is avoided

by only minimizing nogoods that directly trigger backjumping in condition **ngm-c**. The configuration **ngm-c** performs well for all benchmark problems and has only little overhead when no useful information is available, as can be observed in Table 3.3.

Finally, the fact that **qxp** performed better than **ngm** in the taxi assignment benchmark, where io-nogoods typically contain many irrelevant literals, and did not increase the running time much when nearly all literals are relevant (as it is the case for the second experiment using PB-problems) provides supporting evidence for hypothesis (H3.4). This effect results from the number of external calls that have to be performed in the best respectively the worst case by the QuickXplain algorithm compared to sequential minimization.

Overall, minimization of conflicting nogoods by configuration **ngm-c** or **qxp-c** in most cases yielded the best results with only small differences between the two conditions. Hence, they are suggested as the default configurations.

3.6 Related Work

As mentioned in the introduction of this chapter, our work is most closely related to SMT solving, in particular to theory propagation there (Nieuwenhuis et al., 2006), and naturally to constraint ASP solving, as developed by Gebser, Ostrowski, and Schaub (2009), and theory solving in clingo 5 (Gebser et al., 2016).

As for the relation to SMT solving, we observe that the latter typically considers fixed types of theories, while hex is more general and geared towards supporting heterogeneous theories. Using a fixed type of theory is a characteristics of several extensions of ASP with SMT, such as dingo (Janhunen et al., 2011), which uses difference logic, NLP-DL (Eiter, Ianni, Schindlauer, & Tompits, 2005a), which uses description logics, and ASPMT (Lee & Meng, 2013).

Moreover, the abstract level of semantics in terms of input-output relations accommodates even arbitrary non-logical theories. However, there is closer similarity regarding integration schemas and learning techniques. Typical integration schemas for SMT have been identified (Balduccini & Lierler, 2013b), which apply to ASP modulo theories as well (a comparison is given by Balduccini and Lierler (2013a)):

- In *black-box* integration, the SAT-solver blindly generates a model and passes it for checking to the theory solver. If it passes the check, the model is returned, otherwise it is added in constraint form to the instance and the solver restarts. This allows for easy coupling with arbitrary theories but does not enable search space pruning.

- In *grey-box* integration, the theory solver is only called for complete models of the SAT-instance, but the SAT-solver is merely suspended during checking and can continue its search afterwards; integration is still relatively simple.

- Only in *clear-box* integration, the SAT-solver is interleaved with the theory solver, which is called already for partial assignments and in turn may propagate further

truth values or detect inconsistencies. However, the integration is much more challenging as the theory solver must identify atoms implied by the given partial assignment, or by inconsistency reasons, respectively.

Examining HEX, the grey-box schema corresponds to the evaluation algorithms in use before external behavior learning was introduced by Eiter et al. (2012); black-box integration, i.e. resorting to complete restarts, has never been used for HEX solving. With incorporation of such learning, the algorithms fit an intermediate schema between grey- and clear-box integration: external sources were still only evaluated under complete assignments, but the learned nogoods possibly pruned the search space.

Compared to constraint ASP-solving, the HEX-formalism is more general as it supports access to arbitrary external sources which are largely black boxes, and without (implicit) assumptions of their properties. In this respect, constraint ASP can be considered as a special case of HEX with theory-specific knowledge. There are a number of integrations of ASP with constraint programming, as realized e.g. in CLINGCON (Ostrowski & Schaub, 2012), LC2CASP (Cabalar et al., 2016), EZCSP (Balduccini, 2009), and EZSMT (Susman & Lierler, 2016); we refer to Lierler et al. (2016) for an overview of systems. Here, we focus on the work of Ostrowski and Schaub (2012), who considered nogood minimization as we do, but used different algorithms that avoid expensive resets of the constraint solver. However, this is only possible by exploiting properties of the specific theory at hand (monotonic constraint satisfaction), which in our more general setting do not always apply (e.g., for nonmonotonic external atoms); furthermore, the user-friendly plug-and-play integration of external sources does not provide control over the external algorithms. On the other hand, other possibilities for optimizations arise, e.g., simultaneous io-nogood minimization since external atoms can have multiple output values for the same input.

Unlike HEX-programs, theory solving in CLINGO 5 (and constraint ASP as a special instance thereof) does not support external atoms with dedicated input and output. Instead, certain atoms in the logic program are declared as *theory atoms* whose truth values are set via the external theory. In that, CLINGO 5 follows a global perspective where theory atoms may be shared by different rules, rather than the local one of external atoms in HEX-programs where the scope is the rule body (as customary in logic programming).

Related to techniques for nogood minimization, nowadays most SAT-solvers integrate techniques for *learned clause minimization*, which remove redundancies from learned clauses by computationally inexpensive procedures (Sörensson & Biere, 2009). However, the role of io-nogoods learned from external source evaluations in HEX and the role of nogoods learned from conflicts by a respective ASP-solver, even though both serve the purpose of guiding the search procedure, are very different. While conflict nogoods are usually obtained by resolution based on an available implication graph such that minimization techniques as the ones by Sörensson and Biere can be applied, io-nogoods are not learned using an implication graph but from single oracle calls. Due to the black box nature of these oracles, they can only be used to retrieve all correct outputs for external atoms w.r.t. a given assignment. Hence, techniques for learned clause minimization are not directly applicable for learning smaller io-nogoods. Nevertheless, such techniques can still be exploited in our approach for conflict nogood minimization, depending on

whether the respective solver used for performing the main CDNL-search implements them.

Other related work comprises alternative solving techniques, such as the one by Eiter, Fink, Redl, and Stepanova (2014), where the semantics of external atoms is captured by so-called *support sets*, which are similar to our faithful io-nogoods and related to implicants of logical theories (Darwiche & Marquis, 2002; Reiter & de Kleer, 1987). However, different from our approach, the main idea there is to learn all or sufficiently many support sets at the beginning of the solving process, such that satisfaction and unsatisfaction of an external atom under given input is completely covered. External atom evaluation can then be accomplished by matching the support sets against the interpretation; this eliminates external calls during solving entirely, but comes at the price of learning up to exponentially many support sets. A related support-set based approach goes a step further and encodes the semantics of external atoms straight into the ASP-program (Redl, 2017b). The exponential worst-case blowup suggests to use these approaches only for external atoms with a compact and small representation by support sets. Moreover, since they genuinely depend on exhaustive learning of support sets at the beginning, they cannot directly benefit from the possibility of partial evaluation as presented in the previous sections.

The notion of three-valued oracle functions has also been used to extend HEX with *lazy-grounding* techniques (Eiter, Kaminski, & Weinzierl, 2017), where not only the evaluation of external atoms is postponed, but also the grounding of the HEX-program itself, by employing as backend-solver the lazy-grounding ASP-solver ALPHA (Weinzierl, 2017). This approach is presented in Chapter 5.

Finally, Antic et al. (2013) considered partial HEX-semantics before, by employing *Approximation Fixpoint Theory* (*AFT*) (Denecker et al., 2000, 2004) that works on intervals in the power set lattice. While our partial oracle functions amount to their three-valued oracle functions, we only consider two-valued answer sets and we do not apply a fixpoint construction to define the answer set semantics. Similarly, Pelov et al. (2004) have defined a family of partial stable model semantics for logic programs with aggregates using AFT. Assignment-monotonic oracle functions are also related to their *approximating aggregate relations* which must be *precision-monotone* and generalize ordinary aggregate relations to a three-valued semantics.

3.7 Conclusion and Outlook

In this chapter, we have pushed efficient evaluation techniques for ASP with external source access, by introducing three-valued evaluation of external atoms under partial (incomplete) truth value assignments. The techniques we introduced yield a full-fledged clear-box integration. Moreover, due to automatic nogood minimization, developers of external sources do not need to manually describe implied truth values or inconsistency reasons, but only need to implement a three-valued oracle function, which keeps the integration of sources simple.

In our experiments, the new techniques yielded a speedup of up to two orders of magnitude; unsurprisingly, their ranking depends on the instances. This is similar to the observations by Ostrowski and Schaub (2012), who reported mixed results for different propagation delays. Our results are also in line with results in SMT, where theory propagation, if doable with small overhead, is crucial for performance (Dutertre & de Moura, 2006; Lahiri, Nieuwenhuis, & Oliveras, 2006; Nieuwenhuis & Oliveras, 2005). We observed that in most cases learning from complete assignments plus minimization of conflicting nogoods (based on partial assignments) outperforms learning during search; hence, this setting is suggestive as a default. This is explained by the fact that in this case, learning focuses on nogoods that are useful for conflict resolution, thus the information gain is similar and the overhead much smaller. This is in line with the observation by Nieuwenhuis et al. (2006) that conflict analysis uses only a small fraction of the lemmas learned by theory propagation, which can be addressed with *lazy explanations* (Gent, Miguel, & Moore, 2010). The speedup can be exponential, as evidenced by an external atom whose truth value is definite after assigning a single input atom, e.g. *&empty*[*p*]() to check whether an atom over p is true. Each naive nogood eliminates one of exponentially many assignments, but a linear number of minimized ones eliminate all wrong guesses.

There are different directions for ongoing and future work. One topic is to include further heuristics for deciding whether external evaluation is invoked or skipped at some point. This decision might be based, for instance, on the past information gain; other criteria are conceivable. Another topic is further improvement of nogood minimization. To this end, the divide-and-conquer strategy borrowed from Junker's QuickXplain algorithm (2004) might be replaced by a more sophisticated one, e.g. the one that Shchekotykhin et al. (2015) developed for their MergeXplain algorithm. By the latter, multiple minimal conflict sets (resp., nogoods) can be found during one program run; this could be integrated into our approach for obtaining multiple minimal io-nogoods.

CHAPTER 4

Integration of Minimality Checking and External Evaluation

After having developed techniques for tightly integrating external evaluation with the main search of HEX-solving in the previous chapter, we now focus on achieving a tighter integration of the special minimality check required for evaluating HEX-programs (cf. Section 2.4) and external evaluation. Accordingly, in this chapter, which is based on the papers (Eiter et al., 2018) and (Eiter & Kaminski, 2019), we extend partial evaluation to the search for unfounded sets applied for ensuring e-minimality of answer sets, and introduce a novel pruning technique for detecting cases where e-minimality checking is not required.

As described in Section 2.3, not every compatible set is also an answer set of a HEX-program, due to the possibility of cyclic support involving external atoms. Hence, a notable difference to ordinary ASP is that an additional e-minimality check (cf. Part (d) in Algorithm 3.1) is required to avoid unfounded support by external atoms in order to find answer sets of HEX-programs.

For instance, an external atom $\&closeTo[city](X)$ that outputs all cities located close to cities in the extension of the predicate *city* might be utilized in a rule $closeCity(X) \leftarrow \&closeTo[city](X), location(X)$. Now, if the locations in the domain are *osaka*, *kobe*, *bratislava* and *vienna*, and the rule $city(X) \leftarrow closeCity(X)$ as well as the fact $city(osaka)$ are added, only the atom $city(kobe)$ should be contained in an answer set in addition. Even though Bratislava and Vienna are located close to each other, the atoms $city(bratislava)$ and $city(vienna)$ can only cyclically support each other via the two rules and the external atom. The e-minimality check of HEX eliminates spurious answer sets containing the latter two atoms.

So far, we have only considered external evaluations based on partial assignments which are performed during the search for compatible sets. In general, it is also necessary to evaluate external atoms again for finding smaller models of the FLP-reduct because their truth value might change when the truth value of some ordinary atom is switched

from true to false. In previous approaches for HEX-evaluation, similar as in the case of the main search, this evaluation could only be performed after the complete input to an external atom in a potential smaller model had been decided since input atoms were not allowed to be unassigned. This is particularly important as the minimality check accounts for a major share of the overall running time, and usually involves significantly more external evaluation calls than the search for candidates itself.

Furthermore, note that if the rule $city(X) \leftarrow closeCity(X)$ is not added above, cyclic support via the external atom can be ruled out independent of the external semantics. Based on this observation, also a syntactic criterion was presented in (Eiter, Fink, Krennwallner, et al., 2014) for deciding whether the e-minimality check can be skipped for a program, which often results in significant speedups. Alternatively, if the external atom $\&closeWest[city](X)$ is used in the example (only retrieving cities close *to the west* of input cities), cyclic support can also be excluded. This cannot be detected by a syntactic criterion, such that the e-minimality check needs to be performed in any case by the previous approach. Moreover, applying a semantic criterion is challenging, as previously external atoms have largely been considered as black boxes that conceal semantic dependencies.

For this reason, in addition to extending techniques for partial evaluation to the e-minimality check, we develop a new approach for pruning e-minimality checking that also exploits semantic dependencies. It relies on additional information about *input-output* (*io-*)*dependencies* of external atoms, which may be provided by a user, or even generated automatically. Hidden io-dependencies are common in applications involving recursive processing, e.g. over external graphs or *semantic web* data. In this context, supplied dependency information can be incomplete and added flexibly.

The content of this chapter is structured as follows:

- In Section 4.1, we exploit the possibility for evaluation under partial assignments for e-minimality checking. In particular, we discuss how three-valued external evaluation can be interleaved with the search for an unfounded set. As in the previous chapter, learning from external source calls is used for guiding the search; notably, the nogoods learned can be pooled with those in the main search, and thus speed up the latter as well.

- In Section 4.2, we provide a novel formalization of io-dependencies that encode semantic dependency information, and we show under which condition they can safely be used for pruning the e-minimality check. In addition, we state theoretical properties crucial for checking and optimizing io-dependencies, and show when the associated costs can be reduced.

- In Section 4.3, we present an experimental evaluation using illustrative benchmark problems that confirms the advantage of utilizing partial evaluation during the unfounded set search and of exploiting io-dependencies for reducing the number of e-minimality checks that need to be performed.

- In Section 4.4, we discuss related work; and conclude by summarizing and discussing future work in Section 4.5.

Our new approach not only applies to the HEX-formalism, but may also be employed analogously for other approaches that integrate external sources into ASP, such as CLINGO (Gebser et al., 2016), if external cyclic support is not desired.

4.1 Interleaving External Evaluation and Unfounded Set Search

In this section, we discuss how, based on three-valued assignments, the evaluation of external atoms can be interleaved with the search performed during the e-minimality check. As in Chapter 3, the goal is to increase the efficiency by evaluating external atoms as early as possible and thus, to potentially avoid many wrong guesses. In addition, the models of the FLP-reduct (built for a complete assignment) often outnumber the compatible sets of the program (cf. Eiter, Fink, Krennwallner, et al., 2014), and for each such set the guesses for the truth values of external atoms need to be verified.

We consider a more sophisticated variant of the FLP-check that utilizes the concept of *unfounded sets*, introduced in Section 2.4, in order to ensure e-minimality of answer sets in HEX, i.e., that they amount to minimal models of the FLP-reduct with the complete interpretation encoded by a compatible set. Eiter, Fink, Krennwallner, et al. (2014) showed that ensuring the absence of unfounded sets is a more efficient strategy for verifying e-minimality than applying the explicit FLP-check, due to the fact that smaller models of the FLP-reduct do not have to be generated explicitly in the former case. However, truth values of external atoms still need to be checked as described above to verify that candidate unfounded sets that have been detected actually constitute unfounded sets.

Example 4.1. Reconsider the program $\Pi = \{p \leftarrow \&id[p]().\}$ from Example 2.6. As observed, $\mathbf{A} = \{\mathbf{T}p\}$ is not an answer set of the program since it is not a subset-minimal model of $f\Pi^{\mathbf{A}} = \Pi$. This is because there is an unfounded set $U = \{\mathbf{T}p\}$, which intersects with the true atoms in $\mathbf{A}$: the only rule whose head intersects with $\{a \mid \mathbf{T}a \in U\}$ is $p \leftarrow \&id[p]()$, for which condition (2) of Definition 2.6 is satisfied. △

To enable external checks at any point during the search for unfounded sets, even before a candidate unfounded set has been detected, we introduce a novel algorithm for unfounded set checking that exploits external evaluations based on partial assignments. Subsequently, we show the correctness and completeness of the new algorithm. Hereby, interleaving unfounded set search with external evaluations can initiate backjumping as soon as it can be determined that guesses for external atoms violate the conditions for unfounded sets. So far, this could only be detected by means of a post-check. As before, input-output relations learned from oracle calls w.r.t. partial assignments can also be exploited to avoid wrong guesses in the further unfounded set search.

We start by providing background on the previous unfounded set check for HEX-programs (Eiter, Fink, Krennwallner, et al., 2014), which we extend to partial evaluations in the following.

4.1.1 Background on Unfounded Set Search

For detecting unfounded sets of a HEX-program Π w.r.t. a complete assignment $\mathbf{A}$, Eiter, Fink, Krennwallner, et al. (2014) introduced an encoding Ω_Π that is represented by a set of nogoods such that solutions to the encoding that are compatible with the semantics of external sources used in the program Π correspond exactly to the unfounded sets of Π w.r.t. $\mathbf{A}$. The encoding is uniform w.r.t. all executions of the unfounded set check, i.e. it does not depend on the current assignment and thus, only needs to be generated once. Accordingly, a compatible set $\mathbf{A}$ for which the check is performed needs to be injected by adding a set of so-called *assumptions* $\mathcal{A}_\mathbf{A}$, represented by a consistent set of signed literals, that fix the truth values of dedicated atoms. In this way, the encoding does not have to be regenerated for each compatible set from scratch, and in an implementation, assumptions can be treated in a special way such that part of the solver state can be maintained when assumptions are changed. As a result, a SAT-solver can be utilized to detect unfounded set candidates by searching for a solution to Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$.

Because external replacement atoms in Ω_Π do not encode the truth values of external atoms w.r.t. a solution $\mathbf{S}$ of the SAT encoding, but relative to a compatible set modified by $\mathbf{S}$, faithful io-nogoods learned w.r.t. $\mathbf{S}$ cannot be added directly to the encoding. For this reason, Eiter, Fink, Krennwallner, et al. (2014) defined a nogood transformation $\mathcal{T}_\Omega$ that ranges over io-nogoods and yields corresponding nogoods that imply the correct truth value for external replacement atoms in Ω_Π. We do not go into the details of the particular encoding and the nogood transformation here, as they are not relevant for our purposes; we refer to (Eiter, Fink, Krennwallner, et al., 2014) for more information.

4.1.2 Integrated Algorithm for Unfounded Set Detection

As in the search for compatible sets, we can also add the input-output relations that are learned from external evaluations based on partial assignments for the SAT encoding in form of nogoods to the SAT-solver. However, here we have to take into account that external replacement atoms do not encode the truth values of external atoms under a partial assignment in the solver, but represent their evaluation relative to the current compatible set modified by the respective partial assignment for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$. For this reason, we generalize the definition of $\mathbf{A} \dot\cup \neg.\mathbf{X}$ as follows, considering also partial assignments for the SAT encoding.

Definition 4.1 (Partial Assignment Operator)**.** *Given a complete assignment* $\mathbf{A}$ *and a partial assignment* $\mathbf{X}$*, let* $\mathbf{A} \dot\cup \neg.\mathbf{X} = (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\}) \cup \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}\} \cup \{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X} \text{ and } \mathbf{T}a \in \mathbf{A}\}$.

In contrast to Definition 2.6, where U is considered to be a complete assignment, atoms which are unassigned in $\mathbf{X}$ and true in $\mathbf{A}$ remain unassigned in $\mathbf{A} \dot\cup \neg.\mathbf{X}$; those

Algorithm 4.1: HEX-UFSCheck

Input: A ground HEX-program Π, a complete assignment $\mathbf{A}$, a set of nogoods ∇ of Π
Output: *true* if the true part of $\mathbf{A}$ intersects with an unfounded set for Π w.r.t. $\mathbf{A}$ and *false* otherwise, learned nogoods added to ∇

$\Omega'_\Pi \leftarrow \Omega_\Pi \cup \mathcal{A}_\mathbf{A} \cup \{\mathcal{T}_\Omega(N) \mid N \text{ is an io-nogood in } \nabla\}$ `// SAT instance with // assumptions and // io-nogoods from main // search`
$\mathbf{S} \leftarrow \{\mathbf{U}a \mid a \in A(\Omega'_\Pi)\}$ `// all atoms unassigned`
$dl \leftarrow 0$ `// decision level`
while *true* **do**
(a) $\quad\mathbf{S} \leftarrow \mathsf{Propagation}(\Omega'_\Pi, \mathbf{S})$
(b) $\quad$**if** *some nogood in* Ω'_Π *violated by* $\mathbf{S}$ **then**
$\quad\quad$**if** $dl = 0$ **then return** *false*
$\quad\quad$Analyze conflict, add learned nogood to Ω'_Π, set dl to backjump level
(c) $\quad$**else if** $\mathbf{S}$ *is complete* **then**
$\quad\quad$*isUFS* $\leftarrow$ *true*
$\quad\quad$**for** *all external atoms* $\&g[\vec{p}](\vec{c})$ *in* Π **do**
$\quad\quad\quad\nabla \leftarrow \nabla \cup \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})$
$\quad\quad\quad\Omega'_\Pi \leftarrow \Omega'_\Pi \cup \{\mathcal{T}_\Omega(N) \mid N \in \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})\}$
$\quad\quad\quad$**if** $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$, $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ *and* $\mathbf{A} \dot\cup \neg.\mathbf{S} \not\models \&g[\vec{p}](\vec{c})$ **then**
$\quad\quad\quad\quad$*isUFS* $\leftarrow$ *false*
$\quad\quad\quad$**end**
$\quad\quad\quad$**if** $\mathbf{F}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$, $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ *and* $\mathbf{A} \dot\cup \neg.\mathbf{S} \models \&g[\vec{p}](\vec{c})$ **then**
$\quad\quad\quad\quad$*isUFS* $\leftarrow$ *false*
$\quad\quad\quad$**end**
$\quad\quad$**end**
$\quad\quad$**if** *isUFS* **then**
$\quad\quad\quad$Let N be a nogood learned from the UFS
$\quad\quad\quad\nabla \leftarrow \nabla \cup \{N\}$
$\quad\quad\quad$**if** $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$ **then return** *true*
$\quad\quad$**else**
$\quad\quad\quad\Omega'_\Pi \leftarrow \Omega'_\Pi \cup \{\mathbf{S}\}$
$\quad\quad$**end**
(d) $\quad$**else if** *Heuristics evaluates* $\&g[\vec{y}]$ *and* $\Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S}) \not\subseteq \nabla$ **then**
$\quad\quad\nabla \leftarrow \nabla \cup \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})$
$\quad\quad\Omega'_\Pi \leftarrow \Omega'_\Pi \cup \{\mathcal{T}_\Omega(N) \mid N \in \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})\}$
(e) $\quad$**else**
$\quad\quad$Guess σa with $\sigma \in \{\mathbf{T}, \mathbf{F}\}$ for some variable a with $\mathbf{U}a \in \mathbf{S}$
$\quad\quad dl \leftarrow dl + 1$
$\quad\quad\mathbf{S} \leftarrow (\mathbf{S} \setminus \{\mathbf{U}a\}) \cup \{\sigma a\}$
$\quad$**end**
end

atoms can potentially be set to false in $\mathbf{A} \dot\cup \neg.\mathbf{X}'$ w.r.t. some assignment $\mathbf{X}' \succeq \mathbf{X}$. Atoms that are true in $\mathbf{X}$ and $\mathbf{A}$ are false under $\mathbf{A} \dot\cup \neg.\mathbf{X}$ as before.

Example 4.2. Consider the complete assignment $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ and the assignment $\mathbf{X} = \{\mathbf{T}p, \mathbf{F}q, \mathbf{U}r\}$. We then obtain $\mathbf{A} \dot{\cup} \neg.\mathbf{X} = \{\mathbf{F}p, \mathbf{T}q, \mathbf{U}r\}$. △

We note that due to assignment monotonicity of three-valued oracle functions, extending in a partial assignment $\mathbf{A} \dot{\cup} \neg.\mathbf{X}$ the set $\mathbf{X}$ does not change the value of an oracle function call that is determined (i.e., yields true or false). Formally:

Proposition 4.1. *Let* $\mathbf{A}$ *be a complete assignment, let* $\mathbf{X}$ *be a partial assignment, and let* $f_{\&g}$ *be an assignment monotonic three-valued oracle function. Then,* $f_{\&g}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$*,* $X \in \{\mathbf{T}, \mathbf{F}\}$*, implies for every assignment* $\mathbf{X}' \succeq \mathbf{X}$ *that* $f_{\&g}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$*.*

Proof. Let $\mathbf{A}$ be a complete assignment, $\mathbf{X}$ a partial assignment, and $f_{\&g}$ an assignment monotonic three-valued oracle function. Further let $f_{\&g}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$, where $X \in \{\mathbf{T}, \mathbf{F}\}$, and let $\mathbf{X}'$ be an arbitrary partial assignment s.t. $\mathbf{X}' \succeq \mathbf{X}$. We need to show that $f_{\&g}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$. First, we show that $\mathbf{A} \dot{\cup} \neg.\mathbf{X}' \succeq \mathbf{A} \dot{\cup} \neg.\mathbf{X}$. Recall that $\mathbf{A} \dot{\cup} \neg.\mathbf{X} = (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\}) \cup \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}\} \cup \{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X} \text{ and } \mathbf{T}a \in \mathbf{A}\}$, according to Definition 4.1. Since $\mathbf{X}' \succeq \mathbf{X}$, we have that $\{\mathbf{T}a \in \mathbf{X}\} \cup \{\mathbf{F}a \mid \mathbf{F}a \in \mathbf{X}\} \subseteq \mathbf{X}'$, according to the definition of "$\succeq$". Hence, we derive that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\} \succeq \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X}' \text{ or } \mathbf{U}a \in \mathbf{X}'\}$. It follows that $(\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X}' \text{ or } \mathbf{U}a \in \mathbf{X}'\}) \succeq (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\})$. It is also easy to see that $\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}'\} \succeq \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}\}$ and $\{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X}' \text{ and } \mathbf{T}a \in \mathbf{A}\} \succeq \{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X} \text{ and } \mathbf{T}a \in \mathbf{A}\}$. Consequently, we infer that $\mathbf{A} \dot{\cup} \neg.\mathbf{X}' \succeq \mathbf{A} \dot{\cup} \neg.\mathbf{X}$. Because we have that $f_{\&g}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$, and due to assignment monotonicity according to Definition 3.3, from $\mathbf{A} \dot{\cup} \neg.\mathbf{X}' \succeq \mathbf{A} \dot{\cup} \neg.\mathbf{X}$ it follows that $f_{\&g}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$. □

The proposition implies that early external evaluations during the unfounded set search w.r.t. an assignment $\mathbf{A} \dot{\cup} \neg.\mathbf{X}$ yield nogoods N s.t. $f_{\&e}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}', \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ for all extensions $\mathbf{X}'$ of $\mathbf{X}$. The fact that faithful io-nogoods added via the transformation $\mathcal{T}_\Omega$ to the encoding Ω_Π do not remove unfounded sets, as stated in Proposition 15 by Eiter, Fink, Krennwallner, et al. (2014), is based on the latter property.

We are now ready to present our new algorithm for detecting unfounded sets which also exploits learning w.r.t. partial assignments and is formalized by Algorithm 4.1. It extends the unfounded set check The algorithm is used in Part (d) of Algorithm 3.1 in order to check whether a compatible set $\mathbf{A}$ for a HEX-program Π is an answer set, i.e. its true part does not intersect with an unfounded set for Π w.r.t. $\mathbf{A}$.

Algorithm 4.1 receives as input a HEX-program Π, a complete assignment $\mathbf{A}$ representing a compatible set of Π, and a set ∇ of nogoods that have been generated by Algorithm 3.1. It returns *true* if Π has an unfounded set w.r.t. $\mathbf{A}$ that intersects with the true part of $\mathbf{A}$, and false otherwise, i.e. when $\mathbf{A}$ is an answer set of Π. At first, the assumptions $\mathcal{A}_\mathbf{A}$ and the transformations of io-nogoods already learned in the main search are added to the encoding Ω_Π. In our implementation, elements in $\mathcal{A}_\mathbf{A}$ are marked as assumptions and hence, they can be removed from the encoding without the need to reinitialize the SAT-solver completely.

Similar to Algorithm 3.1, Algorithm 4.1 explores the search space in one loop based on the well-known *CDCL procedure* (Marques-Silva et al., 2009), where unit propagation

is performed in Part (a), conflict learning and backjumping in Part (b), and guessing in Part (e). However, to take the semantics of external atoms into account, there are two additional parts integrated into the CDCL procedure, where the first is necessary to ensure correctness of the algorithm, while the second potentially increases its efficiency.

On the one hand, in Part (c), after a solution $\mathbf{S}$ to Ω'_Π has been found, it is checked for each $\&g[\vec{p}](\vec{c})$ in Π whether the truth value assigned to the replacement atom $e_{\&g[\vec{p}](\vec{c})}$ is compatible with the evaluation of the corresponding oracle function under $\mathbf{A} \dot\cup \neg.\mathbf{S}$. It has been shown that when the truth value of an external atom $\&g[\vec{p}](\vec{c})$ under $\mathbf{A}$ coincides with the one for $e_{\&g[\vec{p}](\vec{c})}$ assigned by $\mathbf{S}$, this check can be skipped (cf. Eiter, Fink, Krennwallner, et al., 2014).

If a solution $\mathbf{S}$ passes the external checks, an unfounded set for Π w.r.t. $\mathbf{A}$ has been detected and the algorithm returns *true* in case $\mathbf{S}$ intersects with the true part of the complete assignment $\mathbf{A}$; otherwise, $\mathbf{S}$ is added to Ω'_Π and the search continues. The io-nogoods learned from the external evaluations are added to the nogood store ∇ for use in the search for compatible sets and to Ω'_Π via the nogood transformation $\mathcal{T}_\Omega$, in order to avoid wrong guesses for replacement atoms in the further unfounded set search.

On the other hand, external evaluations can also be performed based on partial assignments for Ω'_Π, which are triggered by a heuristics in Part (d). Accordingly, the respective oracle function is evaluated in Part (c) under an assignment $\mathbf{A} \dot\cup \neg.\mathbf{S}$ as in Definition 4.1. As before, learned nogoods N are added to ∇ and (via the nogood transformation) to Ω'_Π, respectively.

While the details of how the learned nogoods N are constructed are not relevant in the following, we stress that using the unfounded set itself as learned nogood is in general *not* correct.

Example 4.3. Consider the program $\Pi = \{a \vee b \leftarrow; c \leftarrow b; b \leftarrow c\}$, which has two answer sets $\mathbf{A}_1 = \{\mathbf{T}a, \mathbf{F}b, \mathbf{F}c\}$ and $\mathbf{A}_2 = \{\mathbf{F}a, \mathbf{T}b, \mathbf{T}c\}$. Note that $U = \{b, c\}$ is an unfounded set of Π w.r.t. $\mathbf{A} = \{\mathbf{T}a, \mathbf{T}b, \mathbf{T}c\}$. Using $\{\mathbf{T}b, \mathbf{T}c\}$ as learned nogood (constructed from the atoms in U) would eliminate the answer set $\mathbf{A}_2$. Informally, this is the case because U is unfouneded *only w.r.t. the current assignment*, which must be respected in nogood learning. In this case, the nogood learned from U is $N = \{\mathbf{T}a, \mathbf{T}b\}$ (or, alternatively, $N' = \{\mathbf{T}a, \mathbf{T}c\}$). △

For details on the construction of N we refer to (Eiter, Fink, Krennwallner, et al., 2014). Note that Algorithm 4.1 is parametric on the learning function Λ used in Parts (c) and (d). Because the nogood transformation $\mathcal{T}_\Omega$ that has been employed by Eiter, Fink, Krennwallner, et al. (2014) can only be applied to faithful io-nogoods, we assume that Λ only returns faithful io-nogoods in Algorithm 4.1. That is, all other nogoods returned by the learning function Λ are simply ignored. In practice, we employ the learning functions Λ_u and Λ_{mu}.

4.1.3 Properties of the Algorithm

The following proposition, adapted from Theorem 10 in (Eiter, Fink, Krennwallner, et al., 2014), states that by the checks in Part (c) of Algorithm 4.1, we can determine whether

a *complete* solution $\mathbf{S}$ corresponds to an unfounded set for Π w.r.t. $\mathbf{A}$:

Proposition 4.2. *Let Π be a ground HEX-program and let $\mathbf{A}$ be a complete assignment over $A(\Pi)$. If there is a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ such that for all external atoms $\&g[\vec{p}](\vec{c})$ in Π it holds that*

(1) $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$ *and* $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.\mathbf{S} \not\models \&g[\vec{p}](\vec{c})$, *and*

(2) $\mathbf{F}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$ *and* $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.\mathbf{S} \models \&g[\vec{p}](\vec{c})$,

then $U = \{Xa \mid a \in A(\Pi), Xa \in \mathbf{S}, X \in \{\mathbf{T}, \mathbf{F}\}\}$ is an unfounded set for Π w.r.t. $\mathbf{A}$.

Proof. The proof is identical to the proof for Theorem 10 by Eiter, Fink, Krennwallner, et al. (2014) modulo the different representation of assignments by sets of ground atoms used by Eiter, Fink, Krennwallner, et al. (2014). □

Moreover, we can show that for every unfounded set U for Π w.r.t. $\mathbf{A}$ where U intersects with the true part of $\mathbf{A}$, a solution to Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ can be generated that passes the checks in Part (c) of Algorithm 4.1, and that the nogoods added in Part (d) of Algorithm 4.1 do not eliminate the solution:

Proposition 4.3. *Let Π be a ground HEX-program, let $\mathbf{A}$ be a complete assignment over $A(\Pi)$ and suppose Algorithm 4.1 is executed with Π and $\mathbf{A}$ as inputs. If there is an unfounded set U for Π w.r.t. $\mathbf{A}$ s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$, then there is a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$, s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$, that satisfies conditions (1) and (2) of Proposition 4.2 and all transformed nogoods $\mathcal{T}_\Omega(N)$ added to Ω'_Π in Part (d) of Algorithm 4.1.*

Proof. Let Π be a ground HEX-program, let $\mathbf{A}$ be a complete assignment over $A(\Pi)$ and suppose Algorithm 4.1 is executed with Π and $\mathbf{A}$ as inputs. Further, let there be an unfounded set U for Π w.r.t. $\mathbf{A}$ s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$. According to Proposition 8 by Eiter, Fink, Krennwallner, et al. (2014), there is a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$. In addition, it follows directly from Proposition 11 in (Eiter, Fink, Krennwallner, et al., 2014) that $\mathbf{S}$ satisfies conditions (1) and (2) of Proposition 4.2.

It is easy to see that any faithful io-nogood as defined in Definition 3.5 is also a *valid input-output-relationship* according to Definition 9 in (Eiter, Fink, Krennwallner, et al., 2014). Moreover, we only consider faithful io-nogoods returned by the learning function Λ. Consequently, we infer that $\mathbf{S}$ also satisfies all transformed nogoods $\mathcal{T}_\Omega(N)$ added to Ω'_Π in Part (d) of Algorithm 4.1 according to Proposition 15 by Eiter, Fink, Krennwallner, et al. (2014). □

We remark that in case the learning function Λ_u is used in Part (d), backjumping is triggered by the added nogoods as soon as it can be determined that a partial assignment cannot be extended to a solution satisfying conditions (1) and (2). However, we refrain from a formal statement and proof of this behavior in the special case, as it would require to delve into the very details of the uniform encoding and the particular nogood transformation (the respective conflict involves a transformed nogood).

Example 4.4. Consider the HEX-program $\Pi = \{r \leftarrow \&id[q](). \; ; \; q \leftarrow . \; ; \; p \leftarrow \&id[p]().\}$, the complete assignment $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ and a partial assignment $\mathbf{S}$ s.t. $\mathbf{S} \supseteq \{\mathbf{F}e_{\&id[\vec{q}]}(), \mathbf{F}e_{\&id[\vec{r}]}(), \mathbf{T}r, \mathbf{F}q, \mathbf{U}p\}$. Note that $\mathbf{S}$ cannot be extended s.t. it corresponds to an unfounded set of Π w.r.t. $\mathbf{A}$. Accordingly, by performing external evaluations w.r.t. $\mathbf{S}$, we find that it violates condition (2) of Proposition 4.2 as $\mathbf{F}e_{\&id[q]}() \in \mathbf{S}$, $\mathbf{A} \models \&id[q]()$ and $f_{\&id}(\mathbf{A} \dot{\cup} \neg.\mathbf{S}, q) = \mathbf{T}$. This demonstrates that we can detect that $\mathbf{S}$ cannot be extended to a solution corresponding to an unfounded set without constructing a complete solution to Ω'_Π. △

Correctness and completeness of Algorithm 4.1 can be derived from the facts that it returns only solutions for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ that satisfy conditions (1) and (2) of Proposition 4.2, and that no such solution is removed due to the nogoods learned in Part (d).

Theorem 4.1. *Given a ground* HEX*-program* Π *and a complete assignment* $\mathbf{A}$ *over* $A(\Pi)$ *as inputs, Algorithm 4.1 returns true if there is an unfounded set* U *for* Π *w.r.t.* $\mathbf{A}$ *s.t.* $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$*, and false otherwise.*

Proof. Let Π be a ground HEX-program, let $\mathbf{A}$ be a complete assignment over $A(\Pi)$ and suppose Algorithm 4.1 is executed with Π and $\mathbf{A}$ as inputs.

We first show that Algorithm 4.1 returns *true* if there is an unfounded set U for Π w.r.t. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$. Consider the case that there is an unfounded set U for Π w.r.t. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$. According to Proposition 4.3, a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ exists, such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$, which does not violate any nogood added to Ω'_Π in Part (d) of Algorithm 4.1. Consequently, the complete assignment $\mathbf{S}$ is generated by Algorithm 4.1 and Part (c) is executed. Since $\mathbf{S}$ satisfies conditions (1) and (2) of Proposition 4.2, according to Proposition 4.3, the variable *isUFS* is not set to *false* in Part (c), and because $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$ the algorithm returns *true*.

Now we show that Algorithm 4.1 returns *false* if there is no unfounded set U for Π w.r.t. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$. Towards a contradiction, suppose that there is no unfounded set U for Π w.r.t. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$ and that Algorithm 4.1 does not return *false*. This means that *false* is not returned in Part (b) because *true* is returned before the search space has been completely explored. Accordingly, a complete assignment $\mathbf{S}$ is generated by Algorithm 4.1 and Part (c) is executed, which satisfies conditions (1) and (2) of Proposition 4.2. Moreover, it must hold that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$ because otherwise *true* would not be returned in Part (c). However, from Proposition 4.2 we know that $U = \{Xa \,|\, a \in A(\Pi), Xa \in \mathbf{S}, X \in \{\mathbf{T}, \mathbf{F}\}\}$ is an unfounded set for Π w.r.t. $\mathbf{A}$. Since we have that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$, we have that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$ and hence, we infer that there is an unfounded set U for Π w.r.t. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$. Thus, we derive a contradiction, and infer that Algorithm 4.1 returns *false* if there is no unfounded set U for Π w.r.t. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U\} \neq \emptyset$. □

Thus, by employing Algorithm 4.1, we are now also able to exploit partial assignments for evaluating external sources at any point during the unfounded set search, and for learning corresponding io-nogoods that can decrease the number of unfounded set candidates which need to be generated.

4.2 Skipping the Minimality Check Based on Semantic Dependencies

Since an answer set $\hat{\mathbf{A}}$ of a guessing program $\hat{\Pi}$ must be a minimal model of the FLP-reduct $f\hat{\Pi}^{\hat{\mathbf{A}}}$, an e-minimality check is under certain conditions redundant. The criterion by Eiter, Fink, Krennwallner, et al. (2014) for deciding its necessity relies on an atom dependency graph induced by the HEX-program. Informally, an e-minimality check is only needed for programs that allow cyclic support via external atoms, which can be checked efficiently. For instance, the program $\Pi_1 = \{ p \leftarrow \&id[p]() \}$ allows cyclic support for the atom p via $\&id[p]()$, while this is not the case for $\Pi_2 = \{ p \leftarrow \&id[q]();\ q \leftarrow r;\ r \leftarrow q \}$, where the truth value of $\&id[q]()$ is independent of the value of p. If cyclic support via external atoms can be ruled out as for Π_2, the e-minimality check can be skipped for a program, potentially avoiding to invest many resources into a redundant check. Note, however, that an ordinary minimality check is still needed for computing the answer sets of $\hat{\Pi}$.

In this section, we introduce a new technique for skipping the e-minimality check w.r.t. a wider class of programs than previous approaches. More precisely, given Π, we present a new sufficient[1] criterion for deciding if every projection $\mathbf{A}$ of a compatible set $\hat{\mathbf{A}}$ for $\hat{\Pi}$ is an answer set of Π. The criterion exploits that output values of external atoms often do not depend on the complete extensions of their input predicates, which can be determined given additional information concerning dependencies between the inputs and outputs of external atoms.

4.2.1 Dependency Graph Pruning

We start by defining so-called *io-dependencies*, which specify that certain outputs of external atoms only depend on specific argument values of their inputs. For instance, whether a city c is in the output of $\&closeWest[city](X)$ from the beginning of this chapter only depends on cities c' that are located close to the east of c. Hence, the truth value of $\&closeWest[city](kobe)$ clearly only depends on the atom $city(osaka)$, and we want to encode that *kobe* as first output of $\&closeWest[city](X)$ only depends on the element *osaka* as first argument of the first input predicate *city*.

Definition 4.2 (Io-Dependency)**.** *An* io-dependency *for a ge-predicate* $\&g[\vec{p}]$ *is a tuple* $\delta = \langle i, j : J, k : e \rangle$ *where* $1 \leq i \leq ar_I(\&g)$, $1 \leq j \leq ar(p_i)$, $1 \leq k \leq ar_O(\&g)$, $J \subseteq \mathcal{C}$ *and* $e \in \mathcal{C}$*. The set of all* δ *for* $\&g[\vec{p}]$ *is denoted by* $dep(\&g[\vec{p}])$.

[1] Deciding the sufficient *and necessary* criterion is Π_2^p-complete for polynomial-time decidable external atoms and thus ill-suited for our aim to improve performance.

In the sequel, io-dependencies will be used to constrain the possible dependencies between inputs and outputs of external atoms $\&g[\vec{p}](\vec{c})$. Intuitively, an io-dependency $\langle i, j : J, k : e\rangle$ states that if constant e occurs as the k^{th} output of $\&g[\vec{p}](\vec{c})$, then only those input predicates at position i are relevant for its evaluation where the j^{th} argument matches some $e' \in J$. Thus, the io-dependency $\delta = \langle 1, 1 : \{osaka\}, 1 : kobe\rangle$ could be specified for the example above. Io-dependencies induce atom sets relevant for evaluating respective external atoms:

Definition 4.3 (Compliant Atoms). *A ground ordinary atom $p_i(\vec{d})$, with $\vec{d} = d_1, ..., d_l$, is* compliant *with a set $D \subseteq dep(\&g[\vec{p}])$ of io-dependencies for a ground external atom $\&g[\vec{p}](\vec{c})$ if $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D$ with $e = c_k$. The set of all atoms compliant with D for $\&g[\vec{p}](\vec{c})$ is denoted by $comp(D, \&g[\vec{p}](\vec{c}))$.*

For our example, we have $comp(\{\delta\}, \&closeWest[city](kobe))=\{city(osaka)\}$. The semantics of external atoms is related to io-dependencies as follows.

Definition 4.4 (Faithfulness). *A set $D \subseteq dep(\&g[\vec{p}])$ is* faithful, *if for any assignments $\mathbf{A}, \mathbf{A}'$ and ground external atom $\&g[\vec{p}](\vec{c})$, either $\mathbf{A}(p_i(\vec{d})) \neq \mathbf{A}'(p_i(\vec{d}))$ for some $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$ or $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.*

Thus, io-dependencies $D \subseteq dep(\&g[\vec{p}])$ constrain the set of atoms that potentially impact the evaluation of $\&g[\vec{p}](\vec{c})$, i.e. if D is faithful, changing only truth values of atoms $p_i(\vec{d})$ that are not in $comp(D, \&g[\vec{p}](\vec{c}))$ has no effect on the value of $\&g[\vec{p}](\vec{c})$.

In the following, we denote by $D(\&g[\vec{p}]) \subseteq dep(\&g[\vec{p}])$ a set of io-dependencies specified for $\&g[\vec{p}]$. By default, we assume that $D(\&g[\vec{p}])$ is empty, but it can be utilized to supply additional dependency information. To ensure correctness of an algorithm that skips e-minimality checks based on $D(\&g[\vec{p}])$, it is important that $D(\&g[\vec{p}])$ is faithful; and we assume in the following that this is the case. Simultaneously, the goal is to approximate the real dependencies between atoms as close as possible for maximal performance gains. Note that while an extensional specification of $D(\&g[\vec{p}])$ might be very verbose, they can often also be specified more concisely in an intensional manner, as in the following example.

Example 4.5. Consider $\&setDiff[dom, set](c)$, which is true for $c \in \mathcal{C}$ and assignment $\mathbf{A}$ iff $\{\mathbf{T}dom(c), \mathbf{F}set(c)\} \subseteq \mathbf{A}$. Thus, the presence of an output value c only depends on atoms with predicate *dom* or *set* that have c as first argument. Hence, $D(\&setDiff[dom, set]) = \{\langle 1, 1{:}\{c\}, 1{:}c\rangle, \langle 2, 1{:}\{c\}, 1{:}c\rangle \mid c \in \mathcal{C}\}$ is faithful. △

We now introduce a notion of atom dependency in HEX-programs that accounts for io-dependencies and generalizes the corresponding notion from (Eiter, Fink, Krennwallner, et al., 2014).

Definition 4.5 (Atom Dependency). *Given a ground HEX-program Π, a set $D(\&g[\vec{p}])$ for each $\&g[\vec{p}]$ in Π, and ordinary ground atoms $p(\vec{d})$ and $q(\vec{e})$, we say*

- *$q(\vec{e})$* depends on *$p(\vec{d})$, denoted $q(\vec{e}) \rightarrow_d p(\vec{d})$ if for some rule $r \in \Pi$ it holds that $q(\vec{e}) \in H(r)$ and $p(\vec{d}) \in B^+(r)$; and*

- $q(\vec{e})$ depends externally on $p(\vec{d})$, *denoted* $q(\vec{e}) \rightarrow_e p(\vec{d})$ *if some rule* $r \in \Pi$ *and some external atom* $\&g[\vec{p}](\vec{c}) \in B^+(r) \cup B^-(r)$ *with* $p \in \vec{p}$ *exist such that* $q(\vec{e}) \in H(r)$ *and* $p(\vec{d}) \in comp(D(\&g[\vec{p}]), \&g[\vec{p}](\vec{c}))$.

Note that Definition 4.5 generalizes the corresponding one from (Eiter, Fink, Krennwallner, et al., 2014) in that an external dependency is only added if the specified io-dependencies are satisfied. The definitions coincide if $D(\&g[\vec{p}]) = \emptyset$ for all ge-predicates $\&g[\vec{p}]$ in Π.

Example 4.6. Consider $\&suc[node](n)$, which evaluates to true w.r.t. an assignment $\mathbf{A}$ and an external directed graph $\mathcal{G} = (V, E)$ iff $n' \rightarrow n \in E$ for some node n' s.t. $\mathbf{T}node(n) \in \mathbf{A}$. It is utilized in the following HEX-program Π:

$$node(a).\ \ node(X) \leftarrow\ \ \&suc[node](X).$$

Intuitively, the program computes all nodes reachable from node a via the edges in $\mathcal{G}$. If the external graph has nodes $V = \{a, b, c, d\}$ and directed edges $E = \{a \rightarrow b, a \rightarrow c, c \rightarrow d, e \rightarrow d\}$, the grounding of Π produced by the grounding algorithm of the HEX-program solver DLVHEX contains the following rules (omitting facts):

$$node(b) \leftarrow \&suc[node](b).\ \ node(c) \leftarrow \&suc[node](c).\ \ node(d) \leftarrow \&suc[node](d).$$

Without specifying io-dependencies for $\&suc[node]$, it holds, e.g., that $node(a) \rightarrow_e node(b)$ and $node(b) \rightarrow_e node(a)$. However, we can specify $D(\&suc[node]) = \{\langle 1, 1 : \{c_1 \mid c_1 \rightarrow c_2 \in E\}, 1 : c_2\rangle \mid c_2 \in \mathcal{C}\}$, exploiting that the presence of output nodes only depends on input nodes to which they are successors. In this case, $node(a) \rightarrow_e node(b)$ does not hold according to Definition 4.5 as $b \rightarrow a \notin E$. △

The following lemma states that external dependencies according to Definition 4.5 still cover all atoms relevant for deciding the truth value of an external atom, which follows directly from the definition of faithful io-dependencies.

Lemma 4.1. *Let* Π *be a* HEX*-program, let* r *be a rule in* Π*, and let* $\mathbf{A}$ *and* $\mathbf{A}'$ *be two assignments. If* $a \in H(r)$ *and* $\&g[\vec{q}](\vec{c}) \in B^+(r) \cup B^-(r)$*, then* $\mathbf{A} \models \&g[\vec{q}](\vec{c})$ *iff* $\mathbf{A}' \models \&g[\vec{q}](\vec{c})$ *given that* $\mathbf{A}(q(\vec{e})) = \mathbf{A}'(q(\vec{e}))$ *for all atoms* $q(\vec{e})$ *where* $a \rightarrow_e q(\vec{e})$.

Proof. Let Π be a HEX-program, let r be a rule in Π, and let $\mathbf{A}$ and $\mathbf{A}'$ be two assignments. Furthermore, let $a \in H(r)$ and $\&g[\vec{q}](\vec{c}) \in B^+(r) \cup B^-(r)$, and suppose that $\mathbf{A}(q(\vec{e})) = \mathbf{A}'(q(\vec{e}))$ holds for all atoms q where $a \rightarrow_e q(\vec{e})$. According to Definition 4.5, for all $q(\vec{e})$ where $a \rightarrow_e q(\vec{e})$ it holds that $q(\vec{e}) \in comp(D(\&g[\vec{q}]), \&g[\vec{q}](\vec{c}))$. Furthermore, it holds that $a \rightarrow_e q(\vec{e})$ for every $q(\vec{e}) \in comp(D(\&g[\vec{q}]), \&g[\vec{q}](\vec{c}))$ because $q(\vec{e}) \in comp(D(\&g[\vec{q}]), \&g[\vec{q}](\vec{c}))$ is only possible if $q \in \vec{q}$. Hence, $\mathbf{A} \models \&g[\vec{q}](\vec{c})$ iff $\mathbf{A}' \models \&g[\vec{q}](\vec{c})$ follows by Definition 4.4, and since we assume all sets of io-dependencies to be faithful w.r.t. the given HEX-program. □

We are now ready to introduce the atom dependency graph for a given program Π. From this graph, a property of Π can be derived which is subsequently employed to decide the necessity of the e-minimality check w.r.t. Π.

Definition 4.6 (Dependency Graph)**.** *Given a ground* HEX*-program* Π*, the* dependency graph $\mathcal{G}_\Pi^{dep} = (V, E)$ *has the vertices* $V = A(\Pi)$ *and directed edges* $E = \rightarrow_d \cup \rightarrow_e$*;* Π *has an* e-cycle*, if* $\mathcal{G}_\Pi^{dep}$ *has a cycle with an edge* $\rightarrow_e$*.*

In the remainder of this subsection, our goal is to show that the e-minimality check can be skipped for all those HEX-programs that do not have an e-cycle. Notably, while the inverse of $\rightarrow_d$ was additionally included in $\mathcal{G}_\Pi^{dep}$ by Eiter et al. (Eiter, Fink, Krennwallner, et al., 2014), we improve their results by showing that our more general definition suffices.

For proving the main result of this section, i.e. correctness of our new decision criterion, we first need to introduce two lemmas from (Eiter, Fink, Krennwallner, et al., 2014), which are used subsequently in the proof of Theorem 4.2; and the concept of a *cut*, which intuitively represents a set of atoms that do not belong to the *core* of an unfounded set. The following definition is a generalization of Definition 13 by Eiter, Fink, Krennwallner, et al. (2014):

Definition 4.7 (Cut)**.** *Let* U *be an unfounded set of* Π *w.r.t.* $\mathbf{A}$*. A set of atoms* $C \subseteq \{a \mid \mathbf{T}a \in U\}$ *is a* cut *of* $\mathcal{G}_\Pi^{dep}$*, if*

(i) $b \not\rightarrow_e a$*, for all* $a \in C$ *and* $b \in \{a \mid \mathbf{T}a \in U\}$ *(*C *has no incoming e-edge from* U*),*

(ii) $b \not\rightarrow_d a$*, for all* $a \in C$ *and* $b \in \{a \mid \mathbf{T}a \in U\} \setminus C$ *(there are no ordinary edges* $\rightarrow_d$ *from* $\{a \mid \mathbf{T}a \in U\} \setminus C$ *to* C*).*

Next, we reproduce the lemmas and the according proofs from (Eiter, Fink, Krennwallner, et al., 2014) (corresponding to Lemmas 18 and 19 there). The proofs of the lemmas are modifications of the according proofs by Eiter, Fink, Krennwallner, et al. (2014), which account for our more general definitions of cuts and external dependencies.

The first lemma states that atoms in a cut of an unfounded set can be removed while the resulting set still constitutes an unfounded set.

Lemma 4.2. *Let* U *be an unfounded set of* Π *w.r.t.* $\mathbf{A}$*, and let* C *be a cut. Then,* $Y = \{a \mid \mathbf{T}a \in U\} \setminus C$ *is an unfounded set of* Π *w.r.t.* $\mathbf{A}$*.*

Proof. If $Y = \emptyset$, then the result holds trivially. Otherwise, let $r \in \Pi$ with $H(r) \cap Y \neq \emptyset$. Observe that $H(r) \cap \{a \mid \mathbf{T}a \in U\} \neq \emptyset$ because $\{a \mid \mathbf{T}a \in U\} \supseteq Y$. Since U is an unfounded set of Π w.r.t. $\mathbf{A}$, according to Definition 2.6, either

(i) $\mathbf{A} \not\models b$ for some $b \in B(r)$; or

(ii) $\mathbf{A} \dot\cup \neg.U \not\models b$ for some $b \in B(r)$; or

(iii) $\mathbf{A} \models h$ for some $h \in H(r) \setminus \{a \mid \mathbf{T}a \in U\}$.

In case (i), the condition also holds w.r.t. Y. In case (ii), let $a \in H(r)$ such that $a \in Y$, and $b \in B(r)$ such that $\mathbf{A} \dot\cup \neg.U \not\models b$. We make a case distinction: either b is an ordinary literal or an external one.

If b is an ordinary default-negated atom not c, then $\mathbf{A} \dot\cup \neg.U \not\models b$ implies $\mathbf{T}c \in \mathbf{A}$ and $c \notin \{a \mid \mathbf{T}a \in U\}$, and therefore also $\mathbf{A} \dot\cup \neg.Y \not\models b$. So assume b is an ordinary atom. If $b \notin \{a \mid \mathbf{T}a \in U\}$ then $\mathbf{A} \not\models b$ and case (i) applies, so assume $b \in \{a \mid \mathbf{T}a \in U\}$. Because $a \in H(r)$ and $b \in B(r)$, we have $a \to_d b$. By assumption $a \in Y$, and therefore $b \in Y$ because there are no ordinary edges $\to_d$ from Y to C according to Definition 4.7. Hence $\mathbf{A} \dot\cup \neg.Y \not\models b$.

If b is an external literal, then there is no $q \in \{a \mid \mathbf{T}a \in U\}$ with $a \to_e q$ and $q \notin Y$. Otherwise, this would imply $q \in C$ and C would have an incoming e-edge, which contradicts the assumption that C is a cut. Hence, for all $q \in \{a \mid \mathbf{T}a \in U\}$ with $a \to_e q$, also $q \in Y$, and therefore the truth value of b under $\mathbf{A} \dot\cup \neg.U$ and $\mathbf{A} \dot\cup \neg.Y$ is the same, according to Lemma 4.1. Hence $\mathbf{A} \dot\cup \neg.Y \not\models b$.

In case (iii), then also $\mathbf{A} \models h$ for some $h \in H(r) \setminus Y$ because $Y \subseteq \{a \mid \mathbf{T}a \in U\}$ and therefore $H(r) \setminus Y \supseteq H(r) \setminus \{a \mid \mathbf{T}a \in U\}$. □

The second lemma states that for each unfounded set U of a HEX-program Π, U is detected before Part (d) of Algorithm 3.1 when $\hat\Pi$ is evaluated if no input to some external atom is contained in U.

Lemma 4.3. *Let U be an unfounded set of Π w.r.t. $\mathbf{A}$. If there are no $x, y \in \{a \mid \mathbf{T}a \in U\}$ such that $x \to_e y$, then U is an unfounded set of $\hat\Pi$ w.r.t. $\hat{\mathbf{A}}$.*

Proof. If $\{a \mid \mathbf{T}a \in U\} = \emptyset$, then the result holds trivially. Otherwise, suppose $\hat r \in \hat\Pi$ and $H(\hat r) \cap \{a \mid \mathbf{T}a \in U\} \neq \emptyset$. Let $a \in H(\hat r) \cap \{a \mid \mathbf{T}a \in U\}$. Observe that $\hat r$ cannot be an external atom guessing rule because $\{a \mid \mathbf{T}a \in U\}$ contains only ordinary atoms. We show that one of the conditions in Definition 2.6 holds for $\hat r$ w.r.t. $\hat{\mathbf{A}}$.

Because $\hat r$ is no external atom guessing rule, there is a corresponding rule $r \in \Pi$ containing external atoms in place of replacement atoms. Because U is an unfounded set of Π and $H(r) = H(\hat r)$, according to Definition 2.6, either:

(i) $\mathbf{A} \not\models b$ for some $b \in B(r)$; or

(ii) $\mathbf{A} \dot\cup \neg.U \not\models b$ for some $b \in B(r)$; or

(iii) $\mathbf{A} \models h$ for some $h \in H(r) \setminus \{a \mid \mathbf{T}a \in U\}$.

In case (i), let $b \in B(r)$ such that $\mathbf{A} \not\models b$ and $\hat b$ the corresponding literal in $B(\hat b)$ (which is the same if b is ordinary and the corresponding replacement literal if b is external). Then also $\hat{\mathbf{A}} \not\models \hat b$ because $\hat{\mathbf{A}}$ is compatible.

In case (ii), we make a case distinction: either b is ordinary or external.

If b is ordinary, then $b \in B(\hat r)$ and $\mathbf{A} \dot\cup \neg.U \not\models b$ holds because $\mathbf{A}$ and $\hat{\mathbf{A}}$ are equivalent for ordinary atoms. If b is an external atom $\&g[\vec q](\vec c)$ or a default-negated external atom not $\&g[\vec q](\vec c)$, then $q(\vec e) \notin \{a \mid \mathbf{T}a \in U\}$ for all atoms $q(\vec e)$ where $a \to_e q(\vec e)$; otherwise we would have a contradiction to our assumption that $\{a \mid \mathbf{T}a \in U\}$ has no internal e-edges. Hence, we have that $\mathbf{A} \dot\cup \neg.U(q(\vec e)) = \mathbf{A}(q(\vec e))$ for all atoms $q(\vec e)$ where $a \to_e q(\vec e)$. But then $\mathbf{A} \dot\cup \neg.U \not\models b$ implies $\mathbf{A} \not\models b$ according to Lemma 4.1. Therefore we can apply case (i).

In case (iii), also $\hat{\mathbf{A}} \models h$ for some $h \in H(\hat{r}) \setminus \{a \mid \mathbf{T}a \in U\}$ because $H(r) = H(\hat{r})$ contains only ordinary atoms and $\mathbf{A}$ is equivalent to $\hat{\mathbf{A}}$ for ordinary atoms. □

The following theorem represents the main result of this section, and differs from the previous result for e-minimality check skipping by Eiter, Fink, Krennwallner, et al. (2014) in that it is based on our generalized definition of external dependencies. Consequently, it can be applied to a larger class of HEX-programs.

Theorem 4.2. *If a ground HEX-program Π contains no e-cycle, then every projection $\mathbf{A}$ of a compatible set $\hat{\mathbf{A}}$ for $\hat{\Pi}$ is an answer set of Π.*

Proof. The core of the following proof mirrors exactly the proof for Theorem 20 by Eiter et al. (Eiter, Fink, Krennwallner, et al., 2014), but it has been adapted to our more general definition of the atom dependency graph $\mathcal{G}_\Pi^{dep}$.

Since the answer sets of a HEX-program Π are exactly the projections $\mathbf{A}$ of compatible sets $\hat{\mathbf{A}}$ where $\{a \mid \mathbf{T}a \in \mathbf{A} \cap U\} = \emptyset$ for every unfounded set U of Π w.r.t. $\mathbf{A}$ (cf. Section 2.4), we can prove the theorem by showing that the following statement holds:

If a ground HEX-program Π contains no e-cycle, and no unfounded set U of $\hat{\Pi}$ w.r.t. an assignment $\hat{\mathbf{A}}$ s.t. $\{a \mid \mathbf{T}a \in \hat{\mathbf{A}} \cap U\} \neq \emptyset$ exists, then no unfounded set U' of Π w.r.t. $\mathbf{A}$ such that $\{a \mid \mathbf{T}a \in \mathbf{A} \cap U'\} \neq \emptyset$ exists, where $\mathbf{A}$ is the projection of $\hat{\mathbf{A}}$.

We prove the contrapositive. Let Π be a HEX-program that contains no e-cycle, and let $\mathbf{A}$ be the projection of an assignment $\hat{\mathbf{A}}$ s.t. an unfounded set U' for Π w.r.t. $\mathbf{A}$ s.t. $\{a \mid \mathbf{T}a \in \mathbf{A} \cap U'\} \neq \emptyset$ exists. Since U' is an unfounded set for Π w.r.t. $\mathbf{A}$ s.t. $\{a \mid \mathbf{T}a \in \mathbf{A} \cap U'\} \neq \emptyset$, the set $U'' = \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap U'\}$ is also a nonempty unfounded set for Π w.r.t. $\mathbf{A}$. The previous holds because no atom in $\{a \mid \mathbf{T}a \in \mathbf{A}\} \setminus \{a \mid \mathbf{T}a \in U'\}$ is true under $\mathbf{A}$ anyway and hence, conditions (i) to (iii) from the definition of unfounded sets must be satisfied w.r.t. $\{a \mid \mathbf{T}a \in \mathbf{A} \cap U'\}$ as well. We need to show that an unfounded set U of $\hat{\Pi}$ w.r.t. $\hat{\mathbf{A}}$ s.t. $\{a \mid \mathbf{T}a \in \hat{\mathbf{A}} \cap U\} \neq \emptyset$ exists as well.

Let $\leftarrow_d$ be the inverse of $\rightarrow_d$. We define the *reachable set* $R(a)$ from some atom a as

$$R(a) = \{b \mid (a, b) \in \leftarrow_d^*\},$$

i.e. the set of atoms $b \in \{a \mid \mathbf{T}a \in U\}$ reachable from a using edges from $\leftarrow_d$ only but no e-edges.

We first assume that $\{a \mid \mathbf{T}a \in U''\}$ contains at least one e-edge, i.e. there are $x, y \in \{a \mid \mathbf{T}a \in U''\}$ such that $x \rightarrow_e y$. Now we show that there is a $u \in \{a \mid \mathbf{T}a \in U''\}$ with outgoing e-edge (i.e. $u \rightarrow_e v$ for some $v \in \{a \mid \mathbf{T}a \in U''\}$), but such that $R(u)$ has no incoming e-edges (i.e. for all $v \in R(u)$ and $b \in \{a \mid \mathbf{T}a \in U''\}$, $b \not\rightarrow_e v$ holds). Suppose to the contrary that for all a with outgoing e-edges, the reachable set $R(a)$ has an incoming e-edge. We now construct an e-cycle under $\rightarrow_d \cup \rightarrow_e$, which contradicts our assumption. Start with an arbitrary node with an outgoing e-edge $c_0 \in \{a \mid \mathbf{T}a \in U''\}$ and let p_0 be the (possibly empty) path (under $\leftarrow_d$) from c_0 to the node $d_0 \in R(c_0)$ such that d_0 has an incoming e-edge, i.e. there is a c_1 such that $c_1 \rightarrow_e d_0$; note that

Figure 4.1: Full and pruned dependency graph for Π from Example 4.6 (all arrows are "$\to_e$")

$c_1 \notin R(c_0)$[2]. By assumption, also some node d_1 in $R(c_1)$ has an incoming e-edge (from some node $c_2 \notin R(c_1)$). Let p_1 be the path from c_1 to d_1, etc. By iteration we can construct the concatenation of the paths $p_0, (d_0, c_1), p_1, (d_1, c_2), p_2, \ldots, p_i, (d_i, c_{i+1}), \ldots$, where the p_i from c_i to d_i are the paths within reachable sets, and the (d_i, c_{i+1}) are the e-edges between reachable sets. However, as $\{a \mid \mathbf{T}a \in U''\}$ is finite, some nodes on this path must be equal, i.e., a prefix of the constructed sequence represents an e-cycle (in reverse order).

This proves that u is a node with outgoing e-edge but such that $R(u)$ has no incoming e-edges. We next show that $R(u)$ is a cut. Condition (i) of Definition 4.7 is immediately satisfied by definition of u. Condition (ii) is shown as follows. Let $u' \in R(u)$ and $v' \in \{a \mid \mathbf{T}a \in U''\} \setminus R(u)$. We have to show that $v' \not\to_d u'$. Suppose, towards a contradiction, that $v' \to_d u'$. Because of $u' \in R(u)$, there is a path from u to u' under $\leftarrow_d$. But if $v' \to_d u'$, then there would also be a path from u to v' under $\leftarrow_d$ and v' would be in $R(u)$, a contradiction.

Therefore, $R(u)$ is a cut of U'', and by Lemma 4.2, it follows that $\{a \mid \mathbf{T}a \in U''\} \setminus R(u)$ is an unfounded set. Observe that $\{a \mid \mathbf{T}a \in U''\} \setminus R(u)$ contains one e-edge less than $\{a \mid \mathbf{T}a \in U''\}$ because u has an outgoing e-edge. Further observe that $\{a \mid \mathbf{T}a \in U''\} \setminus R(u) \neq \emptyset$ because there is a $w \in \{a \mid \mathbf{T}a \in U''\}$ such that $u \to_e w$ but $w \notin R(u)$. By iterating this argument, the number of e-edges in the unfounded set can be reduced to zero in a nonempty core. Eventually, Lemma 4.3 applies, proving that the remaining set is an unfounded set of $\hat{\Pi}$. Hence, we infer that there is a nonempty unfounded set U of $\hat{\Pi}$ w.r.t. $\hat{\mathbf{A}}$. Moreover, we have that $\{a \mid \mathbf{T}a \in U\} \subseteq \{a \mid \mathbf{T}a \in U''\}$ by construction of U and since $\{a \mid \mathbf{T}a \in U''\} \subseteq \{a \mid \mathbf{T}a \in \mathbf{A}\}$, we also have that $\{a \mid \mathbf{T}a \in \hat{\mathbf{A}} \cap U\} \neq \emptyset$. □

Example 4.7 (cont'd). Figure 4.1 shows the dependency graphs for Π from Example 4.6, with and without specified io-dependencies. The full dependency graph has an e-cycle, but the pruned graph does not. Hence, Π does not require e-minimality checks (cf. Theorem 4.2), but this can only be detected using the pruned graph. △

As a result, we obtain a flexible means for increasing the efficiency of evaluating a class of HEX-programs where the e-minimality check is performed due to an overapproximation of the real dependencies between atoms.

[2]Whenever $x \to_e y$ for $x, y \in \{a \mid \mathbf{T}a \in U''\}$, then there is no path from x to y under $\leftarrow_d$, because otherwise we would have an e-cycle under $\to_d \cup \to_e$.

4.2.2 Properties of Faithful Io-Dependencies

We now consider checking, generating and optimizing io-dependencies.

Informally, given $D_1, D_2 \subseteq dep(\&g[\vec{p}])$, D_1 is better than D_2 if it induces less compliant atoms. We thus say that D_1 *tightens* D_2, denoted $D_1 \leq D_2$, if $comp(D_1, \&g[\vec{p}](\vec{c})) \subseteq comp(D_2, \&g[\vec{p}](\vec{c}))$ holds for all tuples $\vec{c}$. We call D_1 *tight* if no D_2 strictly tightens D_1, i.e., $D_2 \leq D_1$ but $D_1 \not\leq D_2$; furthermore D_1 and D_2 are *equally tight*, denoted $D_1 \equiv D_2$, if $D_1 \leq D_2$ and $D_2 \leq D_1$. We then have:

Proposition 4.4. *Suppose $D_1, D_2 \subseteq dep(\&g[\vec{p}])$ are such that $D_1 \leq D_2$. If D_1 is faithful, then D_2 is also faithful.*

Proof. Towards a contradiction, suppose that $D_1 \leq D_2$ and D_1 is faithful, but D_2 is not. Then there exist assignments $\mathbf{A}, \mathbf{A}'$ and a tuple $\vec{c}$ such that $\mathbf{A}$ and $\mathbf{A}'$ coincide on $comp(D_2, \&g[\vec{p}](\vec{c}))$ but $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$. Because it holds that $comp(D_1, \&g[\vec{p}](\vec{c})) \subseteq comp(D_2, \&g[\vec{p}](\vec{c}))$, it follows that $\mathbf{A}$ and $\mathbf{A}'$ coincide also on $comp(D_1, \&g[\vec{p}](\vec{c}))$; but this contradicts that D_1 is faithful. □

As a consequence, faithfulness is anti-monotonic w.r.t. set-inclusion, and it is monotonic w.r.t. adding subsumed io-dependencies, where $\delta = \langle i, j : J, k : e\rangle$ *subsumes* $\delta' = \langle i, j : J', k : e\rangle$, if $J \subseteq J'$ holds.

Corollary 4.1. *If $D \subseteq dep(\&g[\vec{p}])$ is faithful, then (i) each $D' \subseteq D$ is faithful and (ii) each $D' = D \cup D''$ where each $\delta'' \in D''$ is subsumed by some $\delta \in D$ is faithful.*

Proof. As for (i), that $D' \subseteq D$ implies $D' \leq D$: each atom $p(\vec{d})$ that is compliant for D' is also compliant for D, as $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D'$ with $e = c_k$ trivially implies that $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D$ with $e = c_k$. For (ii), we likewise conclude for $D' = D \cup D''$ where each $\delta'' \in D''$ is subsumed by some $\delta \in D$, that $D' \leq D$ as $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D'$ with $e = c_k$ implies that $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D$ with $e = c_k$. As D is faithful, (i) and (ii) follow thus from Proposition 4.4. □

Consequently, we can tighten a faithful set D by sequentially dropping constants c from io-dependencies $\delta = \langle i, j : J, k : e\rangle$ in D, i.e., check whether $D \cup \delta'$ for $\delta' = \langle i, j : J \setminus \{c\}, k : e\rangle$ is faithful and if so, replace D with $(D \setminus \{\delta\}) \cup \{\delta'\}$.

We can simplify D by exploiting the following equivalences; let $\delta^*(i, j, k{:}e) = \langle i, j : \mathcal{C}, k : e\rangle$ for any possible i, j, and $k : e$.

Proposition 4.5. *For $D \subseteq dep(\&g[\vec{p}])$ and $\langle i, j : J, k : e\rangle \in dep(\&g[\vec{p}])$, we have (i) $D \equiv D \cup \{\delta^*(i, j, k{:}e)\} \equiv D \setminus \{\delta^*(i, j, k{:}e)\}$, and (ii) for any $\delta = \langle i, j : J, k : e\rangle$, $\delta' = \langle i, j : J', k : e\rangle \in D$ that $D \equiv D \cup \{\langle i, j : J \cap J', k : e\rangle\}$.*

Proof. Let $D \subseteq dep(\&g[\vec{p}])$ for a ge-predicate $\&g[\vec{p}]$, and let $\langle i, j : J, k : e\rangle \in dep(\&g[\vec{p}])$.

(i) First, we have that $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$ if and only if $p_i(\vec{d}) \in comp(D \cup \{\delta^*(i, j, k{:}e)\}, \&g[\vec{p}](\vec{c}))$ because $d_j \notin J'$ for some $\langle i, j : J', k : e\rangle \in D$ implies that

$d_j \notin J''$ for some $\langle i, j : J'', k : e\rangle \in D \cup \{\delta^*(i, j, k{:}c)\}$, and it holds trivially that $d_j \in \mathcal{C}$ for every $d_j \in J'$ and every $\langle i, j : J', k : e\rangle \in D \cup \{\delta^*(i, j, k{:}e)\}$.

We can also show that $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$ if and only if $p_i(\vec{d}) \in comp(D \setminus \{\delta^*(i, j, k{:}e)\}, \&g[\vec{p}](\vec{c}))$. First, $d_j \notin J'$ for some $\langle i, j : J', k : e\rangle \in D$ implies that $d_j \notin J'$ for some $\langle i, j : J', k : e\rangle \in D \setminus \{\delta^*(i, j, k{:}c)\}$ because $d_j \in \mathcal{C}$. Moreover, if $d_j \in J'$ for every $\langle i, j : J', k : e\rangle \in D$, then $d_j \in J''$ for every $\langle i, j : J'', k : e\rangle \in D \setminus \{\delta^*(i, j, k{:}c)\}$ since $D \setminus \{\delta^*(i, j, k{:}c)\} \subseteq D$.

(ii) Next, let $\delta = \langle i, j{:}J, k{:}e\rangle$, $\delta' = \langle i, j{:}J', k{:}e\rangle \in D$ be arbitrary io-dependencies in D. We prove that $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$ if and only if $p_i(\vec{d}) \in comp(D \cup \{\langle i, j{:}J \cap J', k{:}e\rangle\}, \&g[\vec{p}](\vec{c}))$. Let $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$ be a compliant atom. Then, $d_j \in J \cap J'$ because $d_j \in J$ and $d_j \in J'$, which shows that $p_i(\vec{d}) \in comp(D \cup \{\langle i, j{:}J \cap J', k{:}e\rangle\}, \&g[\vec{p}](\vec{c}))$. Finally, let $p_i(\vec{d})$ be a ground ordinary atom s.t. $p_i(\vec{d}) \notin comp(D, \&g[\vec{p}](\vec{c}))$. But then $p_i(\vec{d}) \notin comp(D \cup \{\langle i, j{:}J \cap J', k{:}e\rangle\}, \&g[\vec{p}](\vec{c}))$ as $d_j \notin J''$ for some $\langle i, j : J'', k : e\rangle \in D$.

□

That is, $\delta^*(i, j, k{:}e)$ is like a tautology, and we can replace all dependencies for i, j and $k : e$ in D by one which contains the intersection of all their J-sets. We thus can *normalize* D into $nf(D)$ such that for each i, j, and $k : e$ exactly one io-dependency occurs, and then start tightening. We then obtain:

Proposition 4.6. *Given a faithful $D \subseteq dep(\&g[\vec{p}])$, exhaustive tightening of $nf(D)$ results in a tight faithful D'.*

Proof. First, D' is faithful because faithfulness is checked each time before an io-dependency $\delta = \langle i, j : J, k : e\rangle$ is replaced by $\delta' = \langle i, j : J \setminus \{c\}, k : e\rangle$ during tightening. We need to show that D' is also tight.

Towards a contradiction, suppose that D' is exhaustively tightened but that it is not tight, i.e. that there is a set $D'' \subseteq dep(\&g[\vec{p}])$ s.t. $D'' \leq D'$ and $D' \not\leq D''$. This implies that there is some $p_i \in \vec{p}$ s.t. $p_i(\vec{d}) \in comp(D', \&g[\vec{p}](\vec{c}))$ and $p_i(\vec{d}) \notin comp(D'', \&g[\vec{p}](\vec{c}))$ for some output $\vec{c}$ of $\&g[\vec{p}]$. Thus, for some $1 \leq j \leq ar(p)$ we have that $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D'$ with $e = c_k$ but $d_j \notin J$ for some $\langle i, j : J, k : e\rangle \in D''$ with $e = c_k$, according to the definition of compliant atoms. But then, $D' \cup \{\delta'\}$ for $\delta' = \langle i, j : J \setminus \{d_j\}, k : e\rangle$ with $e = c_k$, where $\langle i, j : J, k : e\rangle \in D$ with $e = c_k$, is also faithful due to Proposition 4.4 because it holds that $D'' \leq D' \cup \{\delta'\}$, and D'' is faithful. This means that D' is not exhaustively tightended, which contradicts our assumption. □

The set $D = \emptyset$ is trivially faithful, and $nf(\emptyset)$ consists of all $\delta^*(i, j, k{:}c)$; thus even without user input, a tight faithful set D' for $\&g[\vec{p}]$ is constructible. Moreover, semantically faithful sets of compliant atoms have the intersection property.

Proposition 4.7. *If $D_1, D_2 \subseteq dep(\&g[\vec{p}])$ are faithful, then $D_1 \cup D_2$ is faithful, and for every $\vec{c}$, $comp(D_1 \cup D_2, \&g[\vec{p}](\vec{c})) = comp(D_1, \&g[\vec{p}](\vec{c})) \cap comp(D_2, \&g[\vec{p}](\vec{c}))$.*

Proof. Let $\vec{c}$ be an arbitrary output tuple of $\&g[\vec{p}]$, let $D_1, D_2 \subseteq dep(\&g[\vec{p}])$, and let $C_D = comp(D, \&g[\vec{p}](\vec{c}))$ for any $D \subseteq dep(\&g[\vec{p}])$.

We begin by showing that $C_{D_1 \cup D_2} = C_{D_1} \cap C_{D_2}$. For a ground atom $p(\vec{d})$, with $\vec{d} = d_1, ..., d_l$, it holds that $p(\vec{d}) \in C_{D_1 \cup D_2}$ iff $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D_1 \cup D_2$ with $e = c_k$. The previous holds iff $d_j \in J$ for all $\langle i, j : J, k : e\rangle \in D_1$ and $d_j \in J'$ for all $\langle i, j : J', k : e\rangle \in D_2$ with $e = c_k$, which holds iff $p(\vec{d}) \in C_{D_1}$ and $p(\vec{d}) \in C_{D_2}$. This proves that $p(\vec{d}) \in C_{D_1} \cap C_{D_2}$ iff $p(\vec{d}) \in C_{D_1 \cup D_2}$.

Now, we prove that if D_1 and D_2 are faithful, then $D_1 \cup D_2$ is faithful. Recall that according to Definition 4, a set of io-dependencies $D \subseteq dep(\&g[\vec{p}])$ is faithful iff:

> (*) for any assignments $\mathbf{A}, \mathbf{A}'$ and ground external atom $\&g[\vec{p}](\vec{c})$, either $\mathbf{A}(p_i(\vec{d})) \neq \mathbf{A}'(p_i(\vec{d}))$ for some $p_i(\vec{d}) \in C_D$ or $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.

Suppose that statement (*) holds w.r.t. C_{D_1} and C_{D_2}; we show that it also holds w.r.t. $C_{D_1 \cup D_2} = C_{D_1} \cap C_{d_2}$.

Towards a contradiction, suppose that $C_{D_1 \cup D_2}$ does not satisfy (*). Hence there exist assignments $\mathbf{A}, \mathbf{A}'$ such that $\mathbf{A}(p_i(\vec{d})) = \mathbf{A}'(p_i(\vec{d}))$ for all $p_i(\vec{d}) \in C_{D_1 \cup D_2}$ and $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.

Let $\mathbf{A}_1$ be the assignment such that

$$\mathbf{A}_1(p_i(\vec{d})) = \begin{cases} \mathbf{A}(p_i(\vec{d})) & \text{if } p_i(\vec{d}) \in C_{D_1}, \\ \mathbf{A}'(p_i(\vec{d})) & \text{if } p_i(\vec{d}) \in C_{D_2} \setminus C_{D_1}, \\ \mathbf{T} & \text{otherwise.} \end{cases}$$

Then, as $\mathbf{A}_1$ and $\mathbf{A}$ coincide on C_{D_1} and the latter satisfies (*), it follows $f_{\&g}(\mathbf{A}_1, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$.

Let $\mathbf{A}'_1$ be similarly the assignment such that

$$\mathbf{A}'_1(p_i(\vec{d})) = \begin{cases} \mathbf{A}'(p_i(\vec{d})) & \text{if } p_i(\vec{d}) \in C_{D_2}, \\ \mathbf{A}(p_i(\vec{d})) & \text{if } p_i(\vec{d}) \in C_{D_1} \setminus C_{D_2}, \\ \mathbf{T} & \text{otherwise.} \end{cases}$$

Then, as $\mathbf{A}'_1$ and $\mathbf{A}'$ coincide on C_{D_2} and the latter satisfies (*), it follows $f_{\&g}(\mathbf{A}'_1, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$. As $\mathbf{A}$ and $\mathbf{A}'$ coincide on $C_{D_1 \cup D_2} = C_{D_1} \cap C_{D_2}$, by construction $\mathbf{A}_1 = \mathbf{A}'_1$, and it follows $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}_1, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}'_1, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$; however, this contradicts the assumption that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$. □

Consequently, every ge-predicate has a semantically unique tight set of faithful io-dependencies. However, syntactically different tight faithful sets may exist.

Example 4.8. Consider a ge-predicate $\&g[p]$ which is true for output (a, b) w.r.t. an assignment $\mathbf{A}$ iff $\mathbf{T}p(c) \in \mathbf{A}$, and false for all other output tuples. Then $\{\langle 1, 1 : \{c\}, 1 : a\rangle\}$ and $\{\langle 1, 1 : \{c\}, 2 : b\rangle\}$ are faithful, and both are tight. △

To check faithfulness of a set $D \subseteq dep(\&g[\vec{p}])$, formally the oracle function $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$ must be evaluated for all evaluations of predicates $p \in \vec{p}$ and output tuples $\vec{c}$, which naively is often not feasible in practice.

Example 4.9. Reconsider $\&suc[node](X)$ from Example 4.5. To check faithfulness of the specified io-dependencies w.r.t. output a, the oracle function needs to be evaluated under all possible assignments to atoms with predicate *node*. △

In the worst case, this cannot be avoided by the following result, where we assume that $\&g[\vec{p}](\vec{c})$ is decidable in polynomial time.

Proposition 4.8. *Checking faithfulness of a given set $D \subseteq dep(\&g[\vec{p}])$ is co-NEXP-complete in general, and co-NP-complete for fixed predicate arities.*

Proof. See Appendix A.1, page 193. □

When certain properties of external sources are known, less external calls are needed for faithfulness checking, e.g. for monotonic functions. An input $p_i \in \vec{p}$ of a ge-predicate $\&g[\vec{p}]$ is *monotonic*, if for any assignment $\mathbf{A}$ and output $\vec{c}$, $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$ for every $\mathbf{A}' \geq \mathbf{A}$ s.t. $\mathbf{A}(p_j(\vec{d})) = \mathbf{A}'(p_j(\vec{d}))$ for all predicates $p_j \in \vec{p}$ with $p_j \neq p_i$ (cf. (Eiter et al., 2018)). Based on monotonicity, the number of assignments to consider in a faithfulness check can be decreased.

Proposition 4.9. *If $p_i \in \vec{p}$ for $\&g[\vec{p}]$ is monotonic, a set $D \subseteq dep(\&g[\vec{p}])$ is* faithful for $\&g[\vec{p}]$ *iff for any assignments $\mathbf{A}$, $\mathbf{A}'$ s.t. $\mathbf{T}p_i(\vec{d}) \in \mathbf{A}$ and $\mathbf{F}p_i(\vec{d}) \in \mathbf{A}'$ for every $p_i(\vec{d}) \notin comp(D, \&g[\vec{p}](\vec{c}))$ and $\mathbf{A}(p_i(\vec{d})) = \mathbf{A}'(p_i(\vec{d}))$ for every $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$, it holds that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.*

Proof. Let p_i be a monotonic input parameter of a ge-predicate $\&g[\vec{p}]$, and $D \subseteq dep(\&g[\vec{p}])$. We need to show that D is faithful for $\&g[\vec{p}]$ iff for any two assignments $\mathbf{A}$, $\mathbf{A}'$ s.t. $\mathbf{T}p_i(\vec{d}) \in \mathbf{A}$ and $\mathbf{F}p_i(\vec{d}) \in \mathbf{A}'$ for every atom $p_i(\vec{d}) \notin comp(D, \vec{c})$, and $\mathbf{A}(p_i(\vec{d})) = \mathbf{A}'(p_i(\vec{d}))$ for every atom $p_i(\vec{d}) \in comp(D, \vec{c})$, it holds that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.

($\Rightarrow$) The only-if direction follows directly from Definition 4.4.

($\Leftarrow$) To prove the if-direction, (*) assume that for any two assignments $\mathbf{A}$, $\mathbf{A}'$ s.t. $\mathbf{T}p_i(\vec{d}) \in \mathbf{A}$ and $\mathbf{F}p_i(\vec{d}) \in \mathbf{A}'$ for every atom $p_i(\vec{d}) \notin comp(D, \vec{c})$, and $\mathbf{A}(p_i(\vec{d})) = \mathbf{A}'(p_i(\vec{d}))$ for every atom $p_i(\vec{d}) \in comp(D, \vec{c})$, it holds that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$. We need to show that D is faithful for $\&g[\vec{p}]$ according to Definition 4.4, i.e. that for any two assignments $\mathbf{A}_*$, $\mathbf{A}'_*$ and any possible output tuple $\vec{c}$ for $\&g[\vec{p}]$ it holds that if $\mathbf{A}_*(p_i(\vec{d})) = \mathbf{A}'_*(p_i(\vec{d}))$ for every atom $p_i(\vec{d}) \in comp(D, \vec{c})$, then $f_{\&g}(\mathbf{A}_*, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}'_*, \vec{p}, \vec{c})$.

First consider the case that $f_{\&g}(\mathbf{A}_*, \vec{p}, \vec{c}) = \mathbf{T}$. Let $\mathbf{A}$ be an assignment s.t. $\mathbf{A}_*(p_j(\vec{d})) = \mathbf{A}(p_j(\vec{d}))$ for all atoms $p_j(\vec{d})$ with $p_j \in \vec{p}$ where $p_j \neq p_i$ and all atoms $p_j(\vec{d}) \in comp(D, \vec{c})$ where $p_j = p_i$, and $\mathbf{T}p_i(\vec{d}) \in \mathbf{A}$ for every atom $p_i(\vec{d}) \notin comp(D, \vec{c})$. Furthermore, let $\mathbf{A}'$ be an assignment s.t. $\mathbf{A}'_*(p_j(\vec{d})) = \mathbf{A}'(p_j(\vec{d}))$ for all atoms $p_j(\vec{d})$ with $p_j \in \vec{p}$ where $p_j \neq p_i$ and all atoms $p_j(\vec{d}) \in comp(D, \vec{c})$ where $p_j = p_i$, and $\mathbf{F}p_i(\vec{d}) \in \mathbf{A}'$ for every atom $p_i(\vec{d}) \notin comp(D, \vec{c})$. Since $f_{\&g}(\mathbf{A}_*, \vec{p}, \vec{c}) = \mathbf{T}$, we obtain $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ according to the definition of monotonic input parameter. Moreover, it follows from our assumption (*) that $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$. Finally, $f_{\&g}(\mathbf{A}'_*, \vec{p}, \vec{c}) = \mathbf{T}$ follows again from the definition of monotonic input parameter because we have that $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$, which proves that $f_{\&g}(\mathbf{A}_*, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}'_*, \vec{p}, \vec{c})$.

The case for $f_{\&g}(\mathbf{A}_*, \vec{p}, \vec{c}) = \mathbf{F}$ works analogously. □

Example 4.10 (cont'd). As *node* is a monotonic input parameter of $\&suc[node]$, for checking faithfulness w.r.t. a it suffices to evaluate $f_{\&suc}(\mathbf{A}, node, a)$ under two assignments $\mathbf{A}_t$ and $\mathbf{A}_f$, s.t. $\mathbf{A}_t \subseteq \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{T}node(c), \mathbf{T}node(d)\}$ and $\mathbf{A}_f \subseteq \{\mathbf{F}node(a), \mathbf{F}node(b), \mathbf{F}node(c), \mathbf{F}node(d)\}$. △

Under additional conditions, we obtain tractability:

Corollary 4.2. *If all $p_i \in \vec{p}$ for $\&g[\vec{p}]$ are monotonic and $|comp(D, \&g[\vec{p}](\vec{c}))|$ is bounded, then checking faithfulness is polynomial for fixed predicate arities.*

Proof. See Appendix A.1, page 195. □

The same holds for computing a tight faithful set D for $\&g[\vec{p}]$. In practice, this applies to Example 4.5, if the external graph has bounded degree.

We can under the assertions of the previous corollary compute some tight faithful set for an output $\vec{c}$, by cycling through all sets S of input atoms of size $i = 0, 1, 2$ etc. up to the limit ℓ; for each S, we can test whether S satisfies the faithfulness condition in polynomial time (as argued above), and whether S is represented by some set D of io-dependencies. To this end, we collect in $D = D(S, \&g[\vec{p}](\vec{c}))$ all dependencies $\langle i, j{:}J_j, k{:}c_k\rangle$ where J_j is the set of all constants d_j that appear in atoms $p_i(d_1, \ldots, d_l)$ in S for input p_i as j-th argument, and c_k is the k-th argument of $\vec{c}$, for all k; we then check whether $S = comp(D, \&g[\vec{p}](\vec{c}))$ holds, by inspecting all possible ground input atoms. Overall, this can be done in polynomial time.

Relativized io-dependencies

So far, the context of a given HEX-program has not been exploited for specifying respective io-dependencies. However, without considering how dependencies in an external source may be affected by input parameters, all io-dependencies that may hold under any possible extension of input predicates must be respected. This is illustrated by the following example.

Example 4.11. Consider $\&suc[edge, node](X)$, where edges from *edge* are inserted into $\mathcal{G}$ before successor nodes are output. If it is unknown which edges can be added, io-dependencies must account for the complete graph (all edges), which is a maximal overapproximation. Now, consider the following HEX-program.

$$edge(b, c) \vee n\_edge(b, c). \quad node(a). \quad node(b) \leftarrow \&suc[edge, node](b).$$

As $edge(b, c)$ is the only atom with predicate *edge* that can potentially be true in the input of $\&suc[edge, node](b)$ in any answer set, it suffices to specify io-dependencies w.r.t. the graph $\mathcal{G}' = (V, E \cup \{b \to c\})$ to ensure e-minimality. △

To account for the inputs to external sources that are possible in answer sets, we define faithfulness w.r.t. a HEX-program Π. Let $env(\Pi)$ denote the set of all atoms for Π that are true in some compatible set of Π.

Definition 4.8 (Relativized Faithfulness)**.** *A set $D \subseteq dep(\&g[\vec{p}])$ is* faithful w.r.t. a HEX-program Π*, if for any assignments $\mathbf{A}$, $\mathbf{A}'$ s.t. $\{a \mid \mathbf{T}a \in \mathbf{A} \cup \mathbf{A}'\} \subseteq env(\Pi)$, and*

for any output tuple $\vec{c}$ *for* $\&g[\vec{p}]$*, either* $\mathbf{A}(p_i(\vec{d})) \neq \mathbf{A}'(p_i(\vec{d}))$ *for some atom* $p_i(\vec{d}) \in comp(D, \&g[\vec{p}](\vec{c}))$ *or* $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.

We show that skipping e-minimality checks based on the relativized definition of faithful io-dependencies is still safe.

Proposition 4.10. *Theorem 4.2 still holds if the specified io-dependencies are faithful w.r.t. to the* HEX*-program* Π *at hand according to Definition 4.8.*

Proof. The proof of Theorem 4.2 can be directly adopted because, in order to ensure that the projection $\mathbf{A}$ of a compatible set $\hat{\mathbf{A}}$ for a HEX-program Π is a $\leq$-minimal model of $f\Pi^{\mathbf{A}}$, one only needs to consider assignments $\mathbf{A}'$ s.t. $\mathbf{A}' \leq \mathbf{A}$; and we have that $\mathbf{A}' \leq \{\mathbf{T}a \mid a \in env(\Pi)\}$ for every projection $\mathbf{A}'$ of a compatible set $\hat{\mathbf{A}}'$ for Π. □

The properties of above can be adjusted to this setting.

4.3 Empirical Evaluation

In this section, we present an empirical evaluation regarding the usage of partial evaluation during the unfounded set search, discussed in Section 4.1, and pruning e-minimality checks based on semantic dependency information as discussed in Section 4.2.

4.3.1 Experimental Setup

To experimentally test our new techniques for tightly integrating the e-minimality check and external evaluation, we implemented them in the HEX-solver DLVHEX 2.5.0, which uses GRINGO 4.4.0 and CLASP 3.1.1 as backends (Gebser, Kaufmann, et al., 2011), and tested it on randomly generated instances. Io-depencendies for external atoms are specified by plugin-methods that compute whether a dependency between given input and output values exists.

Next, we first describe the evaluation platform and the different configurations we are comparing. We then describe the benchmarks used for the evaluation. All instances and log files of the experiments in Section 4.3.3 can be found at `http://www.kr.tuwien.ac.at/research/projects/inthex/partialeval`, and the data of the experiments in Section 4.3.4 is available at `www.kr.tuwien.ac.at/research/projects/inthex/dep-pruning`.

Evaluation Platform

We used a Linux machine with two 12-core AMD Opteron 6238 SE CPUs and 512 GB RAM; the timeout was 300 seconds and the memout 8 GB per instance. The average running times of 50 instances for the experiments in Section 4.3.3, respectively 10 instances for the experiments presented in Section 4.3.4, per problem size is reported (in seconds) for computing all answer sets; timeouts are shown in parentheses in the result tables.

Benchmark Configurations

First, we tested two heuristics for interleaving external evaluations with the search for unfounded sets (cf. Algorithm 4.1, Part (d)). For comparison, we also used the configurations **never**, **always** and **qxp-c** introduced in Section 3.5.3. Accordingly, our configurations for testing partial evaluation during the unfounded set search are the following:

- **never**: external atoms are only evaluated w.r.t. *candidate models* (representing the standard configuration of DLVHEX without the new techniques);
- **always**: external atoms are evaluated w.r.t. *partial assignments* after every solver guess during the model search;
- **ufs-a**: external atoms are evaluated w.r.t. *partial assignments* after every solver guess during unfounded set search; and
- **ufs-p**: external atoms are evaluated w.r.t. *partial assignments* only after each 10^{th} solver guess during unfounded set search.

Furthermore, we investigated the effect of enabling external evaluations based on partial assignments both during the main search and the unfounded set search, i.e. combining configurations **always** and **ufs-a**.

We then investigated the effect of applying dependency graph pruning based on specified io-dependencies by adding the configuration **io-dep**. To gain insights into how dependency graph pruning and partial evaluation interact, we considered the frequency of external calls also in our experiments w.r.t. e-minimality check skipping; this is of interest as early external evaluation can speed up model search as well as the e-minimality check and can thus potentially influence the impact of our new technique. More specifically, we compared three different configurations, **never**, **always** and **ufs-a**, each with and without adding dependency pruning (**io-dep**). In the result tables, we show combinations of configurations where interactions are expected.

Benchmark Problems

In our benchmark programs used in Chapter 3, the one for the taxi assignment problem contains cyclic dependencies through external atoms that can be detected syntactically, while this is not the case for the pseudo-boolean and conflicting strategic companies programs. The absence of such cyclic dependencies means that compatible sets of a HEX-program already correspond to its answer sets and the e-minimality check can be skipped even when no io-dependencies are specified. Consequently, partial evaluation during the search for unfounded sets did not have an impact on the performance results for the pseudo-boolean and conflicting strategic companies benchmarks from Section 3.5.3.

For investigating the effect of partial evaluation during the unfounded set search, we thus considered the following two benchmark problems:

- The *Taxi Assignment with Ontology Access* benchmark from Section 3.5.3.
- Two variants of the *Strategic Companies* problem (Cadoli et al., 1997), which is popular with ASP competitions and was already employed in Section 3.5.3, extended with externally computed control relations among companies based on shares (*Strategic Companies (with Nonmonotonic) External Controls Relation*).

In addition, we utilized two further problems for our experiments on minimality check skipping in Section 4.3.4, where the absence of cyclic dependencies involving external atoms can be exploited when io-dependencies are specified:

- A *User Access Selection* problem concerning the assignment of access rights w.r.t. nodes of a computer network, where constraints are imposed based on reachability of nodes within the network.
- *Sequential Allocation of Indivisible Goods* as considered by Kalinowski et al. (2013), where agent preferences over a set of goods are interfaced via an external atom.

The learning function Λ_u was used for the strategic companies benchmark with nonmonotonic external control relation and the sequential allocation benchmark. Due to monotonicity of external sources, the learning function Λ_{mu} can be utilized for the other two benchmarks (cf. Section 3.3.2).

4.3.2 Hypotheses

Our hypotheses concerning the use of partial evaluation in the unfounded set search were the following:

(H4.1) The heuristics **ufs-a** and **ufs-p** decrease the runtime over **never** if useful information is obtainable by early evaluation during the unfounded set search with little runtime overhead, and increase it otherwise, whereby the effect is stronger for **ufs-a**.

(H4.2) The heuristics **ufs-p** performs better than **ufs-a** if external calls need more time or less information can be gained from them, mitigating the tradeoff between information gain and running time invested in additional calls.

(H4.3) If the heuristics **ufs-a** or **ufs-p** are combined with the heuristics **always**, there is a further speedup in case many io-nogoods learned during the unfounded set search are not already learned during the main search. Thus, the combination is expected to be more effective when for learning, the function Λ_u is used instead of the function Λ_{mu}.

Regarding our technique for dependency graph pruning based on semantic dependencies, we made the following predictions:

#	All Answer Sets											
	never		**always**		**ufs-p**		**ufs-a**		**always + ufs-a**		**qxp-c**	
4	0.19	(0)	0.22	(0)	0.20	(0)	0.48	(0)	0.53	(0)	**0.17**	(0)
6	0.35	(0)	0.35	(0)	0.38	(0)	1.64	(0)	1.78	(0)	**0.22**	(0)
8	1.27	(0)	0.70	(0)	1.02	(0)	8.70	(0)	8.31	(0)	**0.33**	(0)
10	7.86	(0)	1.41	(0)	3.67	(0)	33.22	(0)	30.28	(0)	**0.65**	(0)
12	228.11	(17)	4.16	(0)	37.50	(1)	139.01	(10)	93.41	(3)	**1.75**	(0)
14	300.00	(50)	12.25	(0)	186.79	(20)	281.62	(42)	190.64	(21)	**6.45**	(0)
16	300.00	(50)	24.42	(1)	292.14	(47)	300.00	(50)	273.96	(40)	**13.27**	(1)
18	300.00	(50)	81.46	(4)	300.00	(50)	300.00	(50)	295.59	(49)	**58.43**	(3)
20	300.00	(50)	110.71	(8)	300.00	(50)	300.00	(50)	300.00	(50)	**67.77**	(6)
22	300.00	(50)	203.06	(24)	300.00	(50)	300.00	(50)	288.02	(48)	**178.43**	(18)
24	300.00	(50)	243.39	(32)	300.00	(50)	300.00	(50)	294.02	(49)	**209.65**	(26)
26	300.00	(50)	284.93	(45)	300.00	(50)	300.00	(50)	294.01	(49)	**265.89**	(40)
28	300.00	(50)	253.71	(42)	300.00	(50)	300.00	(50)	258.15	(43)	**247.76**	(40)
30	300.00	(50)	294.02	(49)	300.00	(50)	300.00	(50)	294.02	(49)	**294.01**	(49)

Table 4.1: Results for taxi assignment with ontology access w.r.t. partial evaluation during unfounded set search (all answer sets)

(H4.4) Configuration **io-dep** decreases the running time if e-cycles can be removed from the dependency graph.

(H4.5) The speedup is larger when **io-dep** is combined with **always** and smaller when combined with **ufs-a**, whenever partial evaluation is beneficial.

(H4.6) If pruning does not skip e-minimality checks, no significant overhead in terms of running time with **io-dep** is incurred.

4.3.3 Experiments on Partial Evaluation for Minimality Checking

In this section, we begin our empirical investigation by testing the impact of partial evaluations during the unfounded set search, using different heuristics.

Taxi Assignment

For testing the effect of partial evaluation during the unfounded set search w.r.t. the taxi assignment benchmark, we used the same set of problem instances as before (cf. Table 3.5). The results are shown in Tables 4.1 and 4.2, where we also report the running times of the conditions **always** and **qxp-c** for comparison with partial evaluation during the main search.

For this benchmark, the compatible sets are identical to the answer sets, such that no unfounded sets are detected during unfounded set search. However, due to cyclic dependencies through external atoms, minimality w.r.t. the FLP-reduct still needs to be verified for each instance by means of the unfounded set check. We observe that configurations **ufs-p** and **ufs-a** slightly improve the efficiency over **never**, where **ufs-p** yields better results since calls to the external oracle are costly in this benchmark. However, performing early evaluations during the main search in condition **always** is

#	First Answer Set					
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**
4	0.16 (0)	0.17 (0)	0.16 (0)	0.20 (0)	0.21 (0)	**0.15** (0)
6	0.20 (0)	0.19 (0)	0.20 (0)	0.28 (0)	0.28 (0)	**0.18** (0)
8	0.39 (0)	**0.22** (0)	0.37 (0)	0.55 (0)	0.38 (0)	**0.22** (0)
10	1.27 (0)	**0.22** (0)	1.16 (0)	1.65 (0)	0.46 (0)	0.32 (0)
12	22.60 (0)	**0.27** (0)	21.88 (0)	25.55 (1)	0.59 (0)	0.43 (0)
14	137.84 (15)	**0.32** (0)	134.38 (14)	139.96 (15)	0.79 (0)	1.08 (0)
16	273.70 (40)	**0.37** (0)	267.38 (39)	270.47 (40)	1.04 (0)	7.50 (1)
18	300.00 (50)	**0.43** (0)	300.00 (50)	300.00 (50)	1.32 (0)	26.53 (3)
20	300.00 (50)	**0.47** (0)	300.00 (50)	300.00 (50)	1.61 (0)	9.62 (1)
22	300.00 (50)	**0.69** (0)	300.00 (50)	300.00 (50)	2.92 (0)	53.99 (3)
24	300.00 (50)	**0.78** (0)	300.00 (50)	300.00 (50)	3.78 (0)	88.03 (9)
26	300.00 (50)	**0.89** (0)	300.00 (50)	300.00 (50)	4.56 (0)	84.77 (10)
28	300.00 (50)	**1.04** (0)	300.00 (50)	300.00 (50)	5.05 (0)	95.74 (14)
30	300.00 (50)	**1.29** (0)	300.00 (50)	300.00 (50)	8.12 (0)	103.03 (11)

Table 4.2: Results for taxi assignment with ontology access w.r.t. partial evaluation during unfounded set search (first answer set)

$$s(C1) \vee s(C1) \vee s(C3) \vee s(C4) \leftarrow \mathit{producedBy}(\_, C1, C2, C3, C4).$$
$$s(C) \leftarrow \&\mathit{majority}[s](C), \mathit{company}(C).$$

Figure 4.2: Strategic Companies with External Controls Relation Rules

much faster resulting in less timeouts; and combining early evaluation in the main and the unfounded set search does not result in an additional speedup. This is expected: as reasoning in a DL ontology is monotonic and for DL-Lite ontologies the io-nogoods are small, the information that is obtained by early evaluation in the respective parts is largely overlapping.

Strategic Companies with External Controls Relation

We considered a variant of the strategic companies problem from Section 3.5.3, where the controls relation is derived by means of an external atom of the form $\&\mathit{majority}[\mathit{strategic}](c)$, based on the company shares that other companies own (cf. Figure 4.2). A company is then controlled by a suite of other companies if their combined shares exceed 50 %. No conflict relations are added in this benchmark as they only remove compatible sets and do not have a direct influence on the minimality check, while the aim of this experiment is to investigate the effect of partial evaluation on the unfounded set search.

Let $\mathit{shares}(c_1, c_2)$ denote the fraction of shares of company c_2 that company c_1 owns. Given a partial assignment $\mathbf{A}$, a company c, and a predicate $\mathit{strategic}$ representing a set of companies, the corresponding three-valued oracle function $f_{\&\mathit{majority}}(\mathbf{A}, \mathit{strategic}, c)$ is defined as follows:

$$f_{\&\mathit{majority}}(\mathbf{A}, \mathit{strategic}, c) = \begin{cases} \mathbf{T} & \text{if } \sum_{\mathbf{T}\mathit{strategic}(c_i) \in \mathbf{A}} \mathit{shares}(c_i, c) > 50\,\%; \\ \mathbf{F} & \text{if } \sum_{\mathbf{T}\mathit{strategic}(c_i), \mathbf{U}\mathit{strategic}(c_i) \in \mathbf{A}} \mathit{shares}(c_i, c) \leq 50\,\%; \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

#	All Answer Sets												solutions / compatible sets
	never		**always**		**ufs-p**		**ufs-a**		**always + ufs-a**		**qxp-c**		
2	**0.13**	(0)	0.14	(0)	**0.13**	(0)	0.14	(0)	0.14	(0)	0.14	(0)	1.18 / 1.68
4	**0.15**	(0)	0.17	(0)	0.16	(0)	0.19	(0)	0.21	(0)	0.17	(0)	1.70 / 2.96
6	**0.19**	(0)	0.25	(0)	0.21	(0)	0.31	(0)	0.36	(0)	0.26	(0)	2.56 / 4.46
8	**0.29**	(0)	0.41	(0)	0.33	(0)	0.58	(0)	0.71	(0)	0.50	(0)	4.20 / 6.42
10	**0.63**	(0)	0.81	(0)	0.76	(0)	1.27	(0)	1.49	(0)	0.95	(0)	7.44 / 10.10
12	1.80	(0)	**1.56**	(0)	1.91	(0)	2.86	(0)	2.95	(0)	1.89	(0)	9.30 / 13.62
14	4.97	(0)	**3.14**	(0)	5.09	(0)	6.96	(0)	6.31	(0)	3.49	(0)	18.08 / 24.46
16	15.88	(0)	**5.98**	(0)	15.03	(0)	16.97	(0)	12.81	(0)	6.65	(0)	26.96 / 35.04
18	59.10	(0)	**13.52**	(0)	50.48	(0)	47.16	(0)	26.65	(0)	13.86	(0)	36.52 / 47.56
20	169.77	(1)	**30.46**	(0)	132.65	(0)	124.17	(0)	55.99	(0)	31.44	(0)	64.30 / 79.58
22	297.81	(46)	**59.32**	(0)	285.57	(40)	275.11	(32)	108.75	(0)	61.88	(0)	97.02 / 116.22
24	300.00	(50)	128.94	(0)	300.00	(50)	300.00	(50)	210.98	(11)	**127.59**	(4)	≥154.66 / 195.84
26	300.00	(50)	255.72	(20)	300.00	(50)	300.00	(50)	269.39	(36)	**235.22**	(17)	≥242.22 / 366.26
28	300.00	(50)	292.88	(43)	300.00	(50)	300.00	(50)	295.10	(46)	**279.55**	(41)	≥121.64 / 402.66
30	300.00	(50)	300.00	(50)	300.00	(50)	300.00	(50)	300.00	(50)	**298.91**	(49)	≥102.06 / 548.94

Table 4.3: Results for strategic companies with external controls relation (all answer sets)

#	First Answer Set												
	never		**always**		**ufs-p**		**ufs-a**		**always + ufs-a**		**qxp-c**		
2	**0.13**	(0)	0.14	(0)	**0.13**	(0)	0.14	(0)	0.14	(0)	0.14	(0)	
4	**0.14**	(0)	0.16	(0)	0.15	(0)	0.16	(0)	0.17	(0)	0.15	(0)	
6	**0.16**	(0)	0.18	(0)	**0.16**	(0)	0.20	(0)	0.21	(0)	0.21	(0)	
8	**0.19**	(0)	0.22	(0)	0.20	(0)	0.26	(0)	0.29	(0)	0.32	(0)	
10	**0.25**	(0)	0.29	(0)	0.29	(0)	0.39	(0)	0.43	(0)	0.55	(0)	
12	0.51	(0)	**0.46**	(0)	0.55	(0)	0.71	(0)	0.67	(0)	1.01	(0)	
14	0.72	(0)	**0.55**	(0)	0.77	(0)	0.98	(0)	0.79	(0)	1.45	(0)	
16	2.03	(0)	**0.85**	(0)	2.46	(0)	2.13	(0)	1.25	(0)	2.67	(0)	
18	5.01	(0)	**2.00**	(0)	7.49	(0)	5.06	(0)	2.05	(0)	5.87	(0)	
20	14.15	(0)	**2.31**	(0)	15.46	(0)	12.55	(0)	2.41	(0)	11.64	(0)	
22	54.07	(2)	5.10	(0)	54.66	(3)	49.00	(3)	**4.19**	(0)	21.82	(0)	
24	73.45	(3)	10.84	(0)	99.72	(4)	64.73	(2)	**4.92**	(0)	48.33	(1)	
26	144.56	(12)	23.70	(1)	155.93	(15)	124.87	(10)	**7.13**	(0)	73.96	(7)	
28	189.26	(21)	34.27	(2)	190.98	(25)	169.98	(22)	**9.29**	(0)	153.97	(20)	
30	222.31	(31)	86.92	(7)	230.87	(35)	190.72	(27)	**11.35**	(0)	172.17	(22)	

Table 4.4: Results for strategic companies with external controls relation (first answer set)

As the oracle function behaves monotonically, we can employ the learning function Λ_{mu}.

For testing, we randomly generated instances with $N \in [2, 30]$ companies, randomly distributed 50 % to 100 % of the shares of each company over 1 to 4 other companies, and added $5 \times N$ products with randomly assigned producers. The results are shown in Tables 4.3 and 4.4, again with the running times of **always** and **qxp-c**.

In contrast to the taxi assignment benchmark, where all compatible sets were answer sets, now around 20 % of the solution candidates are eliminated by the unfounded set check. While **always** again significantly increases the performance, there is no clear winner among the conditions **never**, **ufs-p** and **ufs-a**. For instance sizes smaller than 16, configuration **never** is faster than **ufs-a**, with **ufs-p** falling in between. However,

for instances with more than 16 companies, this pattern is inverted and **ufs-a** exhibits a slightly better performance than the other two configurations. The reason is that for larger instances, the unfounded set search requires a larger fraction of the overall solving time. Thus, triggering backjumping earlier has a higher impact on the overall running time of the unfounded set search w.r.t. larger instances. Overall, the effect of employing partial evaluations only in the unfounded set search is small since monotonicity of the external source already allows to learn small io-nogoods that are exploited by the unfounded set search. The fact that conditions **always** and **qxp-c** are still very efficient indicates that nogoods learned in the main search help to speed up the unfounded set search as well, but not the other way around. This is also supported by the observation that exploiting early evaluations based on partial assignments both in the main and in the unfounded set search in condition **always** + **ufs-a** decreases the performance compared to **always**.

Notably, configuration **always** + **ufs-a** significantly outperforms all other conditions for computing the first answer set. This is explained by the fact that in this case, different io-nogoods are learned in the main and the unfounded set search, respectively, while the overlap increases when more answer sets are computed. Accordingly, nogoods learned in each of the two search procedures are more likely to complement each other, resulting in lower running times.

Strategic Companies with Nonmonotonic External Controls Relation

In order to test the effect of early external evaluation during the unfounded set check when the external source behaves nonmonotonically, we considered another variant of the strategic companies problem with an external controls relation. In this case, the general learning function Λ_u has to be used instead of Λ_{mu}. As a result, io-nogoods are less general because they also contain the negative input part (cf. Definition 3.7) and thus, nogoods learned in the main search are less likely to be also useful in the unfounded set search.

Here, the same problem instances as in the previous monotonic case are used, but the semantics of the oracle function associated with the external atom $\&majority[strategic](c)$ is modified as follows:

$$f_{\&majority'}(\mathbf{A}, strategic, c) = \begin{cases} \mathbf{T} & \text{if } \sum_{\mathbf{T}strategic(c_i)\in\mathbf{A}} shares(c_i, c) > 50\,\%, \text{ and} \\ & \sum_{\mathbf{T}strategic(c_i),\mathbf{U}strategic(c_i)\in\mathbf{A}} shares(c_i, c) < 100\,\%; \\ \mathbf{F} & \text{if } \sum_{\mathbf{T}strategic(c_i),\mathbf{U}strategic(c_i)\in\mathbf{A}} shares(c_i, c) \leq 50\,\%, \\ & \text{or } \sum_{\mathbf{T}strategic(c_i)\in\mathbf{A}} shares(c_i, c) = 100\,\%; \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

Accordingly, a company is only added to a candidate strategic set via the external atom if its shares owned by other companies in the set exceed 50 %, but are less than 100 %. This is motivated by the fact that selling the full shares, i.e., the entire company, might result in a higher payoff than selling only bits; hence a holding might be inclined to not keep a company that it owns fully. The corresponding results are shown in Tables 4.5 and 4.6.

#	All Answer Sets												solutions / compatible sets
	never		**always**		**ufs-p**		**ufs-a**		**always + ufs-a**		**qxp-c**		
2	**0.13**	(0)	**0.13**	(0)	**0.13**	(0)	0.14	(0)	0.14	(0)	**0.13**	(0)	1.18 / 1.68
4	**0.14**	(0)	0.17	(0)	0.15	(0)	0.18	(0)	0.20	(0)	0.17	(0)	1.70 / 2.96
6	**0.19**	(0)	0.24	(0)	0.21	(0)	0.30	(0)	0.34	(0)	0.28	(0)	2.56 / 4.46
8	**0.41**	(0)	0.43	(0)	0.44	(0)	0.63	(0)	0.66	(0)	0.56	(0)	4.20 / 6.42
10	1.48	(0)	**1.03**	(0)	1.65	(0)	1.91	(0)	1.66	(0)	1.31	(0)	7.44 / 10.10
12	5.48	(0)	**2.31**	(0)	5.65	(0)	5.68	(0)	3.33	(0)	3.14	(0)	9.30 / 13.62
14	21.73	(0)	**5.42**	(0)	21.90	(0)	18.82	(0)	8.16	(0)	8.85	(0)	18.08 / 24.46
16	82.33	(0)	**10.30**	(0)	85.19	(1)	62.76	(0)	16.58	(0)	16.81	(0)	26.96 / 35.04
18	295.92	(38)	**27.77**	(0)	276.44	(25)	223.77	(5)	31.65	(0)	51.90	(2)	36.52 / 47.56
20	300.00	(50)	**66.34**	(1)	300.00	(50)	300.00	(50)	66.86	(0)	104.89	(7)	64.30 / 79.58
22	300.00	(50)	**128.22**	(6)	300.00	(50)	300.00	(50)	130.16	(1)	141.36	(13)	≥96.84 / 116.22
24	300.00	(50)	**216.67**	(15)	300.00	(50)	300.00	(50)	237.73	(15)	234.66	(27)	≥143.32 / 195.74
26	300.00	(50)	291.04	(43)	300.00	(50)	300.00	(50)	**277.07**	(39)	278.78	(38)	≥131.10 / 311.74
28	300.00	(50)	295.25	(47)	300.00	(50)	300.00	(50)	295.78	(46)	**286.65**	(45)	≥120.28 / 280.04
30	300.00	(50)	300.00	(50)	300.00	(50)	300.00	(50)	**299.88**	(49)	300.00	(50)	≥108.38 / 267.14

Table 4.5: Results for strategic companies with nonmonotonic external controls relation (all answer sets)

We did not observe a difference regarding the number of solutions compared to the previous benchmark as 100 % of controlled shares are usually not reached w.r.t. the used instances. Nevertheless, the external source needs to ensure that this limit cannot be reached, before returning the truth value **T** for a particular company in the output. Consequently, the learning function Λ_u has to be utilized such that nogoods are typically larger than in the previous benchmark. Due to the altered semantics of the external source, running times in general increase, while the overall pattern remains similar to the one encountered for the previous benchmark. However, we observe that the running times for configuration **never** increase to a higher degree relative to the other conditions. This indicates that exploiting external evaluations w.r.t. partial assignments has an additional benefit when the external source behaves nonmonotonically. Now, for instances containing more than 12 companies we always observe an advantage of **ufs-a** and **ufs-p** over **never**, whereby **ufs-a** is faster than **ufs-p**.

Again, condition **always** + **ufs-a** proved to be very efficient for computing only the first answer set. Even though the running times for computing one solution in all other conditions significantly increase compared to the previous benchmark setting, the running times for configuration **always** + **ufs-a** are similar to before. This is because, on the one hand, less information about the external source semantics is available to the solver from the preceding search before the first answer set has been computed. On the other hand, io-nogoods learned in conditions **always** and **ufs-a**, respectively, are now less likely to be useful for the main search and the unfounded set search simultaneously, due to nonmonotonicity of the external source. In contrast, configuration **always** + **ufs-a** enables the learning of io-nogoods tailored to each of the two search procedures.

#	First Answer Set					
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**
2	0.13 (0)	0.13 (0)	**0.12** (0)	0.13 (0)	0.14 (0)	0.13 (0)
4	**0.14** (0)	0.15 (0)	**0.14** (0)	0.15 (0)	0.16 (0)	0.15 (0)
6	**0.15** (0)	0.17 (0)	0.16 (0)	0.19 (0)	0.20 (0)	0.21 (0)
8	**0.22** (0)	**0.22** (0)	0.23 (0)	0.27 (0)	0.27 (0)	0.33 (0)
10	0.46 (0)	**0.35** (0)	0.49 (0)	0.55 (0)	0.43 (0)	0.70 (0)
12	1.51 (0)	**0.63** (0)	1.57 (0)	1.53 (0)	0.73 (0)	1.62 (0)
14	3.79 (0)	1.11 (0)	3.77 (0)	3.49 (0)	**0.98** (0)	4.11 (0)
16	13.33 (0)	2.91 (0)	20.98 (1)	10.42 (0)	**1.51** (0)	10.99 (0)
18	47.45 (0)	10.23 (0)	74.04 (5)	38.08 (0)	**2.49** (0)	41.02 (2)
20	147.18 (8)	27.06 (0)	154.22 (12)	129.44 (6)	**3.18** (0)	79.82 (6)
22	241.06 (31)	58.19 (5)	242.13 (32)	237.55 (29)	**5.53** (0)	104.79 (12)
24	280.97 (45)	90.34 (7)	280.88 (44)	277.49 (42)	**6.14** (0)	179.49 (23)
26	294.46 (48)	125.64 (16)	294.04 (48)	294.23 (48)	**8.90** (0)	153.72 (21)
28	300.00 (50)	154.06 (22)	300.00 (50)	300.00 (50)	**10.48** (0)	210.73 (33)
30	300.00 (50)	194.76 (29)	300.00 (50)	300.00 (50)	**12.76** (0)	198.39 (31)

Table 4.6: Results for strategic companies with nonmonotonic external controls relation (first answer set)

#	All Answer Sets						#cyclic
	never	**never + io-dep**	**always**	**always + io-dep**	**ufs-a**	**ufs-a + io-dep**	
10	0.46 (0)	**0.43** (0)	0.60 (0)	0.58 (0)	1.53 (0)	1.36 (0)	7/10
15	2.64 (0)	**2.18** (0)	4.58 (0)	3.91 (0)	7.41 (0)	4.43 (0)	3/10
20	16.43 (0)	**14.71** (0)	44.90 (0)	41.93 (0)	43.87 (0)	31.03 (0)	5/10
25	43.85 (0)	**38.25** (0)	102.39 (1)	93.65 (1)	81.51 (0)	67.59 (0)	5/10
30	110.24 (2)	**91.01** (2)	192.48 (4)	180.58 (4)	168.80 (2)	99.53 (2)	4/10
35	111.62 (1)	**79.69** (1)	217.58 (4)	178.62 (2)	161.86 (2)	83.18 (1)	3/10
40	189.64 (2)	**141.12** (2)	262.35 (6)	231.22 (5)	202.95 (3)	143.12 (2)	5/10
45	264.04 (5)	216.89 (4)	269.49 (6)	227.88 (5)	263.40 (5)	**202.55** (4)	5/10
50	300.00 (10)	227.15 (4)	300.00 (10)	249.55 (6)	300.00 (10)	**220.61** (3)	2/10

Table 4.7: Results for user access selection (all answer sets; few cycles)

$$y\_nd(X) \vee n\_nd(X) \leftarrow domain(X). \qquad \leftarrow nd(X), nd\_f(X).$$
$$nd(X) \leftarrow y\_nd(X). \qquad \leftarrow \text{not } nd(X), nd\_a(X).$$
$$nd(X) \leftarrow \&hasAccess[nd](X). \qquad \leftarrow \#count\{X{:}y\_nd(X)\} > 3.$$

Figure 4.3: User Access Selection Rules

4.3.4 Experiments on Minimality Check Skipping

In this section, we discuss the experiments on pruning e-minimality checks by exploiting semantic dependencies, and report the according results.

User Access Selection

Consider a computer network (C, A) represented by a set of computer nodes C and a set of directed access connections A between nodes, where $n_1 \to n_2 \in A$, for $n_1, n_2 \in C$, holds if and only if node n_1 has access to node n_2. Hence, a node can be accessed directly,

#	All Answer Sets												#cyclic
	never		**never + io-dep**		**always**		**always + io-dep**		**ufs-a**		**ufs-a + io-dep**		
10	0.41	(0)	0.41	(0)	**0.35**	(0)	0.36	(0)	0.46	(0)	0.46	(0)	10/10
15	7.55	(0)	7.61	(0)	**6.17**	(0)	6.38	(0)	7.95	(0)	8.15	(0)	10/10
20	44.03	(1)	43.92	(1)	**6.52**	(0)	6.57	(0)	44.54	(1)	44.66	(1)	10/10
25	107.50	(2)	107.95	(2)	**51.60**	(1)	51.62	(1)	87.53	(1)	87.51	(1)	10/10
30	84.97	(0)	84.64	(0)	**44.23**	(0)	44.73	(0)	85.64	(0)	85.42	(0)	10/10
35	223.56	(5)	222.95	(5)	**111.29**	(1)	110.98	(1)	223.26	(5)	224.26	(5)	10/10
40	268.27	(7)	268.73	(7)	**152.53**	(1)	153.28	(1)	268.86	(7)	269.44	(7)	10/10
45	284.12	(8)	284.33	(8)	**251.08**	(4)	252.54	(4)	286.90	(8)	286.56	(8)	10/10
50	300.00	(10)	300.00	(10)	300.00	(10)	**298.61**	(9)	300.00	(10)	300.00	(10)	10/10

Table 4.8: Results for user access selection (all answer sets; many cycles)

or indirectly via a sequence of intermediate nodes. Now, suppose the task of a network administrator is to assign access rights by selecting a subset C_g of C to which some user will be granted access, whereby the user requires access to a set of nodes $C_a \subseteq C$ and is not permitted to access nodes from a set $C_f \subseteq C$ disjoint from C_a. Thus, the problem formally consists in selecting nodes $C_g \subseteq C$ s.t. every node in C_a is reachable from some node in C_g and no node in C_f is reachable from any node in C_g via edges in A.

In our problem setting, we assume the network is not known initially, but each node can be queried for the set of nodes it can access directly. For this, we use an external atom $\&hasAccess[nodes](n)$, which interfaces external network information, and outputs all nodes that can be accessed by some node in the extension of *nodes*. Accordingly, it evaluates to true for an output node n_2 w.r.t. an assignment $\mathbf{A}$ iff $\mathbf{T}nodes(n_1) \in \mathbf{A}$ for some $(n_1, n_2) \in A$. Moreover, we specify $D(\&hasAccess[nodes]) = \{\langle 2, 1 : \{n_1 \mid n_1 \rightarrow n_2 \in A\}, 1 : n_2\rangle \mid n_2 \in C\}$, i.e. there is a dependency of an output on an input node whenever the latter has access to the former. The HEX-program in Figure 4.3 with facts $domain(n)$ for $n \in C$, facts $node\_a(n)$ for $n \in C_a$, and facts $node\_f(n)$ for $n \in C_f$ encodes user access selection, where at most three nodes can be accessed directly.

First, we randomly generated networks with $N \in [10, 50]$ nodes, where each node has access to another node with probability $\frac{1}{2\times N}$ (cf. Table 4.7). As visible from the rightmost column in Table 4.7, this yields networks roughly half of which have no cyclic access relations and thus, dependency pruning can have an effect on the number of required e-minimality checks. Next, we increased the access probability to $\frac{2}{N}$ (cf. Table 4.8), which effects that all generated instances contain cycles. This allowed us to investigate the effect of pruning when this does not impact the need for an e-minimality check.

Finally, we generated instances again with access probability $\frac{1}{2\times N}$, but removed all cyclic instances (cf. Table 4.9). The goal was to test instances where the e-minimality check can always be skipped, in order to ascertain the maximum speedup obtainable by pruning the dependency graph. The rightmost column in the tables shows the fraction of instances where the computer network contains a cycle.

#	All Answer Sets						#cyclic
	never	**never + io-dep**	**always**	**always + io-dep**	**ufs-a**	**ufs-a + io-dep**	
10	0.37 (0)	0.30 (0)	0.52 (0)	0.45 (0)	0.82 (0)	**0.29** (0)	0/10
15	1.48 (0)	**0.98** (0)	2.62 (0)	2.05 (0)	3.16 (0)	0.99 (0)	0/10
20	5.82 (0)	**3.28** (0)	14.11 (0)	12.51 (0)	10.98 (0)	3.37 (0)	0/10
25	42.02 (1)	**8.59** (0)	88.35 (1)	55.00 (0)	39.17 (0)	8.87 (0)	0/10
30	29.32 (0)	**18.83** (0)	124.64 (2)	115.18 (2)	84.21 (0)	18.88 (0)	0/10
35	73.73 (0)	**42.48** (0)	167.40 (3)	155.55 (3)	135.24 (3)	43.23 (0)	0/10
40	182.62 (2)	**100.21** (0)	238.41 (5)	209.27 (4)	189.40 (3)	103.89 (0)	0/10
45	226.40 (2)	**131.75** (1)	253.80 (5)	205.97 (5)	230.21 (3)	131.95 (1)	0/10
50	226.55 (5)	187.30 (2)	223.71 (5)	198.84 (3)	222.62 (5)	**179.50** (2)	0/10

Table 4.9: Results for user access selection (all answer sets; no cycles)

$$
\begin{aligned}
turn(a\_1, P) \vee turn(a\_2, P) \leftarrow & position(P).\\
picked(A, P, G) \leftarrow & \&pick[alreadyPicked](A, P, G), turn(A, P), item(G).\\
alreadyPicked(P, G) \leftarrow & position(P), position(P1), P1 < P, picked(\_, P1, G).\\
\leftarrow & \text{ not } \&envyFree[picked]().
\end{aligned}
$$

Figure 4.4: Sequential Allocation Rules

Sequential Allocation of Indivisible Goods

Next, we considered a problem from *social choice*, namely dividing a set G of m items among two agents a_1 and a_2 by allowing them to pick items in specific sequences $\sigma = o_1 o_2 ... o_m \in \{a_1, a_2\}^m$ (Kalinowski et al., 2013). Each agent a_i has a linear preference order $>_i$ over G; and the utility of $g \in G$ for a_i is $u_i(g) = |\{g' \mid g >_i g' \in G\}|$. We assume that an agent always picks the remaining item with maximal utility. The goal is to find a sequence σ resulting in an *envy-free* division of items, i.e. where no agent prefers the items of the other agent over its own items.

We use an external atom to obtain the choices of the agents, while their complete preferences are hidden, and a further one that checks whether an allocation is envy-free. The atom $\&pick[alreadyPicked](a_i, p, g)$ evaluates to true w.r.t. assignment $\mathbf{A}$ iff $p \in [1, m]$ and $g >_i g'$ for all g' such that $\mathbf{T}alreadyPicked(p{-}1, g) \notin \mathbf{A}$, where p represents the positions in a respective sequence. Furthermore, let $G(\mathbf{A}, i, j) = \sum_{g \in \{g \mid \mathbf{T}picked(a_i, p, g) \in \mathbf{A}\}} u_j(g)$. Then, the atom $\&envyFree[picked]()$ is true iff $G(\mathbf{A}, 1, 1) < G(\mathbf{A}, 2, 1)$ and $G(\mathbf{A}, 2, 2) < G(\mathbf{A}, 1, 2)$. The encoding is shown in Figure 4.4. Together with facts $position(p)$ and $item(g)$ for all $p, g \in [1, m]$, its answer sets encode all sequences that induce an envy-free allocation.

We specified the io-dependencies $D(\&pick[alreadyPicked]) = \{\langle 1, 1{:}\{p\}, 2{:}p+1\rangle \mid 1 \leq p < m\}$, i.e. items already picked at a sequence position only depend on the previous positions. The io-dependencies eliminate all cyclic dependencies via external atoms in the problem instances; thus e-minimality checks can always be skipped. We tested instances with random preference orders and $N \in [3, 10]$ items. The results are shown in Table 4.10.

#	All Answer Sets					
	never	**never + io-dep**	**always**	**always + io-dep**	**ufs-a**	**ufs-a + io-dep**
3	**0.19** (0)	**0.19** (0)	0.25 (0)	0.24 (0)	0.38 (0)	**0.19** (0)
4	2.74 (0)	1.73 (0)	0.74 (0)	**0.64** (0)	2.54 (0)	1.72 (0)
5	300.00 (10)	78.28 (0)	152.33 (5)	**2.42** (0)	141.76 (1)	78.02 (0)
6	300.00 (10)	300.00 (10)	300.00 (10)	**8.22** (0)	300.00 (10)	300.00 (10)
7	300.00 (10)	300.00 (10)	300.00 (10)	**26.63** (0)	300.00 (10)	300.00 (10)
8	300.00 (10)	300.00 (10)	300.00 (10)	**89.97** (0)	300.00 (10)	300.00 (10)
9	300.00 (10)	300.00 (10)	300.00 (10)	**284.17** (4)	300.00 (10)	300.00 (10)
10	**300.00** (10)	**300.00** (10)	**300.00** (10)	**300.00** (10)	**300.00** (10)	**300.00** (10)

Table 4.10: Results for sequential allocation of indivisible goods (all answer sets)

4.3.5 Discussion of Results

In this section, we first summarize our insights regarding experiments on partial evaluation during the unfounded set search w.r.t. hypotheses (H4.1) to (H4.3). Afterwards, our findings regarding minimality check skipping based on the results from Section 4.3.4 are discussed.

Findings Regarding Partial Evaluation in the Unfounded Set Search

In all benchmarks considered for testing partial evaluation in the unfounded set search, we found that at least one of the configurations **ufs-p** and **ufs-a** improved the performance over **never**, except for smaller instances where the differences are generally small (cf. Tables 4.1 to 4.6). However, the improvement was not as significant as the one we found for partial evaluation in the main search in Chapter 3, and depends on how much room there is for decreasing the running time of the unfounded set search by detecting conflicts early. Thus, hypothesis (H4.1) is partly supported by our experimental results.

In the experiments employing strategic companies problems, where external calls are inexpensive, condition **ufs-a** showed lower running times than **ufs-p** for larger instances (cf. Tables 4.3, 4.4, 4.5 and 4.6), while for the taxi benchmark with more costly external evaluations, it was the other way around (cf. Tables 4.1 and 4.2). This is in line with hypotheses (H4.1) and (H4.2). In support of hypothesis (H4.2), **ufs-p** also performs better than **ufs-a** for small instances of the strategic companies benchmarks, where the information obtained from external calls is less useful due to less time required by the minimality check and low usefulness of learned nogoods w.r.t. the main search.

Finally, we found that for computing all answer sets, the combination **always** + **ufs-a** mostly decreased the efficiency compared to **always**. However, Tables 4.2, 4.4 and 4.6 show that the combination can be very efficient when only the first answer set is computed, where the sets of io-nogoods learned in the main and the unfounded set search, respectively, are less likely to overlap. Moreover, we observe in Table 4.6 that the combination **always** + **ufs-a** has an even higher advantage for computing the first answer set when a nonmonotonic external source is accessed as this as well increases the chance of learning different nogoods in the main search and the unfounded set search, respectively. Accordingly, our experiments confirm hypothesis (H4.3).

We conclude that even when partial evaluation during the unfounded set search does not increase the efficiency for computing all answer sets compared to other configurations, it can be highly effective in combination with partial evaluation during the main search in case only one solution is required.

Findings Regarding Minimality Check Skipping

We observed that when dependency graph pruning skips e-minimality checks, **io-dep** significantly improves the running times for all instance sizes and independent from the configuration it is combined with (cf. Tables 4.7, 4.9 and 4.10). Accordingly, the results support hypothesis (H4.4).

In Table 4.9, we observe an increased benefit of **io-dep** due to the absence of cyclic instances. As expected, by testing single instances, we found that **io-dep** reduces the running times roughly by the amount required for redundant e-minimality checks. In Table 4.8, only a negligible impact on the running time is visible when **io-dep** is added. Since **io-dep** has no advantage for cyclic instances, this shows that pruning the dependency graph yields no significant overhead and provides evidence for the correctness of (H4.6).

Partial evaluation was only beneficial both in the model search and the e-minimality check for the sequential allocation benchmark, while only configuration **always** resulted in a speedup in the second experiment. The reason for partial evaluation increasing the running time in all other cases is that the overhead that results from additional external calls did not outweigh the benefit in terms of additional information gain. As predicted by hypothesis (H4.5), the speedup for **always**+**io-dep** is larger than for **ufs-a**+**io-dep** because **ufs-a** already reduces the running time required for e-minimality checks, while condition **always** needs to invest more time in the e-minimality check. The running times for **never**+**io-dep** and **ufs-a**+**io-dep** are similar; this is expected as **min-part** only applies to the e-minimality check, which is skipped in both conditions.

In summary, even though there is no clear winner among the conditions, adding **io-dep** is suggestive as a default when io-dependencies can be specified.

4.4 Related Work

To the best of our knowledge, external minimality has not been considered in other approaches that integrate external theories into declarative problem solving (cf. Section 1.2.1), such as clingo (Gebser et al., 2016), *SMT* (Barrett et al., 2009) and *constraint ASP* (Lierler, 2014). Accordingly, a distinguishing difference between hex and theory solving as realized in clingo 5 concerns the actual semantics of programs. Roughly speaking, clingo 5 fixes a valuation of the theory atoms and computes then an answer set of the program relative to this valuation; this amounts to using a GL-style reduct where theory atoms are removed from rules. In contrast, for evaluating hex-programs, all external atoms remain in the rules, according to the FLP-reduct. While clingo 5 makes no further minimality check, the e-minimality check for hex-programs may eliminate

candidate answer sets. For example, CLINGO 5 would return $\mathbf{A}_2 = \{\mathbf{T}p\}$ as an answer set for the program $\Pi = \{p \leftarrow \&id[p]()\}$ (adapted to the different formalism), which is eliminated by the e-minimality check of HEX.

Nevertheless, our techniques could also be employed directly by related rule-based formalisms if minimality involving external theories is required. Moreover, cyclic support may arise from *external propagators*, e.g. in the *WASP*-solver (Dodaro et al., 2016), where our approach could be applied as well.

In ordinary ASP-solving, one can distinguish unfounded sets due to positive cycles and unfounded sets due to disjunctions with head cycles. The check for the former is done *a priori* over partial assignments and is feasible in linear time; the check for the latter is coNP-hard and typically done *a posteriori* under complete assignments only. We performed some experiments with searching for unfounded sets over partial assignments, i.e., running Algorithm 4.1 also over partial input assignments, but it soon turned out that the additional cost of more e-minimality checks exceeded the benefits by far. Also Gebser, Kaufmann, and Schaub (2013) conducted some experiments with unfounded set checking for disjunctive ordinary ASP over partial assignments. However, they also reported only moderate computational benefits, although their minimality check is cheaper than ours due to absence of external calls. Therefore, we did not pursue the idea of searching for unfounded sets over partial assignments further.

Finally, our technique for pruning e-minimality checks based on semantic dependency information is related to *domain independence* techniques in (Eiter, Fink, & Krennwallner, 2009), where external atoms are evaluated w.r.t. subsets of the domain while correct outputs are retained. This is similar to our notions of compliant atoms and faithfulness. Yet, io-dependencies are more general because in (Eiter et al., 2009), only disjoint domain partitions for external inputs are considered, and dependencies are not used for argument positions. Another important difference is that their approach employs dependencies for *program splitting* as in (Lifschitz & Turner, 1994), while we aim at detecting redundant e-minimality checks. They do not analyze the costs for generating dependencies.

4.5 Conclusion and Outlook

In this chapter, we have first extended external evaluations based on partial assignments, introduced in Chapter 3, to the unfounded set search, which constitutes an efficient realization of the e-minimality check of HEX. For this, we have developed a new variant of the algorithm for e-minimality checking of HEX-programs, which can now also evaluate external atoms w.r.t. partial model candidates.

Subsequently, we introduced io-dependencies to formalize semantic dependencies over external atoms that approximate the real dependencies more closely than previously possible. Based on this, more e-minimality checks can be skipped, which proved to be beneficial in practice. We also stated properties for checking and optimizing io-dependencies important for automatically constructing tight faithful dependency sets. While faithfulness checking is intractable in general, we identified cases where the costs can be reduced for certain oracles, or where checking is polynomial.

While we only exploited semantic dependencies for e-minimality checking, additional dependency information is also useful for other parts of HEX-solving such as grounding and *external behavior learning* (Eiter et al., 2018). By limiting oracle calls to compliant input atoms, the number of external calls during HEX-evaluation could potentially be reduced significantly, where oracle calls could be restricted to compliant atoms. This has the potential to significantly reduce the number of external calls required during HEX-evaluation; this is expected to be especially beneficial when computing the values of oracle functions takes long. In addition, partial evaluation could in turn be leveraged to learn io-dependencies on-the-fly from external calls.

CHAPTER 5

Integration of Grounding and Solving

As for propositional solving and minimality checking, which require a tight integration with external evaluation as described in Chapters 3 and 4, efficient grounding of HEX-programs is also more challenging than in the case of ordinary ASP. This is mainly due to the generic (previously often black-box) nature of external atoms and value invention, i.e., the import of new constant symbols from the sources into the program. In this chapter, we present an approach that improves the grounding of HEX-programs by tightly integrating the grounding with the solving process as well as with external evaluation, which was developed in (Eiter, Kaminski, & Weinzierl, 2017).

HEX-programs inherit the well-known *grounding bottleneck* of state-of-the-art ASP-solving, as e.g. by CLINGO (Gebser et al., 2016), which may show up in the grounding phase, i.e. during the computation of a propositional program equivalent to the input program, and can cause an exponential blowup. This makes ASP and likewise HEX incapable of solving a number of real-world problems with larger data volume. To mitigate this problem, several advanced optimization methods and techniques have been developed, cf. (Kaufmann et al., 2016), but the grounded program can still be (too) large.

Since grounding of HEX-programs is due to external atoms an even bigger challenge, special program decomposition and component grounding techniques for HEX have been developed as well (cf. Section 1.2.2), which mitigate grounding issues in many cases (Eiter, Fink, Ianni, et al., 2016). However, program decomposition has still a tradeoff with efficient solving because it may split integrity constraints from guesses in a HEX-program such that the former cannot be propagated effectively during solving; thus, the grounding bottleneck is often replaced by a solving bottleneck (Redl, 2017a). At the same time, while monotonic external atoms can be grounded efficiently, exponentially many inputs to a nonmonotonic external atom may have to be considered during grounding when program decomposition is not applied.

For instance, similar to an example by Redl (2017a), consider the following HEX-program for configuring a server cluster:

$$
\begin{aligned}
&comp(a).\ \ comp(b).\ \ comp(c).\\
&config(X) \vee n\_config(X) \leftarrow\ comp(X).\\
&prop(P) \leftarrow\ \&prop[config](P).\\
&\leftarrow\ prop(high\_cost), \text{not } prop(low\_availability).
\end{aligned}
$$

For a set C of selected components in the extension of the predicate $config$ in a candidate solution, the external atom $\&prop[config](P)$ retrieves from an external property checker each property P of C. At this, properties may depend nonmonotonically on the input configuration, e.g. adding components might make the cluster more powerful but may also decrease maintainability. Here, a cluster configuration is desired that does not have low availability and high costs at the same time.

As the external atom used in the example is nonmonotonic, it has to be evaluated w.r.t. all possible combinations of components during grounding (each configuration might produce a new output value). Hence, when more components need to be considered, this quickly results in an explosion of the grounding costs. However, for finding only one solution or when certain combinations of components are excluded by an additional constraint, usually not all configurations have to be tested. Accordingly, the program decomposition by Eiter, Fink, Ianni, et al. (2016) splits the first and the second rule into two components that are solved and grounded separately, exploiting the fact that the first rule does not depend on the second.

More precisely, answer sets of the program consisting of only the first rule and the facts are computed in a first step, and the rest of the program is then evaluated by separately extending it with each of the answer sets. Simultaneously, the integrity constraint is also split from the first guessing rule, while many configurations may not be relevant because a (small) subset of its components already induces some undesired property. Consequently, this fact cannot be exploited for efficient solving anymore after splitting, such that novel evaluation algorithms that avoid the corresponding tradeoff by integrating grounding and solving of HEX-programs more tightly are needed.

To overcome the grounding bottleneck of ASP, lazy-grounding algorithms were devised, that ground rules *on-the-fly* (Palù et al., 2009; Lefèvre & Nicolas, 2009a, 2009b; Dao-Tran et al., 2012; Lefèvre et al., 2017). In an interleaved grounding and solving process, only rules are grounded that are currently useful and thus space explosion is avoided. In this way, problems can be solved that traditional ASP-solving cannot handle. Recent advances in lazy grounding, available in prototype solvers, suggest to explore this approach for evaluating HEX-programs. However, an extension to this setting is non-trivial, due to nonmonotonic dependencies of external atoms on absent information, and in particular due to unknown constants from value invention.

In order to overcome the tradeoff between solving and grounding discussed above, we extend the lazy-grounding approach to HEX-programs by integrating the recent lazy-grounding solver ALPHA (Weinzierl, 2017) into DLVHEX. While earlier lazy-grounding

ASP-solvers are lacking effective conflict-driven learning techniques, which makes them less promising for our purpose, the ALPHA-solver combines lazy-grounding with modern ASP-solving techniques. By employing ALPHA as an alternative backend-solver, which requires the design of a new evaluation algorithm, the usual grounding step can be omitted. The integration enables HEX to handle new classes of problems which could not be solved efficiently before due to the grounding bottleneck.

The rest of this chapter is structured as follows follows:

- In Section 5.1, we show under which conditions an assignment that is defined over an incomplete domain is *input-complete*, i.e. when it can safely be used to evaluate external atoms during lazy-grounding evaluation. Moreover, we introduce *input-safe domains*, which are used to ensure input-completeness of assignments, and show that one can always be obtained by computing a partial *relevant grounding*.

- In Section 5.2, we first develop a novel external source interface to incrementally extend a HEX-program grounding, where new output terms can be generated on-the-fly *during solving*. Subsequently, we present our novel lazy-grounding evaluation algorithm for HEX-programs that can leverage lazy-grounding ASP-solvers such as ALPHA.

- In Section 5.3, we show experimental results which confirm the benefit of the novel algorithm and the integration of ALPHA as backend-solver on illustrative benchmarks. To this end, we also compare the new approach with program decomposition techniques and show that the previous tradeoff between grounding and solving can effectively be avoided.

- In Section 5.4, we discuss related work and conclude in Section 5.5.

The unprecedented integration of lazy-grounding, external source evaluation and value invention is a new perspective to make HEX and ASP more suitable for real-world applications.

5.1 Evaluation of External Sources Based on Partial Groundings

Lazy-grounding means that the grounding $grnd(\Pi)$ of a program Π is computed lazily, i.e., only ground rules deemed necessary are computed. In the following, let $\mathcal{G}_\Pi \subseteq \mathcal{HB}$ denote the set of all atoms occurring in the grounding of Π. Then, partial assignments in the case of lazy-grounding are given with respect to a set of ground atoms $\mathcal{A} \subseteq \mathcal{G}_\Pi \subseteq \mathcal{HB}$.

Definition 5.1 (Partial Assignment Over a Set of Atoms)**.** *A* partial assignment *over a set $\mathcal{A} \subseteq \mathcal{HB}$ of atoms is a set $\mathbf{A}_\mathcal{A}$ of signed literals $\mathbf{T}a$, $\mathbf{F}a$, and $\mathbf{U}a$ with $a \in \mathcal{A}$ s.t. for every $a \in \mathcal{A}$, $|\mathbf{A} \cap \{\mathbf{T}a, \mathbf{F}a, \mathbf{U}a\}| = 1$; it is* complete *(w.r.t. $\mathcal{A}$), if no $\mathbf{U}a$ occurs in it.*

For partial assignments $\mathbf{A}_\mathcal{A}, \mathbf{A}'_{\mathcal{A}'}$ we call $\mathbf{A}'_{\mathcal{A}'}$ an *extension* of $\mathbf{A}_\mathcal{A}$, denoted $\mathbf{A}'_{\mathcal{A}'} \succeq \mathbf{A}_\mathcal{A}$, if $\{\mathbf{T}a \in \mathbf{A}_\mathcal{A}\} \cup \{\mathbf{F}a \in \mathbf{A}_\mathcal{A}\} \subseteq \mathbf{A}'_{\mathcal{A}'}$ and $\mathcal{A} \subseteq \mathcal{A}'$, i.e., some atoms $a \in \mathcal{HB}$ not present in $\mathbf{A}_\mathcal{A}$ may be present in $\mathbf{A}'_{\mathcal{A}'}$ and some unassigned atoms in $\mathbf{A}_\mathcal{A}$ may be flipped to true or false.

Lazy-grounding ASP-solving is founded on the notion of a *computation sequence*, cf. (Lefèvre & Nicolas, 2009a), which is a monotonically growing sequence $(\mathbf{A}_0, \ldots, \mathbf{A}_n)$ of partial assignments such that whenever a (lazily grounded) rule of the input program fires at $\mathbf{A}_i$, it is guaranteed to fire in all later assignments $\mathbf{A}_j$, i.e., $0 \leq i \leq j \leq n$. Furthermore, a grounded rule r only fires in $\mathbf{A}_i$ if it is *applicable*, which means that its positive body $B^+(r)$ is completely true, i.e., $\mathbf{A}_i \models B^+(r)$. Given that ordinary ASP rules are safe, the whole negative body of a ground rule is known once it fires, such that focusing on positive rule bodies is sufficient for completeness of solving under lazy-grounding. For example, if the rule $p(a) \leftarrow q(a), \mathrm{not}\, r(a)$ fires in a computation sequence at $\mathbf{A}_i$, then $\{\mathbf{T}q(a), \mathbf{F}r(a)\} \subseteq \mathbf{A}_i$ and monotonicity guarantees the same for all later $\mathbf{A}_j$, $i \leq j$.

5.1.1 Safety Condition

For (ground) external atoms, it is much harder to ensure that once an output becomes true, it will stay true even for larger input. Recall the notion of assignment-monotonic oracle-function from Chapter 3, which has been used to ensure the former in the case where the grounding of a program is generated prior to solving. Formally, a three-valued oracle function $f_{\&g}$ is *assignment-monotonic*, if $f_{\&g}(\mathbf{A}_{\mathcal{G}_\Pi}, \vec{p}, \vec{c}) = X$, $X \in \{\mathbf{T}, \mathbf{F}\}$, implies $f_{\&g}(\mathbf{A}'_{\mathcal{G}_\Pi}, \vec{p}, \vec{c}) = X$ for all partial assignments $\mathbf{A}'_{\mathcal{G}_\Pi} \succeq \mathbf{A}_{\mathcal{G}_\Pi}$. Intuitively, this guarantees that the oracle-function cannot treat $\mathbf{T}a \notin \mathbf{A}_{\mathcal{G}_\Pi}$ as being equivalent to $\mathbf{F}a \in \mathbf{A}_{\mathcal{G}_\Pi}$. Observe that all atoms $a \in \mathcal{HB} \setminus \mathcal{G}_\Pi$ must be false in every answer set, simply because there is no rule in the grounding of Π whose head is a. As all assignments for the latter program are over $\mathcal{G}_\Pi$, it is thus guaranteed that each atom a relevant for the external source also occurs in $\mathbf{A}_{\mathcal{G}_\Pi}$, either as $\mathbf{T}a$, $\mathbf{F}a$, or $\mathbf{U}a$. For an $\mathbf{A}_\mathcal{A}$ with $\mathcal{A} \subset \mathcal{G}_\Pi$ and $a \in \mathcal{G}_\Pi \setminus \mathcal{A}$, however, an oracle-function treats a as false, i.e., $\mathbf{A}_\mathcal{A}$ equals $(\mathbf{A} \cup \{\mathbf{F}a\})_{\mathcal{A} \cup \{a\}}$ from the perspective of any oracle-function, even for assignment-monotonic ones.

As oracle-functions are black-boxes, HEX cannot determine the relevant input of an assignment-monotonic oracle-function, i.e.: if $f_{\&g}(\mathbf{A}_\mathcal{A}, \vec{p}, \vec{c}) = \mathbf{T}$ for an assignment $\mathbf{A}_\mathcal{A}$, then, without knowing the set of ground atoms $\mathcal{G}_\Pi$ that occur in the grounding of a HEX-program Π, we cannot determine whether some atom $a \in \mathcal{G}_\Pi \setminus \mathcal{A}$ exists s.t. $f_{\&g}((\mathbf{A} \cup \{\mathbf{U}a\})_{\mathcal{A} \cup \{a\}}, \vec{p}, \vec{c}) \neq \mathbf{T}$.

Example 5.1. Consider the program $\Pi = \{\leftarrow \&size[p](0);\ p(X) \leftarrow d(X);\ a \leftarrow \mathrm{not}\, d(c);\ d(c) \leftarrow \mathrm{not}\, a\}$ where the external atom $\&size[p](Z)$ computes the cardinality of p, i.e., $f_{\&size}(\mathbf{A}, p, Z) = \mathbf{U}$ if $\mathbf{U}p(X) \in \mathbf{A}$ for some X, $f_{\&size}(\mathbf{A}, p, Z) = \mathbf{T}$ if $|\{p(X) \mid \mathbf{T}p(X) \in \mathbf{A}\}| = Z$ and $\mathbf{U}p(X) \notin \mathbf{A}$ for every X, and $f_{\&size}(\mathbf{A}, p, Z) = \mathbf{F}$ otherwise. The single answer set of Π is $\{\mathbf{T}d(c), \mathbf{T}p(c), \mathbf{F}a\}$, because it satisfies the first rule, which expresses that the extension of predicate p must not be 0.

Now, assume that $\&size[p](0)$ is evaluated before the first guess. Then, $\&size[p](0)$ is true under $\mathbf{A}_\mathcal{A} = \{\mathbf{U}d(c), \mathbf{U}a\}$ with $\mathcal{A} = \{d(c), a\}$. However, guessing $\mathbf{T}d(c)$ and

grounding the second rule for $X = c$ yields $\mathcal{A}' = \mathcal{A} \cup \{p(c)\}$. For $\mathbf{A}'_{\mathcal{A}'} = \mathbf{A}_{\mathcal{A}} \cup \{\mathbf{U}p(c)\}$, we obtain $f_{\&size}(\mathbf{A}'_{\mathcal{A}'}, p, 0) = \mathbf{U}$, which shows that $\mathbf{A}_{\mathcal{A}}$ was insufficient for deciding the value of $\&size[p](0)$. △

To address this issue, we must ensure that external atoms are evaluated only with assignments being complete for their input predicates. Intuitively, an assignment is input-complete for a program, if it contains all relevant input to every external atom; a ground atom that occurs in no answer set and is not an input-predicate of any external atom is irrelevant for the truth of external atoms and thus ignored. For a partial assignment $\mathbf{A}_{\mathcal{A}}$, let its completion w.r.t. $\mathcal{HB}$ be $\hat{\mathbf{A}}_{\mathcal{A}} = (\mathbf{A} \cup \{\mathbf{F}a \mid a \in \mathcal{HB} \setminus \mathcal{A}\})_{\mathcal{HB}}$. Then input-completeness is as follows:

Definition 5.2 (Input-Completeness)**.** *A partial assignment* $\mathbf{A}_{\mathcal{A}}$ *is* input-complete *for an external atom* $\&g[\vec{p}](\vec{c})$ *occurring in a ground* HEX*-program* Π*, if* $f_{\&g}(\hat{\mathbf{A}}_{\mathcal{A}}, \vec{p}, \vec{c}) = \mathbf{T}$ *only if every answer set* $\mathbf{A}'_{\mathcal{HB}}$ *of* Π *s.t.* $\mathbf{A}'_{\mathcal{HB}} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$ *fulfills* $f_{\&g}(\mathbf{A}'_{\mathcal{HB}}, \vec{p}, \vec{c}) = \mathbf{T}$*, where the assignment* $\mathbf{A}_{\vec{p},\mathcal{A}} = \{Xa \in \mathbf{A}_{\mathcal{A}} \mid a \text{ has predicate } p \in \vec{p}\}$ *constitutes the relevant input to* $\&g$*. A partial assignment* $\mathbf{A}_{\mathcal{A}}$ *is* input-complete *w.r.t. a* HEX*-program* Π*, if it is* input-complete *for each external atom* $\&g[\vec{p}](\vec{c})$ *occurring in* $grnd(\Pi)$*.*

Without the restriction to answer sets, in Example 5.1 no input-complete assignment $\mathbf{A}_{\mathcal{A}}$ on $\mathcal{A} \subset \mathcal{HB}$ for $\&size[p](0)$ would exist with $f_{\&size}(\mathbf{A}, p, 0) = \mathbf{T}$, as infinitely many constants could be added to $p's$ extension when the domain is expanded.

Example 5.2 (cont'd). There is no partial assignment defined over $\{d(c), a\}$ that is input-complete for Π, but the partial assignment $\{\mathbf{T}d(c), \mathbf{T}p(c), \mathbf{F}a\}$ is input-complete for Π. △

In the following, we syntactically characterize sets of ground atoms that are sufficient for input-completeness, resorting to nonmonotonic inputs to external atoms; monotonic inputs cannot cause issues with atoms $a \in \mathcal{HB} \setminus \mathcal{A}$ that do not yet occur in an assignment $\mathbf{A}_{\mathcal{A}}$. Recall that, formally, an input predicate $p \in \vec{p}$ of an external atom $\&g[\vec{p}](\vec{c})$ is *monotonic*, if $f_{\&g}(\mathbf{A}_{\mathcal{A}}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\mathbf{A}'_{\mathcal{A}'}, \vec{p}, \vec{c}) = \mathbf{T}$ for any $\mathbf{A}'_{\mathcal{A}'} \succeq \mathbf{A}_{\mathcal{A}}$ that augments a given $\mathbf{A}_{\mathcal{A}}$ only by atoms with predicate p, cf. (Eiter, Fink, Krennwallner, & Redl, 2016).

It can be shown that if an external atom which only has monotonic input predicates evaluates to $\mathbf{T}$ w.r.t. any partial domain, then this output is correct since by extending the respective domain, it cannot be changed from $\mathbf{T}$ to $\mathbf{F}$. This is formalized by the following proposition.

Proposition 5.1. *Let* $\&g[\vec{p}](\vec{c})$ *be an external atom occurring in a ground* HEX*-program* Π *and let each* $p \in \vec{p}$ *be monotonic. Then, any partial assignment* $\mathbf{A}_{\mathcal{A}}$ *is input-complete for* $\&g[\vec{p}](\vec{c})$*.*

Proof. Let $\&g[\vec{p}](\vec{c})$ be an external atom as in the hypothesis. Further, let $\mathbf{A}_{\mathcal{A}}$ be an arbitrary partial assignment. To show that $\mathbf{A}_{\mathcal{A}}$ is input-complete for $\&g[\vec{p}](\vec{c})$, we need to show that $f_{\&g}(\hat{\mathbf{A}}_{\mathcal{A}}, \vec{p}, \vec{c}) = \mathbf{T}$ only if every answer set $\mathbf{A}'_{\mathcal{HB}}$ of Π s.t. $\mathbf{A}'_{\mathcal{HB}} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$ fulfills $f_{\&g}(\mathbf{A}'_{\mathcal{HB}}, \vec{p}, \vec{c}) = \mathbf{T}$.

Here, we prove a stronger result, namely, that for every subset $\mathbf{A}'_{\mathcal{A}'}$ of an answer set of Π with $\mathbf{A}'_{\mathcal{A}'} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$ the following holds: (*) $f_{\&g}(\hat{\mathbf{A}}'_{\mathcal{A}'}, \vec{p}, \vec{c}) = \mathbf{T}$ if $f_{\&g}(\hat{\mathbf{A}}_{\mathcal{A}}, \vec{p}, \vec{c}) = \mathbf{T}$.

Let $\mathbf{A}'_{\mathcal{A}'}$ be an arbitrary subset of an answer set of Π. We prove this by induction on the cardinality $n = |\mathbf{A}'_{\mathcal{A}'}|$ of $\mathbf{A}'_{\mathcal{A}'}$. It is not possible that $|\mathbf{A}'_{\mathcal{A}'}| < |\mathbf{A}_{\vec{p},\mathcal{A}}|$ as we have that $\mathbf{A}'_{\mathcal{A}'} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$; hence, we only need to consider $n \geq |\mathbf{A}_{\vec{p},\mathcal{A}}|$.

The base case is $n = |\mathbf{A}'_{\mathcal{A}'}| = |\mathbf{A}_{\vec{p},\mathcal{A}}|$. In this case, (*) follows directly due to assignment monotonicity, and because the value of $f_{\&g}(\hat{\mathbf{A}}_{\mathcal{A}}, \vec{p}, \vec{c})$ only depends on the extension of predicates $p \in \vec{p}$ in $\hat{\mathbf{A}}_{\mathcal{A}}$ (cf. Section 2).

Now, we assume that (*) holds for size n, for some natural number $n \geq |\mathbf{A}_{\vec{p},\mathcal{A}}|$ (induction hypothesis), and show that it also holds for $\mathbf{A}''_{\mathcal{A}''}$ with $|\mathbf{A}''_{\mathcal{A}''}| = n + 1$. So, let $\mathbf{A}''_{\mathcal{A}''}$ be a subset of an answer set of Π s.t. $|\mathbf{A}''_{\mathcal{A}''}| = n + 1$. Further, let $\mathbf{A}'''_{\mathcal{A}'''}$ be a partial assignment obtained from $\mathbf{A}''_{\mathcal{A}''}$ by removing an arbitrary element Xa from $\mathbf{A}''_{\mathcal{A}''}$, $X \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$, s.t. $Xa \notin \mathbf{A}_{\vec{p},\mathcal{A}}$ for any $X \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$. Such an element must exist since $|\mathbf{A}''_{\mathcal{A}''}| > |\mathbf{A}_{\vec{p},\mathcal{A}}|$. Then, it holds that $f_{\&g}(\hat{\mathbf{A}}_{\mathcal{A}}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\hat{\mathbf{A}}'''_{\mathcal{A}'''}, \vec{p}, \vec{c}) = \mathbf{T}$, according to the induction hypothesis. In case the atom a does not have a predicate occurring in $\vec{p}$, proposition (*) holds because the value of $f_{\&g}(\hat{\mathbf{A}}_{\mathcal{A}}, \vec{p}, \vec{c})$ only depends on the extension of predicates $p \in \vec{p}$ in $\hat{\mathbf{A}}_{\mathcal{A}}$.

If a has a predicate occurring in $\vec{p}$, we have that $\mathbf{A}''_{\mathcal{A}''} \succeq \mathbf{A}'''_{\mathcal{A}'''}$ and that $\mathbf{A}''_{\mathcal{A}''}$ augments $\mathbf{A}'''_{\mathcal{A}'''}$ only by an atom with a predicate p occurring in $\vec{p}$. Because p is a monotonic input predicate, we derive that $f_{\&g}(\hat{\mathbf{A}'''}_{\mathcal{A}'''}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\hat{\mathbf{A}}''_{\mathcal{A}''}, \vec{p}, \vec{c}) = \mathbf{T}$ and thus, that proposition (*) holds. □

Many external atoms have only monotonic input (e.g., string concatenation, DL-atoms, and the RDF-atom in the dlvhex-library). Regarding nonmonotonic input, let the set $\mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$ contain all predicates occurring as an input to some external atom in program Π that are not monotonic, i.e.,

$$\mathbf{p}_{\overline{\mathbf{m}}}(\Pi) = \{p \in \mathcal{P} \mid \&g[\vec{p}](\vec{c}) \text{ occurs in } \Pi, p \in \vec{p}, p \text{ is not monotonic}\}.$$

Using the set $\mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$, we can specify domains which are not only safe for evaluating external atoms with monotonic input parameters, but which can also safely be used to evaluate external atoms with input parameters that are not monotonic; i.e. the truth value of an external atom cannot change from $\mathbf{T}$ to $\mathbf{F}$ when the domain is expanded by atoms with predicate in $\mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$.

Definition 5.3 (Input-Safe Domain)**.** *For a* hex*-program* Π*, a (finite) set* $\mathcal{A} \subseteq \mathcal{HB}$ *of ground atoms is an* input-safe domain *of* Π*, if it contains each atom* $p(\mathbf{X})$ *where* $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$ *and* $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$ *for some answer set* $\mathbf{A}$ *of* Π*.*

Next, we establish the relation between input-complete assignments and input-safe domains, which will be exploited subsequently for ensuring that external atoms are not spuriously assigned the truth value $\mathbf{T}$ during lazy-grounding hex-evaluation.

Proposition 5.2. *A partial assignment* $\mathbf{A}_{\mathcal{A}}$ *is input-complete w.r.t. a* hex*-program* Π*, if* $\mathcal{A}$ *is an input-safe domain of* Π*.*

Proof. Given a partial assignment $\mathbf{A}_\mathcal{A}$ and a HEX-program Π, let $\mathbf{A}_\mathcal{A}$ be over an input-safe domain $\mathcal{A}$ of Π. This means that $\mathcal{A}$ contains each $p(\mathbf{X})$ s.t. $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$ for some answer set $\mathbf{A}$ of Π and $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$.

We need to show that $\mathbf{A}_\mathcal{A}$ is input-complete w.r.t. Π, i.e. that for each external atom $\&g[\vec{p}](\vec{c})$ occurring in $grnd(\Pi)$ we have that $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c}) = \mathbf{T}$ only if every answer set $\mathbf{A}'_{\mathcal{HB}}$ of Π s.t. $\mathbf{A}'_{\mathcal{HB}} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$ fulfills $f_{\&g}(\mathbf{A}'_{\mathcal{HB}}, \vec{p}, \vec{c}) = \mathbf{T}$. Let $\&g[\vec{p}](\vec{c})$ be an arbitrary external atom occurring in $grnd(\Pi)$.

As in the proof for Proposition 5.1, we prove a stronger result, namely, that for every subset $\mathbf{A}'_{\mathcal{A}'}$ of an answer set of Π with $\mathbf{A}'_{\mathcal{A}'} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$ the following holds:
(*) $f_{\&g}(\hat{\mathbf{A}}'_{\mathcal{A}'}, \vec{p}, \vec{c}) = \mathbf{T}$ if $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c}) = \mathbf{T}$.

Let $\mathbf{A}'_{\mathcal{A}'}$ be an arbitrary subset of an answer set of Π s.t. $\mathbf{A}'_{\mathcal{A}'} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$. Again, we prove this by induction on the cardinality $n = |\mathbf{A}'_{\mathcal{A}'}|$ of $\mathbf{A}'_{\mathcal{A}'}$, and only need to consider $n \geq |\mathbf{A}_{\vec{p},\mathcal{A}}|$.

The base case is $n = |\mathbf{A}'_{\mathcal{A}'}| = |\mathbf{A}_{\vec{p},\mathcal{A}}|$. In this case, (*) follows directly due to assignment monotonicity, and because the value of $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c})$ only depends on the extension of predicates $p \in \vec{p}$ in $\hat{\mathbf{A}}_\mathcal{A}$ (cf. Section 2).

For the induction hypothesis, we assume that statement (*) holds for size n, and prove the same for $\mathbf{A}''_{\mathcal{A}''}$ with $|\mathbf{A}''_{\mathcal{A}''}| = n+1$. Let $\mathbf{A}''_{\mathcal{A}''}$ be a subset of an answer set of Π s.t. $|\mathbf{A}''_{\mathcal{A}''}| = n+1$ and $\mathbf{A}''_{\mathcal{A}''} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$. As before, we obtain $\mathbf{A}'''_{\mathcal{A}'''}$ by removing an arbitrary element Xa from $\mathbf{A}''_{\mathcal{A}''}$, $X \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$, s.t. $Xa \notin \mathbf{A}_{\vec{p},\mathcal{A}}$ for any $X \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$. Then, it holds that $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\hat{\mathbf{A}}'''_{\mathcal{A}'''}, \vec{p}, \vec{c}) = \mathbf{T}$, according to the induction hypothesis. In case the atom a does not have a predicate occurring in $\vec{p}$, statement (*) holds because the value of $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c})$ only depends on the extension of predicates $p \in \vec{p}$ in $\hat{\mathbf{A}}_\mathcal{A}$.

If a has a predicate $p \in \vec{p}$ and p is a monotonic input predicate, the induction step of the proof for Proposition 5.1 applies. So, consider the remaining case, namely, that a has a predicate $p \in \vec{p}$ and p is not a monotonic input predicate, i.e. $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$.

It cannot be the case that $\mathbf{T}a \in \mathbf{A}''_{\mathcal{A}''}$ because then $\mathbf{U}a \in \mathbf{A}_{\vec{p},\mathcal{A}}$ or $\mathbf{T}a \in \mathbf{A}_{\vec{p},\mathcal{A}}$ would also hold, while we have chosen a s.t. $Xa \notin \mathbf{A}_{\vec{p},\mathcal{A}}$ for any $X \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$. The latter is true as $\mathbf{A}''_{\mathcal{A}''} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$ holds, $\mathbf{A}''_{\mathcal{A}''}$ is a subset of an answer set of Π, and $\mathcal{A}$ contains each $p(\mathbf{X})$ s.t. $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$ for an answer set $\mathbf{A}$ of Π for $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$.

It cannot be the case that $\mathbf{U}a \in \mathbf{A}''_{\mathcal{A}''}$ because $\mathbf{A}''_{\mathcal{A}''}$ is a subset of an answer set, which is a complete assignment. So, it is only left to consider the case that $\mathbf{F}a \in \mathbf{A}''_{\mathcal{A}''}$. But then, it holds that $\mathbf{F}a \in \hat{\mathbf{A}}'''_{\mathcal{A}'''}$ according to the definition of the completion $\hat{\mathbf{A}}'''_{\mathcal{A}'''}$ of $\mathbf{A}'''_{\mathcal{A}'''}$ w.r.t. $\mathcal{HB}$, and $\hat{\mathbf{A}}''_{\mathcal{A}''} = \hat{\mathbf{A}}'''_{\mathcal{A}'''}$. By the induction hypothesis, $f_{\&g}(\hat{\mathbf{A}}'''_{\mathcal{A}'''}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c}) = \mathbf{T}$ and consequently, $f_{\&g}(\hat{\mathbf{A}}''_{\mathcal{A}''}, \vec{p}, \vec{c}) = \mathbf{T}$ implies $f_{\&g}(\hat{\mathbf{A}}_\mathcal{A}, \vec{p}, \vec{c}) = \mathbf{T}$. Thus, we obtain that statement (*) holds. □

5.1.2 Relevant Grounding

In our novel HEX-algorithm introduced in Section 5.2, we utilize input-safe domains to ensure correctness in the context of incremental domain expansion by lazy grounding. Notably, Definitions 5.3 and 5.2 are in semantic terms, relying on answer sets of a given

program. Accordingly, these semantic concepts are not suited for generating input-safe domains in our algorithm. Thus, we now capture the respective notions syntactically by the *relevant grounding* $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset)$ for an input predicate p of some external atom that is not monotonic. To this end, we compute a partial grounding of a given HEX-program by only considering the subset of (non-ground) rules that is relevant for obtaining all ground instances of p.

Definition 5.4 (Relevant Rules)**.** *Given a predicate name p, a HEX-program Π, and a set S of predicate names, the* relevant rules of Π w.r.t. p and S *are*

$$rel_\Pi(p,S) = \bigcup_{r\in\Pi_p} \{r\} \cup \{r' \in rel_\Pi(p',S') \mid p' \in \mathcal{P}B^+(r), p' \notin S'\},$$

where $S' = S \cup \{p\}$, Π_p contains all rules of Π where p occurs in the head, and $\mathcal{P}B^+(r)$ consists of all predicate names that occur in $B^+(r)$ either as ordinary atom predicate or as an input to an external atom. Furthermore, the relevant rules of Π w.r.t. p *are defined by $rel_\Pi(p,\emptyset)$.*

In order to obtain all instances of a predicate that are possibly true in some answer set of a program, we employ the following monotone grounding operator G_Π by Eiter, Fink, Krennwallner, and Redl (2016):

$$G_\Pi(\Pi') = \textstyle\bigcup_{r\in\Pi}\{r\theta \mid \exists I \subseteq A(\Pi'), \hat{I} \models B^+(r\theta)\},$$

where $r\theta$ is the ground instance of r under variable substitution $\theta : \mathcal{V} \to \mathcal{C}$, $\hat{I} = \{\mathbf{T}a \mid a \in I\} \cup \{\mathbf{F}a \mid a \in \mathcal{HB} \setminus I\}$, and $A(\Pi')$ is the set of all ordinary ground atoms occurring in Π'. The least fixpoint of $G^{\infty}_\Pi(\emptyset)$ contains all atoms that are true in some answer set of Π.

Example 5.3 (cont'd). We have $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset) = \{d(c) \leftarrow \text{not } a;\ p(c) \leftarrow d(c)\}$, and a partial assignment $\mathbf{A}_\mathcal{A}$ is input-complete w.r.t. Π, if $\mathcal{A} \supseteq \{p(c)\}$. Note that in general, $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset)$ would not contain further rules on which p does not depend. △

The rules in $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset)$ contain all ground instances over p that occur in the grounding $grnd(\Pi)$ of Π. Since all atoms that occur in some answer set of Π also occur in $grnd(\Pi)$, the relevant grounding thus indicates all such atoms, as stated by the following proposition.

Proposition 5.3. *Let Π be a HEX-program. If $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$, for some answer set $\mathbf{A}$ of Π, then $p(\mathbf{X})$ occurs in $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset)$.*

Proof. Given a HEX-program Π, we assume that $\mathbf{A}$ is an answer set of Π s.t. $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$, for some ground atom $p(\mathbf{X})$, and prove that (*) $p(\mathbf{X})$ occurs in $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset)$.

From (Eiter, Fink, Krennwallner, & Redl, 2016), we know that $G^{\infty}_\Pi(\emptyset)$ has the same answer sets as Π. Consequently, it can only hold that $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$ if $p(\mathbf{X})$ occurs in $G^{\infty}_\Pi(\emptyset)$. We prove (*) by showing that $p(\mathbf{X})$ occurs in $G^{\infty}_{rel_\Pi(p,\emptyset)}(\emptyset)$ whenever $p(\mathbf{X})$ occurs in $G^{\infty}_\Pi(\emptyset)$.

For a HEX-program Π, we say that a predicate p positively depends on a predicate q, written $p \to_+ q$, if there is a rule $r \in \Pi$ s.t. p occurs in the head of r and q occurs in $B^+(r)$ as ordinary atom predicate or as an input to an external atom. The associated positive dependency graph $D^+(\Pi)$ has as nodes all predicates occurring in Π and contains all directed edges corresponding to the dependency relation $\to_+$ (and nothing else).[1] With $\to_+^*$ we denote the relation corresponding to the transitive closure of the relation $\to_+$, i.e. $a \to_+^* b$ holds iff b is reachable from a in $D^+(\Pi)$.

Given a predicate p, it is easy to see that if $p(\mathbf{X})$ occurs in $G_\Pi^\infty(\emptyset)$, and we obtain Π' by only removing rules that do not contain a predicate q in the head where $p \to_+^* q$, then it must still hold that $p(\mathbf{X})$ occurs in $G_{\Pi'}^\infty(\emptyset)$ since only the positive rule bodies are considered when deriving new rules by means of the G_Π-operator and because the value of a ground external atom only depends on the extension of predicates in its input.

Now, it is left to show that given a predicate p, $rel_\Pi(p, \emptyset)$ contains all rules from Π that have a predicate q in the head s.t. $p \to_+^* q$. We show that $rel_\Pi(p, \emptyset)$ contains all such rules by induction on the minimum distance n between p and q in $D^+(\Pi)$.

For the base case, let $n = 0$. The only predicate that has distance 0 from p in $D^+(\Pi)$ is p itself, and it follows directly from Definition 5.4 that $rel_\Pi(p, \emptyset)$ contains all rules from Π that have p in the head.

Next, we assume that $rel_\Pi(p, \emptyset)$ contains all rules from Π that have a predicate q in the head where $p \to_+^* q$ s.t. q has distance smaller or equal to n from p (induction hypothesis). We prove that the same holds for $n + 1$. Let q' be a predicate s.t. $p \to_+^* q'$ and q' has distance $n + 1$ from p. We need to prove that $rel_\Pi(p, \emptyset)$ contains all rules where q' occurs in the head. Because we have that $p \to_+^* q'$, we know that there is a rule $r \in \Pi$ where q' occurs in $B^+(r)$ as ordinary atom predicate or as an input to an external atom. Let p' be the predicate of the atom in the head of r. It holds that p' has distance n from p. Thus, due to the induction hypothesis, we know that $rel_\Pi(p, \emptyset)$ contains all rules from Π that have p' in the head. However, for every rule $r' \in rel_\Pi(p, \emptyset)$ also all rules with predicate q'' in the head are contained in $rel_\Pi(p, \emptyset)$, where q'' occurs in $B^+(r)$ as ordinary atom predicate or as an input to an external atom, according to Definition 5.4. Thus, $rel_\Pi(p, \emptyset)$ also contains all rules from Π that have predicate q' in the head. Consequently, $rel_\Pi(p, \emptyset)$ contains all rules from Π that have a predicate q in the head s.t. $p \to_+^* q$. We conclude that $p(\mathbf{X})$ occurs in $G_{rel_\Pi(p,\emptyset)}^\infty(\emptyset)$. □

Considering the relevant grounding of those predicates p that occur in Π as a not monotonic input to some external atom, i.e., considering the relevant grounding of all $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$, we can obtain an input-safe domain of Π and thus obtain input-complete partial assignments. More formally:

Theorem 5.1. *A partial assignment $\mathbf{A}_\mathcal{A}$ is input-complete w.r.t. a HEX-program Π if for all $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$ it holds that $p(\vec{X}) \in \mathcal{A}$ whenever $p(\vec{X})$ occurs in $G_{rel_\Pi(p,\emptyset)}^\infty(\emptyset)$.*

[1] Note that this dependency graph is different from the dependency graph introduced in Section 4.2 as it is defined w.r.t. predicates and only takes the positive rule body into account.

Proof. Let Π be a HEX-program and $\mathbf{A}_\mathcal{A}$ a partial assignment s.t. for all $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$ it holds that $p(\vec{X}) \in \mathcal{A}$ whenever $p(\vec{X})$ occurs in $G^\infty_{rel_\Pi(p,\emptyset)}(\emptyset)$. From Proposition 5.3, we obtain that $\mathcal{A}$ contains each $p(\mathbf{X})$ where $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$ and $\mathbf{T}p(\mathbf{X}) \in \mathbf{A}$ for an answer set $\mathbf{A}$ of Π. Due to Definition 5.3, it follows that $\mathcal{A}$ is an *input-safe domain* of Π. According to Proposition 5.2, we derive that $\mathbf{A}_\mathcal{A}$ is input-complete w.r.t. Π, which concludes the proof. □

5.2 Lazy-Grounding HEX-Evaluation Algorithm

In this section, we present the new evaluation algorithm that interleaves the steps taken by a lazy-grounding solver with the evaluation of external sources, which incrementally introduce new output constants into the program.

Given an input-safe domain $\mathcal{A}$, the algorithm operates on top of a transformation from a HEX-program Π to an ordinary logic program $\alpha(\Pi, \mathcal{A})$, such that an ordinary lazy-grounding solver for ASP can be employed as a host to incrementally ground the rules in $\alpha(\Pi, \mathcal{A})$. Moreover, via a novel interface to external sources, lazy-grounding may import input-output relations over external atoms in form of additional rules.

5.2.1 Program Transformation and External Source Interface

In the program transformation $\alpha(\Pi, \mathcal{A})$, external atoms are replaced by ordinary atoms, and the program is extended by rules that allow to explicitly derive the negative extension of a given input-safe domain of Π.

Definition 5.5 (Program Transformation)**.** *Given a HEX-program* Π *and an input-safe domain* $\mathcal{A}$ *of* Π*, the* ordinary program $\alpha(\Pi, \mathcal{A})$ *results from* Π *by replacing each (non-ground) external atom* $\&g[\vec{p}](\vec{t})$ *with an ordinary (non-ground)* replacement atom $e_{\&g[\vec{p}]}(\vec{t})$*, and by adding for each predicate symbol of some atom in* $\mathcal{A}$ *the rule*

$$\overline{p}(\mathbf{X}) \leftarrow p_d(\mathbf{X}), \text{not } p(\mathbf{X}),$$

and for each $p(\mathbf{X}) \in \mathcal{A}$ *a domain fact* $p_d(\mathbf{X}) \leftarrow$*.*

Without loss of generality, we assume that atoms of form $e_{\&g[\vec{p}]}(\vec{t})$, $\overline{p}(\mathbf{X})$ and $p_d(\mathbf{X})$ do not occur in Π, i.e. they are fresh atoms.

The purpose of the program extension is twofold. On the one hand, it ensures that each atom $p(\mathbf{X})$ in $\mathcal{A}$ is either derived to be true or *explicitly* false (via $\overline{p}(\mathbf{X})$), so that in the end, nonmonotonic external atoms are always evaluated under complete assignments. On the other hand, enabling guessing the values of atoms in $\mathcal{A}$ early during the solving process potentially allows that outputs of nonmonotonic external atoms are derived earlier during search.

Example 5.4 (cont'd). Reconsider Π from Example 5.1. Given $\mathcal{A} = \{p(c)\}$, the first rule is replaced by $\leftarrow e_{\&size[p]}(0)$ in $\alpha(\Pi, \mathcal{A})$, and we add the rules $\{\overline{p}(\mathbf{X}) \leftarrow p_d(\mathbf{X}), \text{not } p(\mathbf{X}).;$ $p_d(c) \leftarrow .\}$. △

For interleaving the solving algorithm with the evaluation of external sources, we first define a means for constructing a partial assignment which is input-complete w.r.t. the given program, from an assignment $\mathbf{A}$ derived by the ASP-solver.

Values of atoms that are true w.r.t. $\mathbf{A}$ are taken directly and false atoms are obtained based on atoms of form $\overline{p}(\vec{X})$, which represent falsity of $p(\vec{X})$ according to Definition 5.5. All other atoms in the domain are considered unassigned. Formally:

Definition 5.6 (External Input Assignment)**.** *Given a partial assignment* $\mathbf{A}$ *and a domain* $\mathcal{A}$*, the corresponding* external input assignment *is the set*

$$\begin{aligned} i(\mathbf{A},\mathcal{A}) =& \{\mathbf{T}p(\vec{X}) \in \mathbf{A}\} \cup \{\mathbf{F}p(\vec{X}) \mid \mathbf{T}\overline{p}(\vec{X}) \in \mathbf{A}\} \cup \\ & \{\mathbf{U}p(\vec{X}) \mid p(\vec{X}) \in \mathcal{A}, \mathbf{T}\overline{p}(\vec{X}) \notin \mathbf{A}, \mathbf{T}p(\vec{X}) \notin \mathbf{A}\}. \end{aligned}$$

Intuitively, the construction of an *external input assignment* from a given solver assignment ensures that atoms which have not been assigned a truth value during solving but which are in the given input-safe domain, are explicitly declared to be unassigned whenever an external source is queried. This is necessary because the external source requires information about all atoms which can potentially become true in the search later on, in order to only yield outputs that remain correct when the grounding is extended.

Note that if there is no atom $p(\vec{X})$ s.t. $\mathbf{T}\overline{p}(\vec{X}) \in \mathbf{A}$ and $\mathbf{T}p(\vec{X}) \in \mathbf{A}$, then $i(\mathbf{A},\mathcal{A})$ is a partial assignment. We assume that the previous holds for all external input assignments used in the following.

Example 5.5. Consider the partial assignment $\mathbf{A} = \{\mathbf{T}a, \mathbf{U}b, \mathbf{F}c, \mathbf{F}d, \mathbf{T}\overline{e}\}$ and domain $\mathcal{A} = \{d, e, f\}$. According to Definition 5.6, the corresponding external input assignment is $i(\mathbf{A},\mathcal{A}) = \{\mathbf{T}a, \mathbf{U}d, \mathbf{F}e, \mathbf{U}f\}$. Observe that $i(\mathbf{A},\mathcal{A})$ only depends on the $\mathbf{T}$-part of $\mathbf{A}$ and that $i(\mathbf{A},\mathcal{A})$ is an assignment. △

The external source interface amounts to a function that yields rules representing the corresponding input-output relations of external atoms. These rules are added to the input program processed by the solver. Accordingly, whenever an output value is obtained based on a solver assignment, a rule is generated that implies the ground replacement atom representing the respective output value relative to the current assignment of the relevant input atoms.

Definition 5.7 (External Evaluation Function)**.** *Given* $\&g[\vec{p}]$*, a partial assignment* $\mathbf{A}$ *and a domain* $\mathcal{A}$*, the* external evaluation function η *yields*

$$\eta(\&g[\vec{p}], i(\mathbf{A},\mathcal{A})) = \{e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{\mathbf{A},\vec{p}} \mid f_{\&g}(i(\mathbf{A},\mathcal{A}), \vec{p}, \vec{c}) = \mathbf{T}\},$$

where $\mathbf{B}_{\mathbf{A},\vec{p}} = \{p'(\vec{X}) \mid \mathbf{T}p'(\vec{X}) \in \mathbf{A}, p' \in \{\overline{p}, p\}, p \in \vec{p}\}$ *is a rule body corresponding to the external atom's input.*

Given a HEX-program Π and an input-safe domain $\mathcal{A}$ of Π, we denote all possible external evaluations by $\eta(\Pi) = \{r \mid \exists \mathbf{A}$ s.t. $r \in \eta(\&g[\vec{p}], i(\mathbf{A},\mathcal{A}))$, $\&g[\vec{p}]$ occurs in $\Pi\}$.

Example 5.6. Consider Π from Example 5.1 again. For input-safe domain $\mathcal{A} = \{p(c)\}$ and $\mathbf{A} = \{\mathbf{T}d(c), \mathbf{F}a\}$, the external input assignment $i(\mathbf{A}, \mathcal{A}) = \{\mathbf{T}d(c), \mathbf{U}p(c)\}$ is input-complete for Π, and we obtain that $\eta(\&size[p], i(\mathbf{A}, \mathcal{A}))=\emptyset$. For the partial assignment $\mathbf{A}'=\{\mathbf{T}d(c), \mathbf{F}a, \mathbf{T}\overline{p}(c)\}$, we obtain the external input assignment $i(\mathbf{A}', \mathcal{A})=\{\mathbf{T}d(c), \mathbf{F}p(c)\}$ and $\eta(\&size[p], i(\mathbf{A}', \mathcal{A})) = \{e_{\&size[p]}(0) \leftarrow \overline{p}(c)\}$. △

5.2.2 HEX-Algorithm Based on Lazy Grounding

Algorithm 5.1 allows us to evaluate a HEX-program using lazy grounding. It is based on the lazy-grounding ASP-solver ALPHA, which incorporates ideas from OMIGA (Dao-Tran et al., 2012; Weinzierl, 2017). The algorithm combines *conflict-driven nogood-learning* (*CDNL*) search (Gebser et al., 2012) with lazy grounding. As mentioned before, CDNL applies techniques from SAT-solving to ASP, by translating a ground program into a set of nogoods, corresponding to clauses in SAT-solving, and running a *DPLL-style* search algorithm. In every iteration of the CDNL search procedure, deterministic consequences are propagated first, and in case some nogood is violated, a conflict nogood is added to the nogood store to avoid running into the same conflict again and backjumping is performed. Whenever no deterministic assignments are possible during CDNL-search, but the solver assignment is still incomplete, an unassigned atom is guessed to be true or false.

Algorithm 5.1 receives as input an ordinary program constructed according to Definition 5.5 from a HEX-program Π and an input-safe domain of Π. In practice, we obtain an input-safe domain based on Definition 5.3 and Proposition 5.3, by computing $G^{\infty}_{rel_{\Pi}(p,\emptyset)}(\emptyset)$ for all $p \in \mathbf{p}_{\overline{\mathbf{m}}}(\Pi)$. Note that requesting an input-safe domain of the input program is not a severe restriction on the class of programs that our approach can handle, as an input-safe domain can be obtained for any HEX-program. After initializing, Algorithm 5.1 explores the search space in one loop, where the first step at each iteration is propagation from the currently known nogoods ∇ and the current assignment $\mathbf{A}$ in Part (a). If some nogood δ is violated, in Part (b), a new nogood is learned from the conflict and backjumping is triggered. If propagation at Part (a) derived new assignments, lazy-grounding of the input program is performed in Part (c).

In Part (d) of Algorithm 5.1, all external sources are queried, employing external evaluation functions and external input assignments. Note that, at this point, it cannot be the case that $\mathbf{T}\overline{p}(\vec{X}) \in \mathbf{A}$ and $\mathbf{T}p(\vec{X}) \in \mathbf{A}$ both hold for any atom $p(\vec{X})$, because atoms of the form $\overline{p}(\vec{X})$ are only defined by the rules added in the program transformation of Definition 5.5 and those rules only fire if $\mathbf{T}p(\vec{X}) \notin \mathbf{A}$. In Part (e), guessing is done, which is different from ordinary CDNL-based guessing: due to lazy grounding, not all atoms may be guessed upon, but only those corresponding to ground instances of *applicable* rules (cf. Section 5.1). Heuristics may be employed for selecting good guesses. Upon reaching Part (f), the iterations of lazy grounding, guessing, and propagating do not yield any more information, i.e., a fixpoint has been reached. In order to complete the assignment (w.r.t. known atoms), all atoms being unassigned in $\mathbf{A}$ are assigned to *false*. In Part (g), the assignment is tested for only containing true or false assignments. This

Algorithm 5.1: Lazy-Grounding HEX-Evaluation

Input: The ordinary program $\alpha(\Pi, \mathcal{A})$ corresponding to a HEX-program Π, given input-safe domain $\mathcal{A}$ of Π
Output: All answer sets $AS(\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi))$ of $\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$

$\mathcal{AS} \leftarrow \emptyset$ `// found answer sets`
$\mathbf{A} \leftarrow \{\mathbf{U}a \mid a \in \mathcal{A}\}$ `// all known atoms unassigned`
$\nabla \leftarrow \emptyset$ `// dynamic nogood storage`
Run lazy grounder (obtain initial nogoods ∇ from facts)
while *search space not exhausted* **do**
(a) $(\mathbf{A}, \nabla) \leftarrow \mathsf{Propagation}(\mathbf{A}, \nabla)$
(b) **if** *some nogood* $\delta \in \nabla$ *violated by* $\mathbf{A}$ **then**
analyze conflict, add learned nogood to ∇, backjump
(c) **else if** $\mathbf{A}$ *changed* **then**
run lazy grounder w.r.t. $\mathbf{A}$ and extend ∇
(d) **else if** *external sources not queried for current* $\mathbf{A}$ **then**
extend ∇ w.r.t. $\eta(\&g[\vec{p}], i(\mathbf{A}, \mathcal{A}))$ for each $\&g[\vec{p}]$ in Π
(e) **else if** *there are guesses left* **then**
select a guess
(f) **else if** *exists* $\mathbf{U}a \in \mathbf{A}$ **then**
replace each $\mathbf{U}a$ by $\mathbf{F}a$ in $\mathbf{A}$
(g) **else if** *all atoms assigned* $\mathbf{T}$ *or* $\mathbf{F}$ *in* $\mathbf{A}$ **then**
$\mathcal{AS} \leftarrow \mathcal{AS} \cup \{\hat{\mathbf{A}}\}$
add enumeration nogood and backtrack
(h) **else**
backtrack
end
end
return $\mathcal{AS}$

is necessary, because the ALPHA solver internally works with *must-be-true* as additional truth value for increased efficiency. For evaluation of external atoms, *must-be-true* is treated as *true*. If the check succeeds, then the current assignment is an answer set of the HEX-program and recorded as such.[2] If the check fails, some *must-be-true* remained and the current assignment is not an answer set, hence backtracking occurs in Part (h).

If an external input-cycle would exist, i.e., an input predicate of an external atom depends on the atom itself, cf. (Eiter, Fink, Krennwallner, et al., 2014), an additional minimality check is required in Part (g), i.e. a variant of the e-minimality check described in Chapter 4 adapted to the setting of lazy grounding, which is outside the scope of this work. Hence, in the following we assume programs Π do not have such cycles.

Algorithm 5.1 returns the answer sets of the program transformation together with all rules encoding possibly relevant input-output relations of external atoms:

Proposition 5.4. *For any HEX-program* Π *and input-safe domain* $\mathcal{A}$ *of* Π*, Algorithm 5.1 yields the answer sets of* $\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$.

[2] In the implementation, false atoms of an answer set $\hat{\mathbf{A}}$ are not stored explicitly.

Proof. See Appendix A.2, page 202. □

Given a HEX-program Π and an input-safe domain $\mathcal{A}$ of Π, if Algorithm 5.1 returns an answer set of $\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$, we obtain an answer set of Π by using for ordinary atoms occurring in Π the respective truth value and by setting all other atoms in $\mathcal{HB}$ to false. Observe that the resulting assignment maps all atoms of the form $e_{\&g[\vec{p}]}(\vec{t})$, $\overline{p}(\mathbf{X})$ or $p_d(\mathbf{X})$ to false as they do not occur in Π. Moreover, each answer set of Π is obtained this way.

Theorem 5.2. *For a HEX-program Π and an input-safe domain $\mathcal{A}$ of Π, the answer sets returned by Algorithm 5.1 correspond exactly to the answer sets of Π; i.e. the set $\mathcal{AS}$ contains one assignment $\mathbf{A}'$ for every answer set $\mathbf{A}$ of Π s.t. $\mathbf{A}$ and $\mathbf{A}'$ coincide regarding ordinary atoms in $\mathcal{G}_\Pi$.*

Proof. See Appendix A.2, page 205. □

To show this result, we rely on the correctness and completeness of ordinary lazy-grounding ASP-solving (cf. Theorem 1 in (Weinzierl, 2017)), which needs to be extended to also take external evaluations into account. As external atoms are evaluated under input-complete assignments only, it is ensured that input-output relations returned by the external evaluation function at any point during search are not contradicted by later external evaluations. Since no cyclic dependencies involving external atoms are allowed, their evaluation only depends on a subprogram that does not contain the respective external atom itself. Because of this, the *Splitting Theorem* from (Eiter, Fink, Ianni, et al., 2016) can be applied for proving correctness of Algorithm 5.1. Completeness intuitively follows from completeness w.r.t. ordinary programs and the fact that the truth values computed for replacement atoms by Algorithm 5.1 coincide with the outputs of the respective oracle functions, given identical assignments to ordinary atoms.

5.3 Empirical Evaluation

In this section, we experimentally evaluate the new algorithm for lazy-grounding HEX-solving introduced in Section 5.2.2.

5.3.1 Experimental Setup

To evaluate the performance of the new HEX-algorithm, we have integrated the ALPHA[3] lazy-grounding solver, which is freely available, with the DLVHEX reasoner (Redl, 2016). The two components communicate via the interface in Section 5.2, where DLVHEX bridges to external sources and handles program decomposition, while ALPHA acts as ordinary ASP-solver. The program transformation in Definition 5.5 allows us to omit the usual guessing program (Eiter, Fink, Ianni, et al., 2016) in the implementation. For comparison, we used DLVHEX with GRINGO and CLASP (Gebser, Kaufmann, et al., 2011) as backends.

[3] https://github.com/alpha-asp/Alpha

The benchmark instances and all results are available at `http://www.kr.tuwien.ac.at/research/projects/inthex/lazyhex`.

Evaluation Platform

The tests were performed on a Linux machine with two 12-core AMD Opteron 6176 SE CPUs and 128 GB RAM. The timeout for each run was 300 seconds and the memory limit 12 GB. We used the *HTCondor* load distribution system[4] to ensure a stable environment that minimizes running time variations between runs on the same problem instance.

Average running times of 10 instances per size (respectively 30 in case of the third benchmark) are reported in seconds for computing all answer sets respectively only the first answer set; timeouts are in parentheses.

Benchmark Configurations

As discussed at the beginning of this chapter and in Section 1.2.2, decomposition techniques have been developed before in order to mitigate grounding issues that arise due to nonmonotonic external atoms (Eiter, Fink, Ianni, et al., 2016). On the other side, program decomposition is often detrimental for solving because it may split guesses from integrity constraints in a program. Since our goal is to overcome this tradeoff between grounding and solving by employing lazy-grounding techniques, we compared our algorithm to traditional HEX-evaluation with and without program decomposition.

We used the following three configurations in our experiments:

- **splitting**: the program is decomposed into independently groundable components, which are processed by an ordinary solver;
- **monolithic**: a grounding of the complete program is generated, and an ordinary solver is run; and
- **alpha**: no program splitting happens and the novel algorithm using the lazy-grounding solver ALPHA as backend is employed.

Benchmark Problems

We considered three different benchmark problems for the experimental evaluation, where the second and the third problem have first been considered by Redl (2017a):

- A problem in the context of *social choice*, where individual preferences need to be aggregated into acyclic preference sets such that a non-empty *choice set* (Duggan, 2007) can be obtained (*Consistent Preferences*).
- A generic problem setting which is applicable whenever some configuration needs to be generated by selecting items from a set such that their combined properties satisfy a given set of constraints (*Generic Configuration*).

[4] `http://research.cs.wisc.edu/htcondor`

- *Failure Diagnosis* using abductive reasoning (Kakas, Kowalski, & Toni, 1992), where the necessary causes of a machine failure need to be determined.

All three problems have in common that pre-grounding of the respective encodings is challenging, either because a nonmonotonic external atom needs to be called with exponentially many inputs under monolithic evaluation while decomposition splits a program constraint from a relevant guess (in case of benchmarks two and three); or due to the import of a large number of constants by an external atom (in the consistent preferences benchmark).

The benchmark problems used in this evaluation have many parameters, and randomly generated instances easily turn out to be either *over-* or *under-constrained* (i.e. constraints either eliminate all answer sets or none). Hence, by choosing the particular parameters used in our experiments, we aimed to avoid both cases, which arguably are of less interest in the view of realistic application scenarios.

5.3.2 Hypotheses

Our hypotheses regarding the comparison of lazy-grounding HEX-solving and previous evaluation algorithms in terms of performance were the following:

(H5.1) Configuration **alpha** performs better than **splitting**, in case many guesses violate constraints and decomposition splits the latter from the guessing part.

(H5.2) Configuration **alpha** performs better than **monolithic**, if generating the respective grounding before solving requires a lot of resources due to nonmonotonic external atoms.

5.3.3 Experiments on Lazy-Grounding HEX-Evaluation

In this section, we discuss each of the experiments performed for evaluating lazy-grounding HEX-evaluation in detail and present the according results.

Consistent Preferences

This benchmark considers a problem where many new constants are imported by an external atom based on a guess, which obstructs intelligent grounding techniques. An important task in *social choice* consists in producing *choice sets* (Duggan, 2007) from sets of preferences, i.e. determining the maximal elements w.r.t. the given preferences. If preferences are cyclic and thus, not consistent, the construction of a non-empty choice set may fail since the corresponding preference relation, e.g. obtained by aggregating the preferences of a group of individuals, may not possess maximal elements.

Accordingly, the purpose of the benchmark encoding considered here is to aggregate preferences of a group of individuals into a consistent preference set, i.e. one that allows the construction of a non-empty choice set. In this regard, a HEX-program selects a subset P' of a pool P of persons p and checks if the union of individual preferences

#	All Answer Sets			First Answer Set		
	splitting	**monolithic**	**alpha**	**splitting**	**monolithic**	**alpha**
4	**0.16** (0)	**0.16** (0)	1.22 (0)	**0.13** (0)	**0.13** (0)	1.13 (0)
6	0.80 (0)	**0.43** (0)	1.68 (0)	0.61 (0)	**0.31** (0)	1.43 (0)
8	7.62 (0)	4.09 (0)	**2.48** (0)	5.95 (0)	3.76 (0)	**1.98** (0)
10	67.52 (0)	88.54 (0)	**4.82** (0)	55.31 (0)	85.11 (0)	**3.43** (0)
12	300.00 (10)	189.79 (6)	**9.15** (0)	295.73 (9)	158.15 (5)	**5.47** (0)
14	300.00 (10)	300.00 (10)	**17.37** (0)	300.00 (10)	300.00 (10)	**9.52** (0)
16	300.00 (10)	300.00 (10)	**27.79** (0)	300.00 (10)	300.00 (10)	**14.93** (0)
18	300.00 (10)	300.00 (10)	**54.00** (0)	300.00 (10)	300.00 (10)	**25.44** (0)
20	300.00 (10)	288.67 (9)	**132.08** (0)	300.00 (10)	288.27 (9)	**50.67** (0)
22	300.00 (10)	300.00 (10)	**225.47** (0)	300.00 (10)	300.00 (10)	**66.91** (0)
24	**300.00** (10)	**300.00** (10)	**300.00** (10)	300.00 (10)	300.00 (10)	**119.20** (0)

Table 5.1: Results for consistent preferences

$$
\begin{aligned}
sel(X) &\leftarrow \text{not } n\_sel(X), person(X).\\
n\_sel(X) &\leftarrow \text{not } sel(X), person(X).\\
preferred(X,Y) &\leftarrow \&prefs[sel](X,Y).\\
preferred(X,Y) &\leftarrow preferred(X,Z), preferred(Z,Y).\\
&\leftarrow preferred(X,X).
\end{aligned}
$$

Figure 5.1: Consistent preferences rules

$pref(p,I) \subseteq I \times I$ over items I is consistent (i.e., acyclic). The answer sets of the rules from Figure 5.1 plus the facts $\{person(p) \mid p \in P\}$ correspond to all $P' \subseteq P$ where this holds. The item set I and the preferences $pref(p,I)$ are not part of the HEX-program, but imported via an external atom $\&prefs[sel](X,Y)$ for the selected persons. Given a partial assignment $\mathbf{A}$ and an output tuple (i,i'), its oracle function $f_{\&prefs}(\mathbf{A}, sel, i, i')$ evaluates to

- true, if some p fulfills $\mathbf{T}sel(p) \in \mathbf{A}$ and $(i,i') \in pref(p,I)$;
- false, if $\mathbf{F}sel(p) \in \mathbf{A}$ holds for all p s.t. $(i,i') \in pref(p,I)$;
- and to unassigned otherwise.

Thus, the input parameter sel is monotonic, but evaluating the external atom under its maximal extension may cause a large amount of constants to be imported into the program.

We ran tests for randomly generated instances with $N \in [4,24]$ persons and $2 \times N$ items, where each individual preference (i,i') uniformly occurs with 5% probability. The results are presented in Table 5.1.

Generic Configuration

We consider configuration problems such as the assembly of a committee or a server cluster (cf. the example at the beginning of this chapter) (Redl, 2017a). At this, the properties of a particular configuration usually depend on the properties of its parts,

#	All Answer Sets			First Answer Set		
	splitting	**monolithic**	**alpha**	**splitting**	**monolithic**	**alpha**
10	2.06 (0)	**0.40** (0)	1.62 (0)	1.05 (0)	**0.31** (0)	1.38 (0)
12	8.71 (0)	**1.05** (0)	1.87 (0)	6.67 (0)	**0.88** (0)	1.44 (0)
14	39.29 (0)	**4.04** (0)	4.56 (0)	22.20 (0)	3.17 (0)	**1.83** (0)
16	200.54 (0)	14.83 (0)	**3.91** (0)	126.96 (0)	13.89 (0)	**2.87** (0)
18	300.00 (10)	57.20 (0)	**7.29** (0)	249.94 (8)	55.49 (0)	**3.83** (0)
20	300.00 (10)	300.00 (10)	**128.01** (4)	233.52 (7)	300.00 (10)	**7.42** (0)
22	300.00 (10)	300.00 (10)	**133.87** (4)	190.75 (6)	300.00 (10)	**5.60** (0)
24	300.00 (10)	300.00 (10)	**214.84** (7)	257.36 (8)	300.00 (10)	**37.51** (1)
26	**300.00** (10)	**300.00** (10)	**300.00** (10)	212.42 (7)	300.00 (10)	**37.96** (1)
28	300.00 (10)	300.00 (10)	**243.17** (8)	109.73 (3)	300.00 (10)	**6.67** (0)
30	300.00 (10)	300.00 (10)	**272.53** (9)	240.28 (8)	300.00 (10)	**38.66** (1)

Table 5.2: Results for generic configuration

as well as their interplay; i.e. the dependency may be of nonmonotonic nature in the case that adding some part eliminates a property of the configuration. Using ASP for configuration has a long tradition (e.g. Soininen, Niemelä, Tiihonen, and Sulonen (2001) considered product configuration and more recently Gebser, Ryabokon, and Schenner (2015) the railway domain). Here, we address a generic formalization[5] that is likely to occur in real-world scenarios as those mentioned.

A *configuration* is a subset C' of a set C of components, which has an associated set $m(C') \subseteq P$ of properties from a set P. An *admissible* C' must fulfill a set R of requirements (e.g. costumer demands) of the form $(R^+, R^-) \in 2^P \times 2^P$, which means that $R^+ \not\subseteq m(C')$ or $R^- \cap m(C') \neq \emptyset$ holds. For example, w.r.t. the concrete problem of assembling a committee of employees as discussed by Redl (2017a), C is a set of employees, P are properties such as "has technical expertise" or "has financial authority" and a requirement could be that the committee should be able to decide in technical as well as financial affairs, but should not have more than five members.

In the HEX-program, we guess a configuration $C' \subseteq C$ in a predicate $config$ and compute its properties with an external atom $\&prop[config](P)$. As $config$ is a non-monotonic input parameter, traditional grounding must evaluate the oracle function $f_{\&prop}$ for all possible inputs. For a partial configuration C' and property p, $f_{\&prop}$ is

- true, if $p \in m(C'')$ for every $C'' \supseteq C'$;
- false, if $p \notin m(C'')$ for every $C'' \supseteq C'$; and
- unassigned otherwise.

We tested random instances with $N \in [10, 30]$ components, $N/5{+}1$ properties and up to $2 \times N$ positive constraints as requirements, where a property occurs in a requirement with probability 30% and depends on a component with probability 10%; each requirement was added with probability 50 %. The results are shown Table 5.2.

[5] We exploit the benchmark implementation from `https://github.com/hexhex/core/tree/master/benchmarks/genericmapping`.

#	All Answer Sets			First Answer Set		
	splitting	**monolithic**	**alpha**	**splitting**	**monolithic**	**alpha**
5	**0.18** (0)	0.38 (0)	1.39 (0)	**0.14** (0)	0.37 (0)	1.13 (0)
10	**2.64** (0)	6.33 (0)	9.30 (0)	**1.25** (0)	5.89 (0)	1.61 (0)
15	**54.54** (1)	224.03 (2)	56.14 (3)	36.38 (0)	218.92 (1)	**2.10** (0)
20	273.95 (23)	300.00 (30)	**96.49** (9)	262.49 (21)	300.00 (30)	**3.40** (0)
25	300.00 (30)	300.00 (30)	**111.42** (11)	300.00 (30)	300.00 (30)	**8.65** (0)
30	300.00 (30)	300.00 (30)	**102.14** (10)	300.00 (30)	300.00 (30)	**15.02** (0)
35	300.00 (30)	300.00 (30)	**83.31** (8)	300.00 (30)	300.00 (30)	**23.92** (0)
40	300.00 (30)	300.00 (30)	**55.88** (5)	300.00 (30)	300.00 (30)	**27.74** (0)
45	300.00 (30)	300.00 (30)	**88.35** (8)	300.00 (30)	300.00 (30)	**63.07** (2)
50	300.00 (30)	300.00 (30)	**81.79** (7)	300.00 (30)	300.00 (30)	**80.34** (6)

Table 5.3: Results for failure diagnosis

Failure Diagnosis

Another classical use-case of ASP is *abduction-based diagnosis* (Kakas et al., 1992), i.e. the problem of finding possible explanations for the observed effects of a system. Suppose possible causes of a machine failure should be given from certain (Boolean) measurement values that are only partially available. The task is to compute, respecting the open measurement values, all necessary causes that entail the measurement values. In that, information about combinations of failure causes that can be excluded may be available.

According to Redl (2017a), a further use case of this problem setting is medical diagnosis based on reported symptoms and lab results. In this context, certain lab tests may be inconclusive such that measurements are partial, but the tests still have a fixed set of possible outcomes. Moreover, certain diagnoses may be excluded due to additional information provided by a patient. By computing sets of measurements that would imply specific diagnoses, the open lab tests could be performed in a goal-oriented manner.

The problem can be modeled[6] using sets M and M' of known respectively unknown measurements, a set H of possible causes, a logic program P relating measurements and possible causes, and a set $\mathcal{C}$ of constraints that exclude specific combinations of causes. We want to compute the intersection $\mathcal{D}$ of all possible diagnoses $D \subseteq H$ w.r.t. measurement values $\mathcal{M} = M \cup M''$ with $M'' \subseteq M'$, s.t. $\mathcal{D} \not\subseteq C$ for all $C \in \mathcal{C}$. For this, we guess $M'' \subseteq M'$ in the HEX-program and employ a nonmonotonic external atom $\&diagnosis[P, \mathcal{M}](\mathcal{D})$ to obtain the necessary failure causes.

In the tests, we used random instances with $N \in [5, 50]$ measurement values, each available at 20 % (e.g. due to unfinished measurements), and up to $2 \times N$ constraints to exclude combinations of causes, where each occurs in a constraint with probability 30 %. The results are shown in Table 5.3.

5.3.4 Discussion of Results

In all three benchmarks, lazy grounding (setting **alpha**) exhibits a significant advantage in running time over **splitting** and **monolithic**. This matches our hypotheses (H5.1) and

[6] We adopt the implementation from `https://github.com/hexhex/core/tree/master/benchmarks/diagnosis`.

(H5.2) as under **monolithic**, the external atom must be grounded for exponentially many input combinations in the last two benchmarks, and under **splitting**, the search space cannot be pruned effectively due to the separation of guesses and constraints. We observe that **splitting** outperforms **monolithic** for failure diagnosis because computing the diagnoses is resource-intense and must be executed for every input during the grounding step. Here, this outweighs the costs related to less search space pruning in configuration **splitting**. Note that in general, **alpha** finds the first answer set much faster than the other configurations, and notably, was very fast when no answer set exists. However, in computing all answer sets it often timed out when instances have a large number of solutions. Hence, with increasing instance size, the number of instances with many solutions has a stronger impact on the average running times for **alpha**. Methodologically, this suggests to restrict the solution space of a problem by adding further constraints when using lazy grounding.

Somewhat surprising, **alpha** outperformed **monolithic** for consistent preferences, despite feasible grounding for the instance sizes. Our analysis explains this by the large number of guesses usually added for evaluation of external atoms in HEX during grounding. Hence, considerably more time is required for solving. In contrast, no additional guesses must be introduced in our program transformation (cf. Definition 5.5) as here the external atom only has monotonic input parameters and new constants can be imported on-the-fly.

5.4 Related Work

Our work builds on partial evaluation of external atoms (Eiter, Kaminski, et al., 2016) as discussed in Chapter 3, and on the recently developed ALPHA solver (Weinzierl, 2017). It is the first time that lazy grounding has been considered for the HEX-framework. We are not aware of similar approaches for related systems, such as CLINGO (Gebser et al., 2016), which however, supports no value invention based on the respective answer set as the HEX-formalism. Lazy-grounding ASP-solvers like ASPERIX (Lefèvre & Nicolas, 2009b), GASP (Palù et al., 2009), and OMIGA (Dao-Tran et al., 2012) could in theory be employed, but likely result in worse performance, as they are not based on CDNL-search.

Since grounding is a central bottleneck of ASP-solving, several other strategies for tackling grounding issues have been developed. Techniques for *incremental grounding* have been developed based on *module theory* (Oikarinen & Janhunen, 2006), and have been deployed in the ASP-systems *iclingo* (Gebser et al., 2008) and *oclingo* (Gebser, Grote, et al., 2011). Both systems are based on CLASP and GRINGO (Gebser, Kaufmann, et al., 2011). At this, *iclingo* is the first system that incrementally solves and grounds extensions to a problem while avoiding to re-process the whole problem; and *oclingo* enriches *iclingo* with online capabilities for reactive solving and grounding. The advantage of these systems with respect to grounding has been shown, e.g., in the area of planning and other domains where a parameter determines the maximum size of a solution. For instance, in the case of planning, a set of time steps can incrementally be extended in case no solution can be found for a given time frame.

Multi-shot ASP-solving (Gebser et al., 2019) is a recent paradigm for handling problem specifications that evolve over time, and it has been implemented in the CLINGO system (Gebser et al., 2016). The approach is very flexible and also incorporates incremental grounding; at this, it supersedes the previous formalisms realized by *oclingo* and *iclingo*. To this end, CLINGO 5 provides a rich API for controlling the grounding and the solving process using scripting languages. However, leveraging these broad capabilities requires quite some experience on the user side, while lazy grounding can be applied directly for evaluating ordinary ASP- and HEX-programs. At the same time, sophisticated solving and grounding techniques used in the background remain hidden from the user in our approach.

Furthermore, it is important to note that often grounding an encoding on a large domain can be avoided by outsourcing grounding-intense subtasks. For instance, the theory interface of CLINGO 5 or external atoms of HEX can be utilized for this purpose. This method is also at the core of constraint-ASP systems such as CLINGCON (Ostrowski & Schaub, 2012), which avoid the import of large constraint domains. We employ the strategy for our HEX-application presented in Chapter 6, where we limit the domain of an encoding for logic-based machine learning by outsourcing the background knowledge of a learning task.

5.5 Conclusion and Outlook

We have introduced a new algorithm for HEX-programs that interleaves external evaluation plus value invention with lazy-grounding ASP-solving. It employs a tailored interface between the two components with a program transformation based on input-safe domains and a novel evaluation function that adds rules for input-output relations over external atoms to the program. Monotonic external atoms are directly evaluated on partial groundings, with the benefit that no additional guesses are needed. Due to the black-box nature of external atoms, computing a restricted grounding w.r.t. inputs that are not monotonic is necessary; however, this usually involves only a small subset of the complete grounding.

The benchmark results of our prototype implementation are promising, and show the potential of the new algorithm based on the ALPHA-solver. In the special setting of HEX-programs, lazy grounding exhibits a significant benefit already for relatively small instances since program splits and guessing for monotonic external atoms can be avoided.

We plan to integrate an advanced e-minimality check, similar to the one described in Chapter 4, into the new lazy-grounding HEX-algorithm, so that cycles over nonmonotonic external atoms can be handled as well. However, previous algorithms for e-minimality checking operate on the ground program, such that they need to be lifted to the setting of lazy grounding. Moreover, an evaluation mixing full grounding and lazy grounding, each for different parts of a given HEX-program, may increase overall performance.

Part II

Applications of HEX-Programs in Machine Learning

CHAPTER 6

Meta-Interpretive Learning

In this chapter, which is mainly based on the papers (Kaminski et al., 2018b, 2018a), we introduce the first of two applications of HEX-programs in the area of machine learning. More precisely, we apply the HEX-formalism to *meta-interpretive learning* (*MIL*) (Muggleton et al., 2015), an approach for relational machine learning, and show that external atoms are essential for realizing MIL in ASP.

MIL has recently attracted a lot of attention in the area of *inductive logic programming* (*ILP*). The formalism learns definite logic programs from positive and negative examples given some background knowledge by instantiating so-called *meta-rules*. The latter can be viewed as templates specifying the shapes of rules that may be used in the induced program. The formalism is very powerful as it enables *predicate invention*, i.e. to use new predicate symbols in the induced program, and it supports learning of recursive programs, while the hypothesis space can be constrained effectively by using meta-rules.

MIL has been implemented in the *Metagol* system (Cropper & Muggleton, 2016b), which is based on a classical *Prolog* meta-interpreter. The system is very efficient by exploiting the query-driven procedure of Prolog to guide the instantiation of meta-rules in a specific order. In contrast (and complementary) to a common *declarative bias* in ILP which constrains the hypothesis space, this constitutes a *procedural bias* that may affect efficiency (or even termination).

While traditionally most ILP-systems are based on Prolog, the advantages of ASP for ILP were recognized and several ASP-based systems have been developed, e.g. (Otero, 2001; Ray, 2009; Law et al., 2014). Some benign features of ASP are its pure declarativity, which allows to modularly restrict the search space by adding rules and constraints to an encoding without risking non-termination, and that enumeration of solutions is easy. Furthermore, the efficiency and optimization techniques of modern ASP-solvers as well as conflict propagation and learning can be exploited. Muggleton et al. (2014) already considered an ASP-version of Metagol, which used only one specific meta-rule and was tailored to inducing grammars. The authors observed that ASP can have an advantage

for MIL over Prolog due to effective pruning, but that it performs worse when the background knowledge is more extensive or only few constraints are present.

Implementing general MIL by ASP comes with its own challenges; and solving MIL-problems efficiently by utilizing a straightforward ASP-encoding turns out to be infeasible in many cases. The first challenge is the large search space as a result of an unguided search due to a lack of procedural bias. Consequently, the search space must be carefully restricted in an encoding in order to avoid many irrelevant instantiations of meta-rules. The second and more severe challenge concerns the *grounding bottleneck* of ASP, which is not present in Prolog, where only relevant terms are taken into account by *unification*. Finally, a third challenge are recursive manipulations of structured objects, such as strings or lists, that are common for defining background knowledge in Metagol and easy to realize in Prolog, but are less supported in ASP.

In this chapter, we address the mentioned challenges for a class of MIL-problems that is widely encountered in practice, by developing different HEX-encodings for solving MIL-problems.

The content of this chapter is structured as follows:

- In Section 6.1, we introduce the problem setting by defining MIL-problems and their solutions.
- In Section 6.2.1, we introduce our novel MIL approach based on HEX-programs for general MIL-problems. In the first encoding, $\Pi(\mathcal{M})$, we restrict the search space by interleaving derivations at the object level and the meta level such that new instantiations of meta-rules can be generated based on pieces of information that are already derived w.r.t. partial hypotheses of rules. Furthermore, we outsource the background knowledge and access it by means of external atoms, which enables the manipulation of complex objects such as strings or lists.
- In Section 6.2.2, we then define the class of *forward-chained* MIL-problems, for which the grounding can be restricted. Informally, in such problems the elements X, Y in the binary head $p(X, Y)$ of a rule must be connected via a path $p_1(X_1, X_2)$, $p_2(X_2, X_3)$, $\ldots$, $p_k(X_k, X_{k+1})$ in the body, where $X = X_1$ and $X_{k+1} = Y$. This allows us to guard the import of new terms from the background knowledge in a second encoding, $\Pi_f(\mathcal{M})$, by using already imported terms in an inductive manner.
- In Section 6.2.3, we additionally develop a top-down variant, $\Pi_f^{td}(\mathcal{M})$, of the $\Pi_f(\mathcal{M})$-encoding, where a query-driven search as performed by Prolog is simulated in order to derive positive examples in a more goal-directed manner.
- In Section 6.3, we develop a technique to abstract from object-level terms in a fourth encoding, $\Pi_{sa}(\mathcal{M})$, by externally computing sequences of background knowledge atoms that derive all positive examples, and by checking non-derivability of negative examples with an external constraint.

- In Section 6.4, we present results of an empirical evaluation based on known benchmark problems; they provide evidence for the potential of using a HEX-based approach for MIL.
- In Section 6.5, we discuss further aspects of our approach, along with its limitations and possible future mitigations thereof.
- In Section 6.6, we discuss related work; and conclude the chapter in Section 6.7.

While our encoding is inspired by the implementation presented in (Muggleton et al., 2014), to the best of our knowledge, a general implementation of MIL using ASP has not been considered in the literature so far, and neither strategies to compensate for the missing procedural bias nor to mitigate grounding issues have been investigated. Despite the use of the HEX-formalism, our results may be applied to other ASP-formalisms and approaches as well.

6.1 Background on Meta-Interpretive Learning

In addition to the sets of predicate symbols $\mathcal{P}$, constant symbols $\mathcal{C}$, and first-order variable symbols $\mathcal{X}$ introduced in Chapter 2, we assume a further set $\mathcal{H}$ of *higher-order variables*, which is disjoint from the sets $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{X}$. A *higher-order atom* a is of the form $p(t_1, ..., t_n)$, where $t_i \in \mathcal{C} \cup \mathcal{X}$ for $1 \leq i \leq n$ and $p \in \mathcal{H}$; its *arity* is n. While we usually denote first-order variables by upper-case letters X, Y and Z (possibly with indices), we will use upper-case letters P, Q, and R to denote higher-order variables from $\mathcal{H}$ in order to distinguish them from elements in $\mathcal{X}$.

The *meta-interpretive learning* (*MIL*) approach by Muggleton et al. (2015) learns definite logic programs from examples by instantiating so-called *meta-rules*. Here, we focus on meta-rules of the form

$$P(X, Y) \leftarrow Q_1(X_1, Y_1), ..., Q_k(X_k, Y_k), R_1(Z_1), ..., R_n(Z_n), \tag{6.1}$$

where P, Q_i, $1 \leq i \leq k$, and R_j, $1 \leq j \leq n$, are higher-order variables, and X,Y,X_i,Y_i, $1 \leq i \leq k$, and Z_j, $1 \leq j \leq n$, are first-order variables s.t. X and Y also occur in the body. That is, we consider meta-rules with binary atoms in the head and with binary and/or unary atoms in the body. Meta-rules with unary head atoms can be simulated by using atoms of the form $p(X, X)$, and we allow meta-rules of arbitrary (finite) length, such that the program class H^2_m is covered (cf. Cropper and Muggleton (2014)). A *meta-substitution* of a meta-rule R is an instantiation of R where all higher-order variables are substituted by predicate symbols.[1] Examples of concrete meta-rules with names as used by Cropper and Muggleton (2016a) are shown in Figure 6.1.

We are now ready to formally introduce the setting of MIL, adapted to our approach.

[1] Even though we do not consider constants in meta-substitutions, they can easily be simulated by using e.g. a dedicated atom $=_X (X)$ in the body, where $=_X$ is defined in the background knowledge and binds X to a specific constant.

Precon:	$P(X,Y) \leftarrow Q(X), R(X,Y)$	Postcon:	$P(X,Y) \leftarrow Q(X,Y), R(Y)$
Chain:	$P(X,Y) \leftarrow Q(X,Z), R(Z,Y)$	Tailrec:	$P(X,Y) \leftarrow Q(X,Z), P(Z,Y)$

Figure 6.1: Examples of Meta-Rules

Definition 6.1 (MIL-Problem)**.** *A* meta-interpretive learning *(*MIL-*)*problem *is a quadruple* $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$*, where*

- B *is a definite program, called* background knowledge*;*
- E^+ *and* E^- *are finite sets of binary ground atoms called* positive *resp.* negative examples*;*
- $\mathcal{R}$ *is a finite set of meta-rules.*

We say that B *is* extensional *if it contains only ground atoms. A* solution *for* $\mathcal{M}$ *is a* hypothesis $\mathcal{S}$ *consisting of a set of meta-substitutions of meta-rules in* $\mathcal{R}$ *s.t.* $B \cup \mathcal{S} \models e^+$ *for each* $e^+ \in E^+$ *and* $B \cup \mathcal{S} \not\models e^-$ *for each* $e^- \in E^-$*.*

In order to obtain solutions that generalize well to new examples, by *Occam's Razor*, simple solutions to MIL-problems are desired; thus Metagol computes a *minimal solution* containing a minimal number of meta-substitutions (i.e. rules).

Example 6.1. Consider the MIL-problem $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$, with $B = \{m(ann, bob), f(john, bob), m(sue, ann), f(tim, ann)\}$, $E^+ = \{a(sue, bob), a(tim, bob), a(john, bob)\}$, $E^- = \{a(bob, tim)\}$, abbreviating $\underline{m}other$, $\underline{f}ather$ and $\underline{a}ncestor$, and meta-rules $\mathcal{R} = \{P(X,Y) \leftarrow Q(X,Y);\ P(X,Y) \leftarrow Q(X,Z), R(Z,Y)\}$. A minimal solution for $\mathcal{M}$ is $\mathcal{S} = \{p1(X,Y) \leftarrow f(X,Y);\ p1(X,Y) \leftarrow m(X,Y);\ a(X,Y) \leftarrow p1(X,Y);\ a(X,Y) \leftarrow p1(X,Z), a(Z,Y)\}$, where $p1$ is an invented predicate intuitively representing the concept *parent*. △

Muggleton et al. (2015) showed that MIL-problems as in Definition 6.1 are decidable if no proper function symbols (i.e., only constants) are used, and $\mathcal{P}$ and $\mathcal{C}$ are finite, but are undecidable in general. Yet, in practice, complex terms such as lists are often used for MIL. Hence, we assume some suitable restriction, e.g. to consider only a finite set of flat lists, s.t. in slight abuse of notation, complex ground terms (e.g., $[a, b, c]$) are technically regarded as constants in $\mathcal{C}$.

6.2 HEX-Encodings for Meta-Interpretive Learning

In this section, we first introduce our main encoding for solving general MIL-problems, where the background knowledge is stored externally and interfaced by means of external atoms. Subsequently, we present a modification of the encoding which reduces the number of constants that need to be considered during grounding in case only a certain type of meta-rules is used, as well as a top-down variant of this encoding.

A major motivation for developing an ASP-based approach to solve MIL-problems is that constraints given by negative examples can be efficiently propagated by an ASP-solver, while Metagol checks them only at the end. This can be shown by simple synthetic examples; e.g. consider the background knowledge of facts $q_i^j(i)$, $q_i^{11}(i)$ and $q_i^j(0)$, for $1 \leq i, j \leq 10$. For the positive examples $p(1)$, ..., $p(10)$ and the negative example $p(0)$, Metagol finds no solution within one hour using the meta-rule $P(X) \leftarrow Q(X)$. In contrast, the problem can be solved by a simple ASP-encoding instantly. The reason is that e.g. $p(1)$ can only be derived by the rule $p(X) \leftarrow q_1^{11}(X)$ given the negative example $p(0)$, and Metagol explores a huge number of rule combinations before this is detected.

While the issue of negative examples can be tackled by using ordinary ASP, we employ here HEX-programs as they enable us to outsource the background knowledge from the encoding. This allows us to conveniently specify intensional background knowledge using, e.g. string or list manipulations, which are usually not available in ASP. Another advantage of outsourcing the background knowledge is that the approach becomes parametric w.r.t. the formalization of the background knowledge, as it is in principle possible to plug in arbitrary (monotonic) external theories (e.g. a description logic ontology). Beyond this flexibility provided by HEX, external atoms are essential to limit the background knowledge that is imported as described in Section 6.2.2, and for realizing our state abstraction technique in Section 6.3.

6.2.1 General HEX-MIL-Encoding

As we consider meta-rules using unary and binary atoms, we introduce external atoms for importing the relevant unary and binary atoms that are entailed by the background knowledge in an encoding.

Definition 6.2 (Background Knowledge Atoms)**.** *For MIL-problem* $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$*, we call the external atom* $\&bkUnary[deduced](X, Y)$ unary background knowledge *(*BK*)-*atom *and the external atom* $\&bkBinary[deduced](X, Y, Z)$ binary background knowledge *(*BK*)*-atom*, where the associated oracle functions, given an assignment* $\mathbf{A}$*, fulfill that* $f_{\&bkUnary}(\mathbf{A}, deduced, X, Y)$=$\mathbf{T}$ *iff* $B \cup \{p(a, b) \mid \mathbf{T}deduced(p, a, b) \in \mathbf{A}\} \models X(Y)$*, resp. that* $f_{\&bkBinary}(\mathbf{A}, deduced, X, Y, Z)$=$\mathbf{T}$ *iff* $B \cup \{p(a, b) \mid \mathbf{T}deduced(p, a, b) \in \mathbf{A}\} \models X(Y, Z)$*.*

The BK-atoms receive as input the extension of the predicate *deduced*, which represents the set of all atoms that can be deduced from the program that results from the meta-substitutions of the current hypothesis. Their output constants represent unary, resp., binary atoms that are entailed by the background knowledge augmented with the atoms described by *deduced*.

In theory, MIL can be encoded by applying the well-known *guess-and-check* methodology, i.e. by generating all combinations of meta-substitutions from the given meta-rules and available predicate symbols, deriving all entailed atoms, and checking compatibility with examples using constraints. However, this results in a huge search space due to the many possible combinations of meta-substitutions, on top of many meta-substitutions

that can be generated by different combinations of predicate symbols. At the same time, a large fraction of meta-substitutions is irrelevant for inducing a hypothesis as the resulting rule bodies can never be satisfied based on atoms that are deduced using other rules from the hypothesis and the background knowledge.

For this reason, we interleave guesses on the meta level and derivations on the object level, i.e. deductions using meta-substitutions already guessed to be part of the hypothesis, and we model a procedural bias ensuring that meta-substitutions can only be added if their body is already satisfied by atoms deducible on the object level. Note that while Metagol's top-down mechanism effects that only meta-substitutions necessary for deriving a goal atom are generated, our basic encoding works bottom-up such that the procedural bias is inverted. Guarding the guesses of meta-substitutions in this way has not been considered by Muggleton et al. (2014); this constitutes the basis for techniques that restrict the size of the grounding discussed later on.

As in the Metagol implementation of MIL (Muggleton et al., 2015), given a MIL-problem $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$, we associate each meta-rule $R \in \mathcal{R}$ with a unique identifier R_{id} and a set of *ordering constraints* $R_{ord} \subseteq \{ord(P,Q) \mid P, Q \in \mathcal{H} \text{ occur in } R\}$; and we assume a predefined total ordering $\succeq_{\mathcal{P}}$ over the predicate symbols in $\mathcal{P}$. The ordering constraints can be utilized to constrain the search space, and are necessary in Metagol in order to ensure termination. A meta-substitution of a meta-rule R with head predicate p instantiated for the higher-order variable P satisfies the ordering constraints R_{ord} in case $p \succeq_{\mathcal{P}} q$ for every binary body predicate q instantiated for a higher-order variable Q s.t. $ord(P,Q) \in R_{ord}$. Here, we apply ordering constraints only to pairs of head and body predicates, but in general this can be extended to arbitrary pairs of predicates in a meta-substitution. Moreover, we assume that a set $\mathcal{SK} \subseteq \mathcal{P}$ of *Skolem predicates* can be used for predicate invention, where no element in $\mathcal{SK}$ occurs in $\mathcal{M}$.

We are now ready to present our main encoding for solving MIL-problems using HEX. In the HEX-encodings presented in this chapter, we make use of *choice atoms* of the form $\{a\}$ in the heads of rules (Calimeri et al., 2013), which syntactically can be replaced by a disjunctive head $a \vee \overline{a}$, where $\overline{a}$ is a fresh atom.

Definition 6.3 (HEX-MIL-Encoding)**.** *Given a MIL-problem $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$ and a finite set of Skolem predicates $\mathcal{S}$, let Sig be the set that contains each $p \in \mathcal{SK}$ and each predicate symbol p that occurs either in $E^+ \cup E^-$ or in a rule head in B. The* HEX-MIL-encoding *for $\mathcal{M}$ is the* HEX*-program $\Pi(\mathcal{M})$ containing*

(1) a fact $sig(p) \leftarrow$ for each $p \in Sig$, and a fact $ord(p,q) \leftarrow$ for all $p, q \in Sig$ s.t. $p \succeq_{\mathcal{P}} q$

(2) the rules $unary(X,Y) \leftarrow \&bkUnary[deduced](X,Y)$ and $deduced(X,Y,Z) \leftarrow \&bkBinary[deduced](X,Y,Z)$

(3) for each meta-rule $R = P(X,Y) \leftarrow Q_1(X_1,Y_1), ..., Q_k(X_k,Y_k), R_1(Z_1), ..., R_n(Z_n) \in \mathcal{R}$ and $\{ord(P,Q_{i_1}), ..., ord(P,Q_{i_m})\} = \{ord(P,Q_i) \in R_{ord} \mid i \in \{1, ..., k\}\}$,

(a) a rule

$$\{meta(R_{id}, X_P, X_{Q_1}, ..., X_{Q_k}, X_{R_1}, ..., X_{R_n})\} \leftarrow sig(X_P), sig(X_{Q_1}), ..., sig(X_{Q_k}), sig(X_{R_1}), ..., sig(X_{R_n}), ord(X_P, X_{Q_{i_1}}), ..., ord(X_P, X_{Q_{i_m}}), deduced(X_{Q_1}, X_1, Y_1), ..., deduced(X_{Q_k}, X_k, Y_k), unary(X_{R_1}, Z_1), ..., unary(X_{R_n}, Z_n)$$

(b) and a rule

$$deduced(X_P, X, Y) \leftarrow meta(R_{id}, X_P, X_{Q_1}, ..., X_{Q_k}, X_{R_1}, ..., X_{R_n}), deduced(X_{Q_1}, X_1, Y_1), ..., deduced(X_{Q_k}, X_k, Y_k), unary(X_{R_1}, Z_1), ..., unary(X_{R_n}, Z_n)$$

(4) a constraint $\leftarrow$ not $deduced(p, a, b)$, *for each* $p(a, b) \in E^+$, *and*
a constraint $\leftarrow deduced(p, a, b)$, *for each* $p(a, b) \in E^-$

In the encoding, the predicate *meta* contains meta-substitutions added to an induced hypothesis, and *deduced* captures all atoms that can be deduced from a guessed hypothesis together with the background knowledge. As we consider examples to be binary atoms and only binary atoms can be derived from meta-substitutions, those binary atoms entailed by the background knowledge are directly derived to be in the extension of *deduced*, while unary atoms can only be derived from the background knowledge such that they do not need to be added to the extension of *deduced* and are imported via the predicate *unary* in item (2).

Item (3) constitutes the core of the encoding, which contains the meta-level guessing part (a) and the object-level deduction part (b). A meta-substitution can be guessed to be part of the hypothesis only if first-order instantiations of its body atoms can already be deduced, i.e. only if it is potentially useful for deriving a positive example. At this, predicate names must be from the signature Sig and the ordering constraints must be satisfied as stated by the facts in item (1). Finally, item (4) adds the constraints imposed by the positive and negative examples.

For a given MIL-problem, solutions constituted by induced logic programs can directly be obtained from the answer sets of the respective HEX-MIL-encoding. The induced logic program represented by the *meta*-atoms in an assignment is extracted as follows:

Definition 6.4 (Induced Program)**.** *Given a set of meta-rules* $\mathcal{R}$*, the* definite logic program induced by *an assignment* $\mathbf{A}$ *consists of all rules obtained from a signed literal of the form* $\mathbf{T}meta(R_{id}, X_P, X_{Q_1}, ..., X_{Q_k}, X_{R_1}, ..., X_{R_n}) \in \mathbf{A}$ *such that the meta-rule* $R = P(X, Y) \leftarrow Q_1(X_1, Y_1), ..., Q_k(X_k, Y_k), R_1(Z_1), ..., R_n(Z_n)$ *is in* $\mathcal{R}$*, by substituting* P *by* X_P*,* Q_i *by* X_{Q_i} *for* $1 \leq i \leq k$*, and* R_j *by* X_{R_j} *for* $1 \leq j \leq n$*.*

In the following, we assume that the set of meta-rules $\mathcal{R}$ is given by the respective MIL-problem at hand.

Every answer set of a HEX-MIL-encoding represents a solution for the respective MIL-problem, and all solutions $\mathcal{S}$ that only contain *productive* rules, i.e. rules such that all atoms in the body of some ground instance are entailed by $B \cup \mathcal{S}$, can be generated in this way.

Theorem 6.1. *Given a MIL-problem $\mathcal{M}$, (i) if $\mathbf{A}$ is an answer set of $\Pi(\mathcal{M})$, the logic program $\mathcal{S}$ induced by $\mathbf{A}$ is a solution for $\mathcal{M}$; and (ii) if $\mathcal{S}$ is a solution for $\mathcal{M}$ s.t. all rules in $\mathcal{S}$ satisfy R_{ord} and are productive, then there is an answer set $\mathbf{A}$ of $\Pi(\mathcal{M})$ s.t. $\mathcal{S}$ is the logic program induced by $\mathbf{A}$.*

Proof. (i) The program $\mathcal{S}$ must be a solution for $\mathcal{M}$ according to Definition 6.1 because the constraints in item (4) of Definition 6.3 ensure that every positive example $e^+ \in E^+$ is derivable by rules generated by the background knowledge imported in item (2), the rules generated by item (3b) and meta-substitutions corresponding to signed literals $\mathbf{T}meta(R_{id}, X_P, X_{Q_1}, ..., X_{Q_k}, X_{R_1}, ..., X_{R_n}) \in \mathbf{A}$ obtained from guesses added by item (3a), and that no negative example $e^- \in E^-$ is derivable. In addition, atoms not imported from the background knowledge by item (2) are not relevant for deriving examples according to Definition 6.2 as they are not entailed by $B \cup \mathcal{S}$. Moreover, the facts generated by item (1) are only used in the positive bodies of guessing rules generated by item (3a) such that they only constrain the guesses for meta-substitutions.

(ii) We know that $\mathcal{S}$ is a solution for $\mathcal{M}$ s.t. all meta-substitutions in $\mathcal{S}$ satisfy the respective ordering constraints and are productive. Let $\mathbf{A}$ be the assignment that assigns $\mathbf{T}$ to all atoms corresponding to facts generated by item (1) of Definition 6.3 w.r.t. $\mathcal{M}$, the atoms $unary(p, a)$ and $deduced(p, a, b)$ for all $p(a)$ and $p(a, b)$, resp., s.t. $B \cup \mathcal{S} \models p(a)$ and $B \cup \mathcal{S} \models p(a, b)$, and the atom $meta(R_{id}, p, q_1, ..., q_k, r_1, ..., r_n)$ for every rule $p(X, Y) \leftarrow q_1(X_1, Y_1), ..., q_k(X_k, Y_k), r_1(Z_1), ..., r_n(Z_n) \in \mathcal{H}$ that is a meta-substitution of the meta-rule R, and $\mathbf{F}$ to all other atoms. It can be shown that $\mathbf{A}$ is an answer set of $\Pi(\mathcal{M})$ s.t. $\mathcal{S}$ is the logic program induced by $\mathbf{A}$. □

6.2.2 Forward-Chained HEX-MIL-Encoding

Although the general HEX-MIL-encoding in Definition 6.3 works well when only a small number of constants is introduced by the BK-atoms, the grounding can quickly become prohibitively large when many constants are generated (e.g. due to list operations). This results from the fact that constants produced by item (2) in Definition 6.3 are also relevant for instantiating the rules defined in items (3a) and (3b), which contain many variables, causing a combinatorial explosion.

Example 6.2. Consider a MIL-problem $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$, with background knowledge $B = \{remove([X|R], R) \leftarrow\}$, and the positive examples $E^+ = \{remove2([a, a, a], [a]), remove2([b, b], [])\}$. Here, the definition of the background knowledge should be read as an abbreviation for a set of facts, e.g. containing $remove([a, a], [a])$, $remove([a], [])$, etc., using the list notation of Prolog. Accordingly, the predicate *remove* drops the first element from a list, and a corresponding hypothesis intuitively needs to remove the first two elements from the list in the first argument of an example to yield the second one.

Now, assume that $\mathcal{C}$ contains lists with letters from the set $\{a, b, c\}$ up to some length n. Then, the background knowledge contains, e.g., $remove([c, c], [c])$, $remove([c, c, c], [c, c])$, etc., up to length n, which are imported via the BK-atoms. However, lists containing the letter c are irrelevant w.r.t. $\mathcal{M}$ because they cannot be obtained from lists appearing in the examples using the operations in the background knowledge. △

Next, we introduce a class of meta-rules that allows us to restrict the number of constants imported from the background knowledge, based on the observation from the previous example.

Definition 6.5 (Forward-Chained Meta-Rule)**.** *A* forward-chained *meta-rule is of the form*

$$P(Z_0, Z_k) \leftarrow Q_1(Z_0, Z_1), \ldots, Q_i(Z_{i-1}, Z_i), \ldots, Q_k(Z_{k-1}, Z_k), R_1(X_1), \ldots, R_l(X_l),$$

where $1 \leq i \leq k$*,* $0 \leq l$*, and* $X_j \in \{Z_0, \ldots, Z_k\}$ *for all* $1 \leq j \leq l$*. A MIL-problem* $\mathcal{M}$ *is* forward-chained *if* $\mathcal{R}$ *only contains forward-chained meta-rules.*

Intuitively, all first-order variables in the body of a forward-chained meta-rule are part of a chain between the first and second argument of the head atom. Viewing binary predicates in the background knowledge as mappings from their first to their second argument, only atoms from an extensional background knowledge are relevant that occur in a chain between the first and the second argument of examples. Hence, atoms from the background knowledge only need to be imported when their first argument occurs in the examples or in a deduction w.r.t. background knowledge that has already been imported. However, when the derivable background knowledge depends on guessed meta-substitutions, additional atoms might be relevant, and thus, we only consider extensional background knowledge in the following.

For restricting the import of background knowledge, we introduce a modification of the external atoms from Definition 6.2, where the output is guarded by an input constant.

Definition 6.6 (Forward-Chained BK-Atoms)**.** *Given a forward-chained MIL-problem* $\mathcal{M}$ *with extensional* B*, we call the external atoms* $\&fcUnary[Y](X)$ *and* $\&fcBinary[Y](X, Z)$ unary *and* binary forward-chained BK-atom*, resp., where, for arbitrary assignment* $\mathbf{A}$*, it holds that* $f_{\&fcUnary}(\mathbf{A}, Y, X) = \mathbf{T}$ *iff* $X(Y) \in B$*, resp.,* $f_{\&fcBinary}(\mathbf{A}, Y, X, Z) = \mathbf{T}$ *iff* $X(Y, Z) \in B$*.*

As we assume the background knowledge to be extensional, the input parameter *deduced* is not needed for forward-chained BK-atoms. Based on the previous definition, we can modify our HEX-MIL-encoding such that only relevant atoms from the background knowledge are imported, where forward-chained BK-atoms receive as input all constants that already occur in a deduction or the examples.

Definition 6.7 (Forward-Chained HEX-MIL-Encoding)**.** *Given a forward-chained MIL-problem* $\mathcal{M}$ *where* B *is extensional, the* forward-chained HEX-MIL-encoding *for* $\mathcal{M}$ *is the* HEX*-program* $\Pi_f(\mathcal{M})$ *containing items (1), (3) and (4) from Definition 6.3, and the rules*

(f1) $unary(X, Y) \leftarrow \&fcUnary[Y](X), s(Y)$

$$
\begin{aligned}
unary(X,Y) &\leftarrow \&fcUnary[Y](X), s(Y). && \text{(B1)}\\
deduced(X,Y,Z) &\leftarrow \&fcBinary[Y](X,Z), s(Y). && \text{(B2)}\\
s(X) &\leftarrow pos\_ex(\_,X,\_). && \text{(B3)}\\
s(X) &\leftarrow neg\_ex(\_,X,\_). && \text{(B4)}\\
s(Y) &\leftarrow deduced(\_,\_,Y). && \text{(B5)}\\
deduced(P,X,Y) &\leftarrow meta(postcon,P,Q,R), deduced(Q,X,Y), unary\_bg(R,Y). && \text{(B6)}\\
deduced(P,X,Y) &\leftarrow meta(chain,P,Q,R), deduced(Q,X,Z), deduced(R,Z,Y). && \text{(B7)}\\
\{meta(chain,P,Q,R)\} &\leftarrow sig(P), sig(Q), sig(R), ord(P,Q), ord(P,R),\\
&\qquad deduced(Q,X,Z), deduced(R,Z,Y). && \text{(B8)}\\
\{meta(postcon,P,Q,R)\} &\leftarrow sig(P), sig(Q), ord(P,Q), deduced(Q,X,Y), unary(R,Y). && \text{(B9)}\\
&\leftarrow pos\_ex(P,X,Y), \text{not } deduced(P,X,Y). && \text{(B10)}\\
&\leftarrow neg\_ex(P,X,Y), deduced(P,X,Y). && \text{(B11)}
\end{aligned}
$$

Figure 6.2: Illustration of the forward-chained HEX-MIL-encoding

(f2) $deduced(X,Y,Z) \leftarrow \&fcBinary[Y](X,Z), s(Y)$

(f3) $s(a) \leftarrow$ *for each* $p(a,\_) \in E^+ \cup E^-$

(f4) $s(Y) \leftarrow deduced(\_,\_,Y)$

The main difference between $\Pi_f(\mathcal{M})$ and $\Pi(\mathcal{M})$ is that the import of background knowledge is guarded by the predicate s in items (f1) and (f2), whose extension contains all constants appearing as first argument of an example, due to item (f3), and all constants that appear in deductions based on the already imported BK, due to item (f4).

Figure 6.2 shows a concrete instance of the forward-chained HEX-MIL-encoding (omitting the facts generated by item (1) of Definition 6.3) for a MIL-problem $\mathcal{M}$ with meta-rules $\mathcal{R} = \{P(X,Y) \leftarrow Q(X,Z), R(Z,Y);\ P(X,Y) \leftarrow Q(X,Y), R(Y)\}$, where positive and negative examples are assumed to be given by ground atoms of the form $pos\_ex(p,a,b)$ and $neg\_ex(p,a,b)$, respectively.

The rules (B1) and (B2) import all terms that can be derived from unary and binary predicates defined by the background knowledge w.r.t. already imported terms in an inductive manner via rule (B5) by utilizing external atoms. At this, the import starts from terms that occur as the first argument of an example according to rules (B3) and (B4), and new terms are added incrementally to the extension of the predicate s. The predicate *meta* contains all meta-substitutions added to an induced hypothesis. The rules (B6) and (B7) define the predicate *deduced*, which captures all atoms that can be deduced from the meta-substitutions in a guessed hypothesis together with the imported background knowledge. In turn, new meta-substitutions are guessed by rules (B8) and (B9), where (B8) generates substitutions of the first (*chain*) meta-rule in $\mathcal{R}$, and (B9) of the second (*postcon*) meta-rule; and the heads of the rules encode that an arbitrary number of instances of the head may be true whenever the rule body is satisfied.

Every answer set of the forward-chained HEX-MIL-encoding still corresponds to a solution of the respective MIL-problem, but not all solutions may be obtained. Nonetheless,

it is ensured that a minimal solution (i.e., with fewest meta-substitutions) is encoded by some answer set if it exists:

Theorem 6.2. *Let $\mathcal{M}$ be a forward-chained MIL-problem with extensional B. Then, (i) for every answer set $\mathbf{A}$ of $\Pi_f(\mathcal{M})$, the logic program $\mathcal{S}$ induced by $\mathbf{A}$ is a solution for $\mathcal{M}$; and (ii) there is an answer set $\mathbf{A}'$ of $\Pi_f(\mathcal{M})$ s.t. the logic program induced by $\mathbf{A}'$ is a minimal solution for $\mathcal{M}$ if one exists.*

Proof. (i) Compared to the general HEX-MIL-encoding of Definition 6.3, only item (2) is changed by the encoding $\Pi_f(\mathcal{M})$ such that the import of background knowledge is guarded by the predicate s. As before, item (4) ensures that every positive example $e^+ \in E^+$ is derivable. It is only left to show that if a negative example $e^- \in E^-$ is entailed by $B \cup \mathcal{S}$, then it can also be derived w.r.t. the background knowledge imported via items (f1) and (f2) of Definition 6.7. Since all meta-rules are assumed to be forward-chained, only meta-substitutions of the form $p(X,Y) \leftarrow p_1(X_1,Y_1),\ldots,p_k(X_k,Y_k),r_1(X_1),\ldots,r_l(X_l)$ are usable for deriving examples in which X is connected to Y by a chain of atoms $p_i(X_i,Y_i)$ in the body, where $Y_i = X_{i+1}$, for $1 \leq i \leq k-1$, $X = X_1$ and $Y = Y_k$. Furthermore, (f2) imports every binary atom in the background knowledge where the first argument already occurs as first argument in an example or as second argument in an atom previously imported from the background knowledge, due to items (f3) and (f4).

Similarly, all unary atoms in the background knowledge are imported by (f1) where the single argument occurs in a binary atom from the background knowledge that has already been imported. Hence, all background knowledge that is relevant for derivations by means of meta-substitutions w.r.t. forward-chained meta-rules is imported, and the second constraint of item (4) is violated in case a negative example is entailed by $B \cup \mathcal{S}$.

(ii) Every minimal solution $\mathcal{S}$ for $\mathcal{M}$ contains only productive rules as defined right before Theorem 6.1 because rules which are not productive are not necessary for deriving a positive example. Since we only consider forward-chained meta-rules, an answer set $\mathbf{A}$ of $\Pi_f(\mathcal{M})$ such that $\mathcal{S}$ is the logic program induced by $\mathbf{A}$ only needs to assign $\mathbf{T}$ to those binary atoms from the background knowledge that occur in a chain that connects the first argument of each positive example $e^+ \in E^+$ to its second argument; because only those atoms are necessary for ensuring that each rule in $\mathcal{S}$ is productive. Now, answer sets of $\Pi_f(\mathcal{M})$ are modulo the guess in (3a) least models that can be constructed bottom-up incrementally in a fixpoint iteration, and that contain the atoms they logically entail. Hence, all atoms in the corresponding chain are incrementally imported from the background knowledge by the rules in items (f2) and (f4). Accordingly, there is an answer set $\mathbf{A}$ of $\Pi_f(\mathcal{M})$ s.t. the induced logic program $\mathcal{S}$ w.r.t. $\mathbf{A}$ is a minimal solution for $\mathcal{M}$.

Note that this suffices for finding minimal solutions in practice as our implementation finds any productive solution that is encoded by $\Pi_f(\mathcal{M})$. □

Since, in practice, we employ iterative deepening search for computing a minimal solution, any minimal solution encoded by an answer set of $\Pi_f(\mathcal{M})$ is guaranteed to be found. Thus, we can obtain minimal solutions while grounding issues are mitigated

by steering the import of background knowledge. An additional search space reduction results from the pruning of the grounding.

6.2.3 Top-Down HEX-MIL-Encoding

The HEX-MIL-encodings introduced so far guess new meta-substitutions based on facts which are already derivable in a *bottom-up* fashion from a partial candidate solution. While this limits the search to those meta-substitutions that can potentially derive new facts (starting with the given background knowledge), the generation of the search space differs from the query-driven generation employed by Metagol, which steers the search towards instantiations of only those meta-rules that are needed for deriving positive examples.

Accordingly, even though conflicts resulting from negative examples can be propagated effectively by using the forward-chained HEX-MIL-encoding, the query-driven procedure of Metagol still has an advantage w.r.t. finding derivations of positive examples as it generates exactly one instantiation of a meta-rule for proving the next subgoal during SLD-resolution. This difference is particularly relevant for MIL-problems where the set of negative examples is empty. Ideally, an approach would combine the effective *top-down* derivation of positive examples of Metagol with the propagation of conflicts resulting from negative examples for early backtracking.

In this section, we describe a modification of the forward-chained HEX-MIL-encoding from the previous section, which simulates a top-down derivation of positive examples with the aim to prune the search space more effectively. We start by informally describing the main ideas behind our top-down HEX-MIL-encoding, and provide its formalization subsequently.

First of all, note that the forward-chained HEX-MIL-encoding in Figure 6.2 generates guesses over all meta-substitutions in rules (B8) and (B9) based on pieces of information that are derived in a bottom-up fashion, but independent from the concrete positive examples that must be derived. This has the disadvantages that (i) missing rules for deriving some subgoal in the derivation of positive examples are only detected late after checking the constraint, and (ii) more meta-substitutions than necessary may be added to a solution. The latter increases the sizes of candidate solutions, and makes the derivability of negative examples more likely. In order to better target the guesses, the variant of the forward-chained HEX-MIL-encoding presented next selects exactly one ground instance of a meta-substitution for each subgoal that needs to be proven in the derivation of positive examples.

Now, a straightforward approach would generate all possible instances of meta-substitutions during grounding and select one of them for each subgoal. However, this would make grounding infeasible because it would not take into account that many meta-substitutions can never be *productive*. Hence, we need to limit the number of ground meta-substitutions produced during grounding. To this end, we first compute all relevant instances that can be obtained from the imported background knowledge and meta-rules in a bottom-up fashion, similar as done by the rules (B6) and (B7) of the previous encoding. The difference is that the introduction of rule instances does not

depend on a guess as in the previous encodings, but all relevant ground rules are generated in a first step. Accordingly, an envelope for the ground instances of meta-substitutions that are used for the top-down derivation of positive examples is produced.

Our modified encoding that simulates backward-chaining as performed by Metagol is constructed for a MIL-problem as follows.

Definition 6.8 (Top-Down HEX-MIL-Encoding)**.** *Given a forward-chained MIL-problem* $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$ *and a finite set* $\mathcal{SK}$ *of Skolem predicates, let* $Sig \supseteq \mathcal{SK}$ *contain each predicate symbol* p *that occurs either in* $E^+ \cup E^-$ *or in a rule head in* B*, and let* η *and* ρ *be the maximum numbers of binary, respectively unary, atoms in any meta-rule in* $\mathcal{R}$*. Furthermore, let* ϵ_i*,* $i \in \mathbb{N}$*, be fresh constants that are used as placeholders. The* top-down HEX-MIL-encoding *for* $\mathcal{M}$ *is the HEX-program* $\Pi_f^{td}(\mathcal{M})$ *containing*

(1) a fact $sig(p) \leftarrow$ *for each* $p \in Sig$*, and a fact* $ord(p, q) \leftarrow$ *for all* $p, q \in Sig$ *s.t.* $p \succeq_{\mathcal{P}} q$

(2) the rules
$unary(X, Y) \leftarrow \&fcUnary[Y](X), s(Y)$,
$deduced(X, Y, Z) \leftarrow \&fcBinary[Y](X, Z), s(Y)$,
$s(a) \leftarrow$ *for each* $p(a, b) \in E^+ \cup E^-$,
$s(Y) \leftarrow deduced(\_, \_, Y)$ *and*
$deduced_a(bg, X, \epsilon_1, ..., \epsilon_\eta, \epsilon_1, ..., \epsilon_\rho, Y, Z, \epsilon_1, ..., \epsilon_{\eta-1}) \leftarrow \&fcBinary[Y](X, Z), s(Y)$

(3) for each $R = P(Z_0, Z_k) \leftarrow Q_1(Z_0, Z_1), ..., Q_k(Z_{k-1}, Z_k), R_1(X_1), ..., R_n(X_n) \in \mathcal{R}$ *and* $\{ord(P, Q_{i_1}), ..., ord(P, Q_{i_m})\} = \{ord(P, Q_i) \in R_{ord} \mid i \in \{1, ..., k\}\}$*, where* $\vec{X_Q} = X_{Q_1}, ..., X_{Q_k}$*,* $\vec{X_R} = X_{R_1}, ..., X_{R_n}$*,* $\vec{Z} = Z_1, ..., Z_{k-1}$*, and* $\_^j$*,* $j \geq 0$*, represents a sequence of* j *anonymous variables* $\_$*,*[2]

(a) a rule

$$deduced_a(R_{id}, X_P, \vec{X_Q}, \epsilon_{k+1}, ..., \epsilon_\eta, \vec{X_R}, \epsilon_{n+1}, ..., \epsilon_\rho, Z_0, Z_k, \vec{Z}, \epsilon_{k+1}, ..., \epsilon_{\eta-1}) \leftarrow$$
$$sig(X_P), sig(X_{Q_1}), ..., sig(X_{Q_k}), sig(X_{R_1}), ..., sig(X_{R_n}), ord(X_P, X_{Q_{i_1}}), ...,$$
$$ord(X_P, X_{Q_{i_m}}), deduced_a(\_, X_{Q_1}, \_^{\eta+\rho}, Z_0, Z_1, \_^{\eta-1}), ...,$$
$$deduced_a(\_, X_{Q_k}, \_^{\eta+\rho}, Z_{k-1}, Z_k, \_^{\eta-1}), unary(X_{R_1}, X_1), ...,$$
$$unary(X_{R_n}, X_n),$$

(b) a rule

$$deduced(X_P, Z_0, Z_k) \leftarrow meta(R_{id}, X_P, X_{Q_1}, ..., X_{Q_k}, X_{R_1}, ..., X_{R_n}),$$
$$deduced(X_{Q_1}, Z_0, Z_1), ..., deduced(X_{Q_k}, Z_{k-1}, Z_k), unary(X_{R_1}, X_1), ...,$$
$$unary(X_{R_n}, X_n),$$

[2] For instance, $\_^3 = \_, \_, \_$.

$$
\begin{aligned}
unary(X,Y) &\leftarrow \&fcUnary[Y](X), s(Y). && \text{(T1)}\\
deduced(X,Y,Z) &\leftarrow \&fcBinary[Y](X,Z), s(Y). && \text{(T2)}\\
s(X) &\leftarrow pos\_ex(\_,X,\_). && \text{(T3)}\\
s(X) &\leftarrow neg\_ex(\_,X,\_). && \text{(T4)}\\
s(Y) &\leftarrow deduced(\_,\_,Y). && \text{(T5)}\\
deduced(P,X,Y) &\leftarrow meta(postc,P,Q,R), deduced(Q,X,Y), unary\_bg(R,Y). && \text{(T6)}\\
deduced(P,X,Y) &\leftarrow meta(chain,P,Q,R), deduced(Q,X,Z), deduced(R,Z,Y). && \text{(T7)}\\
&\leftarrow pos\_ex(P,X,Y), \text{not } deduced(P,X,Y). && \text{(T8)}\\
&\leftarrow neg\_ex(P,X,Y), deduced(P,X,Y). && \text{(T9)}\\
deduced_a(bg,P,n,n,X,Y,n) &\leftarrow \&fcBinary[Y](X,Z), s(Y). && \text{(T10)}\\
deduced_a(postc,P,Q,R,X,Y,n) &\leftarrow ord(P,Q), deduced_a(\_,Q,\_,\_,X,Y,\_), unary\_bg(R,Y). && \text{(T11)}\\
deduced_a(chain,P,Q,R,X,Y,Z) &\leftarrow ord(P,Q), ord(P,R), deduced_a(\_,Q,\_,\_,X,Z,\_), && \text{(T12)}\\
&\qquad deduced_a(\_,R,\_,\_,Z,Y,\_). &&\\
goal(P,X,Y) &\leftarrow pos\_ex(P,X,Y). && \text{(T13)}\\
goal(Q,X,Z) &\leftarrow deduced_u(chain,\_,Q,\_,X,\_,Z). && \text{(T14)}\\
goal(R,Z,Y) &\leftarrow deduced_u(chain,\_,\_,R,\_,Y,Z). && \text{(T15)}\\
goal(Q,X,Y) &\leftarrow deduced_u(postc,\_,Q,\_,X,Y,\_). && \text{(T16)}\\
\{deduced_u(M,P,Q,R,X,Y,Z) : deduced_a(M,P,Q,R,X,Y,Z)\} = 1 &\leftarrow goal(P1,X,Y). && \text{(T17)}\\
meta(M,P,Q,R) &\leftarrow deduced_u(M,P,Q,R,X,Y,Z), M \neq bg. && \text{(T18)}
\end{aligned}
$$

Figure 6.3: Illustration of top-down HEX-MIL-encoding

(c) and the rules

$$
\begin{aligned}
&goal(X_{Q_1},Z_0,Z_1) \leftarrow deduced_u(\_,\vec{X_Q},\_^{\eta+\rho},Z_0,Z_k,\vec{Z},\epsilon_{k+1},...,\epsilon_{\eta-1}),\\
&...,\\
&goal(X_{Q_k},Z_{k-1},Z_k) \leftarrow deduced_u(\_,\vec{X_Q},\_^{\eta+\rho},Z_0,Z_k,\vec{Z},\epsilon_{k+1},...,\epsilon_{\eta-1})
\end{aligned}
$$

(4) a fact $goal(p,a,b) \leftarrow$*, for each* $p(a,b) \in E^+$*,*
a constraint $\leftarrow \text{not } deduced(p,a,b)$*, for each* $p(a,b) \in E^+$*, and*
a constraint $\leftarrow deduced(p,a,b)$*, for each* $p(a,b) \in E^-$

(5) the rules

$$
\begin{aligned}
&\{deduced_u(M,X_P,X_{Q_1},...,X_{Q_\eta},X_{R_1},...,X_{R_\rho},X,Y,Z_1,...,Z_{\eta-1}) :\\
&\quad deduced_a(M,X_P,X_{Q_1},...,X_{Q_\eta},X_{R_1},...,X_{R_\rho},X,Y,Z_1,...,Z_{\eta-1})\} = 1 \leftarrow\\
&\quad goal(X_P,X,Y), \text{ and}\\
&meta(M,X_P,X_{Q_1},...,X_{Q_\eta},X_{R_1},...,X_{R_\rho}) \leftarrow\\
&\quad deduced_u(M,X_P,X_{Q_1},...,X_{Q_\eta},X_{R_1},...,X_{R_\rho},\_^{\eta+1}), M \neq bg
\end{aligned}
$$

Compared to the HEX-MIL-encodings presented in Sections 6.2 and 6.2.2, items (1) and (3b) of Definition 6.8 correspond exactly to the ones in Definition 6.3, where the latter

derives all facts deducible w.r.t. a candidate hypothesis; and the first four rules in item (2) of Definition 6.8 are the same as used in the forward-chained HEX-MIL-encoding from Definition 6.7 to import all relevant atoms from the background knowledge. Moreover, item (4) of Definition 6.8 adds the constraints imposed by positive and negative examples as before, but additionally declares positive examples to be 'goals'. On the other side, the essential difference to the previous encodings consists in the fact that meta-substitutions which are contained in an induced hypothesis are not guessed as by the rules generated by item (3a) of Definition 6.3, but added via the rules in item (5) when some respective ground instance is selected in a recursive derivation of subgoals that are generated by item (3c).

The top-down variant of the HEX-MIL-encoding from Figure 6.2 is shown in Figure 6.3. We now use this instance of the encoding to illustrate concretely how a top-down search is simulated by our top-down HEX-MIL-encoding. First, the rules (T10)-(T12) produce *all* ground instances of meta-subsitutions that can be derived, starting from the background knowledge using all possible meta-substitutions, and store them in the extension of the predicate $deduced_a$. Second, to simulate the top-down search for proving the positive examples, rule (T13) defines positive examples as initial goals by adding *goal*-atoms for all positive examples. Rule (T17) states that for each new subgoal there needs to be exactly one ground instance of a meta-substitution that allows to derive it, where *used* instances are stored in the extension of the predicate $deduced_u$. The rules (T14)-(T16) recursively add new subgoals to the predicate *goal*, based on the rule instances selected by rule (T17). Finally, rule (T18) accumulates all meta-substitutions representing a computed hypothesis in the extension of the predicate *meta*. Negative and positive examples are then checked as before by generating all facts deducible from a candidate hypothesis via the rules (T6) and (T7), and by checking the corresponding constraints (T8) and (T9).

As for the forward-chained HEX-MIL-encoding, we can show that the top-down HEX-MIL-encoding is correct and always yields a minimal solution of the given MIL-problem in case it has a solution:

Theorem 6.3. *Let $\mathcal{M}$ be a forward-chained MIL-problem with extensional B. Then, (i) for every answer set $\mathbf{A}$ of $\Pi_f^{td}(\mathcal{M})$, the logic program induced by $\mathbf{A}$ is a solution for $\mathcal{M}$; and (ii) there is an answer set $\mathbf{A}'$ of $\Pi_f^{td}(\mathcal{M})$ s.t. the logic program induced by $\mathbf{A}'$ is a minimal solution for $\mathcal{M}$ if one exists.*

Proof. (i) Let $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$ be a forward-chained MIL-problem with extensional B, and $\mathbf{A}$ an answer set of $\Pi_f^{td}(\mathcal{M})$. Let $\mathcal{S}$ be the logic program induced by $\mathbf{A}$ according to Definition 6.4. We need to show that $\mathcal{S}$ is a solution for $\mathcal{M}$. According to Definition 6.1, $\mathcal{S}$ is a solution for $\mathcal{M}$ iff $B \wedge \mathcal{S} \not\models e^-$ for each $e^- \in E^-$ and $B \wedge \mathcal{S} \models e^+$ for each $e^+ \in E^+$. By the same reasoning as in the proof for Theorem 6.2, item (3b) of the top-down HEX-MIL-encoding from Definition 6.8 ensures that $\mathbf{T}deduced(p, a, b) \in \mathbf{A}$ iff $B \wedge \mathcal{S} \models p(a, b)$ for every ground atom $p(a, b)$, under the restricted import of background knowledge. Moreover, due to the constraints in item (4), we know that $\mathbf{T}deduced(p, a, b) \in \mathbf{A}$ for all $p(a, b) \in E^+$ and $\mathbf{T}deduced(p, a, b) \notin \mathbf{A}$ for all $p(a, b) \in E^-$. It follows that $B \wedge \mathcal{S} \not\models e^-$ for each $e^- \in E^-$ and $B \wedge \mathcal{S} \models e^+$ for each $e^+ \in E^+$ and thus, that $\mathcal{S}$ is a solution for $\mathcal{M}$.

(ii) Let $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$ be a forward-chained MIL-problem that has a minimal solution $\mathcal{S}$. We show that there is an answer set $\mathbf{A}$ of $\Pi_f^{td}(\mathcal{M})$ s.t. $\mathcal{S}$ is the logic program induced by $\mathbf{A}$.

Since $\mathcal{S}$ is a solution of $\mathcal{M}$, according to Definition 6.1, we have that $B \wedge \mathcal{S} \models e^+$ for each $e^+ \in E^+$, and hence, as $B \wedge \mathcal{S}$ is a definite logic program, there must be a top-down derivation for each $e^+ \in E^+$, which we denote by $td(e^+, B \wedge \mathcal{S})$, that only uses instances of rules in $B \wedge \mathcal{S}$. Moreover, $\mathcal{S}$ only contains rules of which at least one instance occurs in the top-down derivation of some $e^+ \in E^+$ because otherwise, such a rule could be removed while $\mathcal{S}$ would still be a solution, which would contradict that $\mathcal{S}$ is minimal.

Now, the instantiations of meta-substitutions represented by the $deduced_a$-atoms of item (3a) in Definition 6.8 are an envelope for all ground rules that can possibly occur in a top-down derivation of some $e^+ \in E^+$. The previous holds because the rules of item (3a) generate all instances of all possible meta-substitutions w.r.t. $\mathcal{M}$ in a bottom-up fashion, starting from the background knowledge B (cf. the last rule of item (2)). In this context, the atom $s(Y)$ in the body of the last rule of item (2) prevents the import of background knowledge atoms that do not occur in a chain that connects the first argument of a positive example $e^+ \in E^+$ to its second argument; such atoms cannot occur in a top-down derivation due to our restriction to forward-chained meta-rules.

In addition, the first rule of item (5) selects exactly one $deduced_u$-atom corresponding to some $deduced_a$-atom for each subgoal represented by a *goal*-atom. Consequently, for each $e^+ \in E^+$ and any top-down derivation $td(e^+, B \wedge \mathcal{S})$ of e^+, the set of $deduced_u$-atoms matching the rule instances used in $td(e^+, B \wedge \mathcal{S})$ can be generated by recursively adding subgoals to the extension of the predicate *goal* in item (3c) and selecting the corresponding $deduced_u$-atoms. Then, the corresponding meta-substitutions are aggregated in the resulting answer set $\mathbf{A}$ by the second rule of item (5), and we obtain that $\mathcal{S}$ is the logic program induced by $\mathbf{A}$.

Finally, note that the constraints generated by item (4) only eliminate answer sets $\mathbf{A}'$ where $\mathbf{T}deduced(p, a, b) \in \mathbf{A}'$ for some $p(a, b) \in E^-$, or $\mathbf{T}deduced(p, a, b) \notin \mathbf{A}'$ for some $p(a, b) \in E^+$, i.e. answer sets where the induced logic program $\mathcal{S}'$ is not a solution for $\mathcal{M}$ because $B \wedge \mathcal{S}' \models e^-$ for some $e^- \in E^-$, or $B \wedge \mathcal{S}' \not\models e^+$ for some $e^+ \in E^+$. Therefore, S' is not eliminated by the integrity constraints. □

6.3 State Abstraction

Based on the observation that operations represented by binary background knowledge predicates can be applied sequentially when only forward-chained meta-rules are used, we introduce in this section a further technique that eliminates object-level constants from the encoding entirely. While the $\Pi_f(\mathcal{M})$-encoding focuses the import of constants to those obtainable from constants that already occur in deductions, the number of relevant constants can still be large if many binary background knowledge atoms share the first argument; and all of them must be considered during grounding. However, only one background knowledge atom is needed for each element in a chain that derives a positive example $p(X, Y)$ by connecting X and Y. In fact, the $\Pi_f(\mathcal{M})$-encoding solves

two problems at the same time: (1) finding sequences of binary background knowledge predicates that derive positive examples; and (2) inducing a (minimal) program that calls the predicates in the respective sequences, and prevents the derivation of negative examples.

Example 6.3. Consider the MIL-problem $\mathcal{M}$ where B contains the extension of *remove* from Example 6.2, and extensional background knowledge represented by the predicates $switch([X,Y|R],[Y,X|R]) \leftarrow$ and $firstA([a|R]) \leftarrow$. Furthermore, let $E^+ = \{p([c,a,b,a,b],[c])\}$, $E^- = \{p([c,b,a,b,b],[c])\}$, and $\mathcal{R} = \{P(X,Y) \leftarrow Q(X,Z), R(Z,Y); P(X,Y) \leftarrow Q(X,Y), R(Y); P(X,Y) \leftarrow Q(X,Y)\}$. Intuitively, a solution program needs to memorize c and delete the rest; this requires to repeatedly switch the first two elements and remove the first element. For success, the input list must have 'a' at position 2. This is captured by the hypothesis $\mathcal{H} = \{p(X,Y) \leftarrow p1(X,Z), p(Z,Y); p(X,Y) \leftarrow remove(X,Y); p1(X,Y) \leftarrow switch(X,Y), firstA(Y); p1(X,Y) \leftarrow remove(X,Z), switch(Z,Y)\}$, where $p1$ is an invented predicate; this is in fact a minimal solution for $\mathcal{M}$. In addition, any program which enables derivations that alternate between calling *switch* and *remove* and prevents to derive the negative example using $firstA$ as a guard would be a solution. Notably, the search space of Metagol also contains hypotheses that have no alternation between *switch* and *remove* and thus cannot be solutions. △

The previous example illustrates that the derivability of positive examples depends on the sequences by which binary background knowledge predicates are called in the induced program. Here, finding a correct sequence for a given example can be viewed as a *planning problem*, where object-level constants represent *states*, binary background knowledge predicates are viewed as *actions*, and unary background knowledge predicates constitute *fluents*. The *state abstraction* technique described in the sequel exploits the insight that the tasks of (1) solving the planning problem and (2) finding a matching hypothesis can be separated, where the HEX-program encodes task (2), and computations involving states are performed externally. The advantage of task separation and state abstraction increases with the number of actions that are applicable in a state, as usually more actions not occurring in a derivation of a positive example can be ignored; this reduces the search space and the size of the grounding.

We represent possible plans to derive positive examples by sequences of binary background knowledge atoms. To this end, cyclic sequences (or plans) have to be excluded by requiring that constants (states) occur only once because otherwise, we may obtain infinitely many sequences for a positive example:

Definition 6.9 (Derivation Sequences)**.** *Given a forward-chained MIL-problem $\mathcal{M}$ where B is extensional, the function Seq maps each positive example $p(c_1,c_k) \in E^+$ to the set $Seq(p(c_1,c_k))$ containing all sequences $p_1(c_1,c_2),\ldots,p_{k-1}(c_{k-1},c_k)$, where $p_i(c_i,c_{i+1}) \in B$ for all $1 \leq i < k$, and $c_i \neq c_j$ if $i \neq j$.*

Example 6.4 (cont'd). Reconsider $\mathcal{M}$ from Example 6.3. Then $Seq(p([c,a,b,a,b],[c])) =$

$\{seq\}$, $(s = switch, r = remove)$,

$$seq = s([c,a,b,a,b],[a,c,b,a,b]), r([a,c,b,a,b],[c,b,a,b]), s([c,b,a,b],[b,c,a,b]),$$
$$r([b,c,a,b],[c,a,b]), s([c,a,b],[a,c,b]), r([a,c,b],[c,b]), s([c,b],[b,c]), r([b,c],[c]).$$

△

In order to make information about action sequences that derive positive examples and fluents that hold in states available to the HEX-encoding, we next introduce two external atoms that import such information. States are simply represented by integers in the output as their structure is irrelevant for combining sequences into a hypothesis that generalizes the plans.

Definition 6.10 (State Abstraction Atoms)**.** *For a forward-chained MIL-problem $\mathcal{M}$ where B is extensional, let e^+_{id} and seq_{id} be unique identifiers for each positive example $e^+ \in E^+$ and derivation sequence $seq \in \bigcup_{e^+ \in E^+} Seq(e^+)$, respectively. The external atoms $\&saUnary[](X,Y)$ and $\&saBinary[](X,Y,Z)$ are called* unary *and* binary state abstraction *(*sa-*)*atoms*, resp., where, for arbitrary assignment* $\mathbf{A}$*,*

- $f_{\&saUnary}(\mathbf{A},X,Y) = \mathbf{T}$ *iff* $X = r$, $Y = (e^+_{id}, seq_{id}, i)$, *and* $r(c_i) \in B$*; resp.*
- $f_{\&saBinary}(\mathbf{A},X,Y,Z) = \mathbf{T}$ *iff* $X = p_i$, $Y = (e^+_{id}, seq_{id}, i)$, *and* $Z = (e^+_{id}, seq_{id}, i+1)$,

with $e^+ \in E^+$, $seq = p_1(c_1,c_2),\ldots,p_{k-1}(c_{k-1},c_k) \in Seq(e^+)$, *and* $i \in \{1,\ldots,k-1\}$.

For instance, for $\mathcal{M}$ from Example 6.3, $\&saBinary[](switch,(e^+_{id},seq_{id},1),(e^+_{id},seq_{id},2))$ is true, where e^+_{id} is the identifier of the positive example, seq_{id} is the identifier of the sequence shown in Example 6.4, and the integers 1 and 2 represent the states $[c,a,b,a,b]$ and $[a,c,b,a,b]$, respectively, where the second state can be reached from the first state by applying the action *switch*.

In our encoding with state abstractions we also need information about the start and end states of sequences associated with positive examples, as a hypothesis needs to encode a plan for each positive example. This information is accessed via an external atom as well.

Definition 6.11 (Sequence Import)**.** *For a forward-chained MIL-problem $\mathcal{M}$, we define the external atom $\&checkPos[](X_1,X_2,Y,Z)$ which fulfills $f_{\&checkPos}(\mathbf{A},X_1,X_2,Y,Z) = \mathbf{T}$ iff $X_1 = e^+_{id}$, $X_2 = p$, $Y = (e^+_{id}, seq_{id}, 1)$, and $Z = (e^+_{id}, seq_{id}, k+1)$ for some $p(a,b) = e^+ \in E^+$ and $seq = p_1(a,c_2),\ldots,p_k(c_k,b) \in Seq(e^+)$, for arbitrary assignment* $\mathbf{A}$*.*

Finally, it can only be determined w.r.t. the background knowledge whether a candidate hypothesis derives a negative example, s.t. the corresponding check cannot be performed in an encoding without importing relevant atoms from the background knowledge. As our goal is to abstract from explicit states in the background knowledge, we also need to outsource the check for non-derivability of negative examples by means of an external constraint.

Definition 6.12 (External Check w.r.t. Negative Examples)**.** *Given a MIL-problem $\mathcal{M}$, the oracle function $f_{\&failNeg}(\mathbf{A}, meta)$ associated with the external atom $\&failNeg[meta]()$ returns* $\mathbf{T}$ *iff $B \cup \mathcal{S} \models e^-$ for some $e^- \in E^-$, where $\mathcal{S}$ is the logic program induced by $\{\mathbf{T}meta(R_{id}, x_P, x_{Q_1}, ..., x_{Q_k}, x_{R_1}, ..., x_{R_n}) \in \mathbf{A}\}$.*

In the implementation, the external atom $\&failNeg[meta]()$ receives information about meta-substitutions already guessed by the solver to be in the respective hypothesis. It can be evaluated to true as soon as a negative example is derivable w.r.t. its input, as definite logic programs are monotonic; as this may violate a constraint, backtracking in a solver can be triggered.

Example 6.5. Consider MIL-problem $\mathcal{M}$ with $B = \{q(a,b), q(a,c), r(a,b)\}$, $E^+ = \{p(a,b)\}$, $E^- = \{p(a,c)\}$, and $\mathcal{R} = \{R = P(X,Y) \leftarrow Q(X,Y)\}$. For any assignment $\mathbf{A} \supseteq \{\mathbf{T}meta(R_{id}, p, q)\}$, we obtain that $f_{\&failNeg}(\mathbf{A}, meta) = \mathbf{T}$ as the negative example can be derived from $B \cup \{p(X,Y) \leftarrow q(X,Y)\}$; a solver can exploit the information that $p(X,Y) \leftarrow q(X,Y)$ cannot belong to any solution. △

Utilizing the external atoms introduced in this section, we define an encoding which separates the planning from the generalization problem and contains no object-level constants.

Definition 6.13 (State Abstraction HEX-MIL-Encoding)**.** *Given a forward-chained MIL-problem $\mathcal{M}$ where B is extensional, its* state abstraction *(*sa-*)*HEX-MIL-encoding *is the HEX-program $\Pi_{sa}(\mathcal{M})$ that contains all rules in items (1) and (3) of Definition 6.3, where Sig additionally contains e^+_{id} for each $e^+ \in E^+$, and the rules*

(s1) $unary(X,Y) \leftarrow \&saUnary[](X,Y)$

(s2) $deduced(X,Y,Z) \leftarrow \&saBinary[](X,Y,Z)$

(s3) $\leftarrow$ not $pos1(e^+_{id})$*, for each* $e^+ \in E^+$

(s4) $\{pos(X_{id}, X, Y, Z)\} \leftarrow \&checkPos[](X_{id}, X, Y, Z)$

(s5) $pos1(X_{id}) \leftarrow pos(X_{id}, \_, \_, \_)$

(s6) $\leftarrow$ not $deduced(X,Y,Z), pos(\_, X, Y, Z)$

(s7) $\leftarrow \&failNeg[meta]()$

Items (s1) and (s2) in $\Pi_{sa}(\mathcal{M})$ import the fluents for all relevant states and state transitions w.r.t. sequences that derive positive examples, where states are abstracted. The external atom $\&checkPos[](X_1, X_2, Y, Z)$ in item (s4) imports all tuples representing the start and end state of each sequence for each positive example. The choice atom in the head of (s4) enables each tuple representing a sequence to be guessed to be in the extension of the predicate *pos*, which represents all sequences that are modeled by the induced program. While a minimal hypothesis is guaranteed when the guess is over all possible sequences, in practice, we can preselect sequences returned by the atom

&checkPos[](X_1, X_2, Y, Z). Moreover, the guess can be omitted if the planning problem is deterministic, i.e. if for each positive example there is exactly one sequence of binary atoms from the background knowledge that derives its second argument from its first argument. Items (s3) and (s5) ensure that at least one sequence for each positive example is selected such that the corresponding end state can be derived from the start state by the induced program. Finally, (s6) and (s7) state the constraints regarding positive respectively negative examples.

As can be shown, $\Pi_{sa}(\mathcal{M})$ only yields correct solutions, and a minimal one if all sequences that derive positive examples are acyclic. More formally:

Theorem 6.4. *Let $\mathcal{M}$ be a forward-chained MIL-problem with extensional background knowledge B. Then, (i) for every answer set $\mathbf{A}$ of $\Pi_{sa}(\mathcal{M})$, the logic program induced by $\mathbf{A}$ is a solution for $\mathcal{M}$; and (ii) there is an answer set $\mathbf{A}'$ of $\Pi_{sa}(\mathcal{M})$ s.t. the logic program induced by $\mathbf{A}'$ is a minimal solution for $\mathcal{M}$ if one exists and every sequence of binary background knowledge atoms that derives a positive example in E^+ is acyclic.*

Proof (Sketch). This result can be shown similarly as the previous Theorem 6.2. Each unary and binary atom introduced via the items (s1) and (s2) of Definition 6.13, respectively, whose arguments occur in an acyclic sequence of binary atoms from the background knowledge that connects the first argument a of each positive example $p(a, b) \in E^+$ to its second argument b, can be mapped to exactly one unary and binary atom, respectively, that is introduced by items (f1) and (f2) of Definition 6.7. In this regard, the only difference is that object-level constants in (f1) and (f2) are replaced by abstract states of the form (e^+_{id}, seq_{id}, i) according to Definition 6.10 in (s1) and (s2). Furthermore, all acyclic sequences representing a possible chain that connects the first argument of each positive example to its second argument are imported by the external atom in item $(s4)$.

Then, the only essential remaining differences between $\Pi_f(\mathcal{M})$ and $\Pi_{sa}(\mathcal{M})$ consist in the facts that sequences that are modeled by a solution and correspond to derivations of positive examples are guessed in item (s4), and that instead of the second constraint from item (4) of Definition 6.3, the derivability of negative examples is checked by means of the external atom in item (s7). However, a minimal solution for $\mathcal{M}$ needs to model at least one sequence for deriving each positive example, which is ensured jointly by items (s3), (s5) and (s6). Moreover, no minimal solution w.r.t. the restriction to acyclic sequences of Part (ii) of Theorem 3 is lost by excluding cyclic sequences in Definition 6.9. Finally, item (s7) removes like the second constraint from item (4) all hypotheses that entail a negative example. □

Hence, we have an alternative means to find solutions for forward-chained MIL-problems where planning and generalization are separated in a way such that the background knowledge can be outsourced completely. It is also straightforard to combine state abstraction with the top-down HEX-MIL-encoding from Section 6.2.3, in order to search for sequences of binary background knowledge predicates that derive positive examples in a top-down manner instead of bottom-up. For this, we combine the rules (s1)-(s7) from Definition 6.13 with items (1), (3) and (5) from Definition 6.8, and add

the rules $deduced_a(bg, X, \epsilon_1, ..., \epsilon_\eta, \epsilon_1, ..., \epsilon_\rho, Y, Z, \epsilon_1, ..., \epsilon_{\eta-1}) \leftarrow \&saBinary[](X, Y, Z)$ and $goal(X, Y, Z) \leftarrow pos(\_, X, Y, Z)$, so that sequences are induced using the abstracted states.

6.4 Empirical Evaluation

In this section, we evaluate our approach by comparing it to Metagol in terms of efficiency.

6.4.1 Experimental Setup

For experimentation, we utilized an iterative deepening strategy which incrementally increases a limit for the maximal number of guessed meta-substitutions imposed via a constraint to obtain minimal solutions. In addition, we incrementally increased the number of invented predicates w.r.t. each limit, which proved to be beneficial for performance.

We computed answer sets of our encodings with *hexlite*[3] 0.3.20, which is based on CLINGO 5.1.0. For comparison, we used *SWI-Prolog* 7.2.3 to run Metagol 2.2.0 (Cropper & Muggleton, 2016b). Experiments were run on a Linux machine with 2.5 GHz dual-core Intel Core i5 processor and 8 GB RAM; the timeout was 600 seconds per instance. The results w.r.t. the average running times in seconds are shown in Figures 6.4, 6.6, 6.8, and 6.10, where error bars indicate the *standard error of the mean* ($= s/\sqrt{n}$, where s is the *standard deviation* and n the number of instances) per instance size. In addition, the average running times required for the grounding step are shown in Figures 6.5, 6.7, 6.9, and 6.11. We compared the encodings $\Pi_f(\mathcal{M})$, $\Pi_f^{td}(\mathcal{M})$ and $\Pi_{sa}(\mathcal{M})$ to Metagol for the first two benchmarks, and only used $\Pi_{sa}(\mathcal{M})$ for the third benchmark as discussed below.

For each MIL-problem in this section, we used the meta-rules shown in Figure 6.1, and we implemented it in Metagol and used our HEX-MIL-encodings. External atoms are realized as Python-plugins in our implementation. For operations defined by the background knowledge, we utilized custom list manipulations. The external atoms $\&checkPos[](X_1, X_2, Y, Z)$ and $\&failNeg[meta]()$ in $\Pi_{sa}(\mathcal{M})$ employ breadth-first search for computing all sequences w.r.t. positive examples and for checking the derivability of negative examples, respectively.

The encodings for the benchmark problems and all instances used in the experiments are available at
`www.kr.tuwien.ac.at/staff/kaminski/thesis/hexmil-experiments.zip`.

6.4.2 Hypotheses

The starting hypotheses which we aimed to test with our experiments were the following:

(H6.1) The $\Pi_f(\mathcal{M})$- and $\Pi_f^{td}(\mathcal{M})$-encoding perform better than Metagol as well as the $\Pi_{sa}(\mathcal{M})$-encoding for problems where only few binary predicates are defined by the background knowledge, due to propagation of negative examples by the ASP-solver and since grounding is expected to be feasible in this case.

[3] https://github.com/hexhex/hexlite/

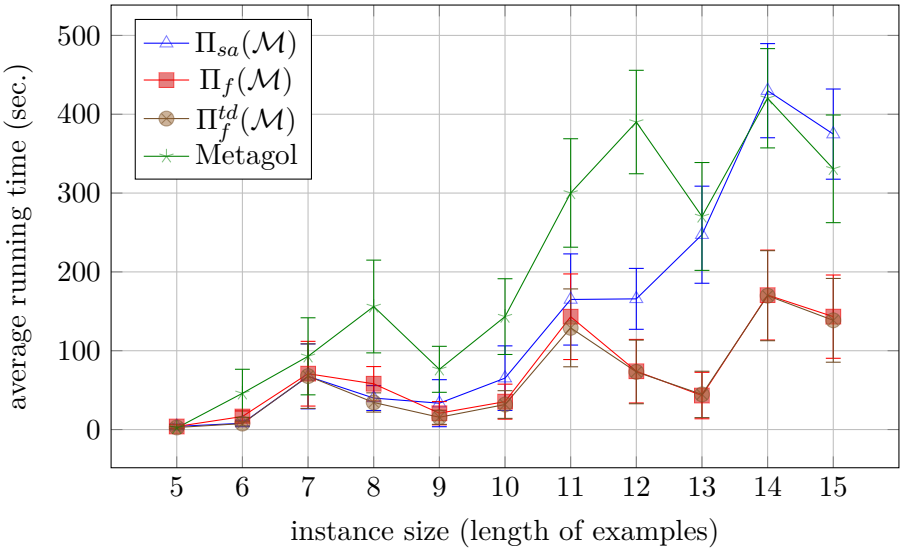

Figure 6.4: Average overall running times for *String Transformation (BM1)*

(H6.2) The $\Pi_{sa}(\mathcal{M})$-encoding performs best when more binary predicates from the background knowledge are applicable to the states of the underlying planning problem because this increases the number of terms that need to be imported from the background knowledge by the $\Pi_f(\mathcal{M})$- and $\Pi_f^{td}(\mathcal{M})$-encoding; separating the planning from the induction problem is expected to be beneficial in this case.

(H6.3) The $\Pi_f^{td}(\mathcal{M})$-encoding has an advantage over the $\Pi_f(\mathcal{M})$-encoding for MIL-problems that do not contain negative examples as it models a goal-driven search for proofs of positive examples as also employed by Metagol. However, the grounding of the $\Pi_f^{td}(\mathcal{M})$-encoding is expected to be larger than the grounding of the $\Pi_f(\mathcal{M})$-encoding since all ground instances of meta-substitutions are generated.

6.4.3 Experiments on Meta-Interpretive Learning

We employed four different benchmark problems based on problems from the literature on MIL and ILP. The respective problems have varying properties regarding the amount of available background knowledge, the number and lengths of training examples and the presences of negative examples; and instances were generated randomly.

String Transformation (BM1)

Our first benchmark is based on Example 6.3, and akin to inducing *regular grammars* as considered by Muggleton et al. (2014). Learning grammars is a suitable use case for MIL as it enables recursive string processing and predicate invention to represent substrings. In contrast to Muggleton et al. (2014), we also allow switching the first two

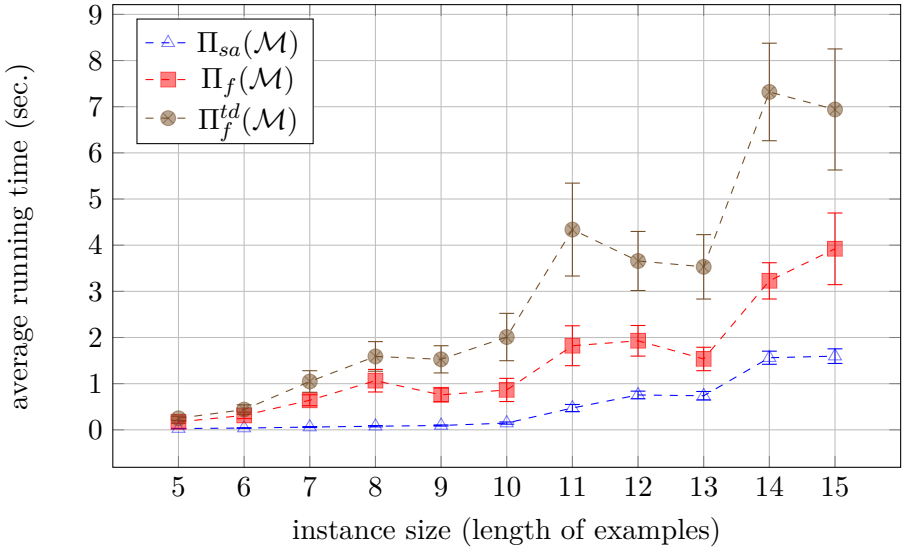

Figure 6.5: Average grounding times for *String Transformation (BM1)*

letters in a string in addition to removing elements, which increases the search space and makes conflict propagation and state abstraction more relevant. For the instances used by Muggleton et al. (2014), Metagol performs much better due to limited branching in the search space. We used positive and negative examples of the form $p([c|X],[c])$, where X is a random sequence of letters a and b. The predicates contained in the background knowledge are *remove*, *switch*, *firstA*, *firstB* and *firstC* (cf. Example 6.3). For this experiment, we used problems containing one positive and one negative example of the same length, and tested lengths $n \in \{1,...,15\}$. The average overall running times of 20 randomly generated instances per n are shown in Figure 6.4 and the average grounding times are shown in Figure 6.5.

East-West Trains (BM2)

The *East-West train challenge* by Larson and Michalski (1977) is a popular ILP-benchmark. The task is to learn a theory that classifies trains based on features (e.g. shapes of cars and types of loads) to be either east- or westbound. In our benchmark, eastbound trains are positive and westbound trains are negative examples, where trains are represented by lists. The background knowledge defines the operation *removeCar* which removes the first car from a train; and we declare 50 different unary predicates, e.g. *shape_rectangle* or *load_3_triangles*, for checking properties of the remaining part of a train. We used a data set of 10 eastbound and 10 westbound trains proposed by Michie et al. (1994) that was also considered by Muggleton et al. (2015). We generated instances of size $n \in \{4,6,8,10,12,14,16\}$ by randomly selecting n from the 20 trains, s.t. $n/2$ were eastbound, and averaged the running times of 10 instances for each problem size. The results w.r.t. the overall running times can be found in Figure 6.6 and the

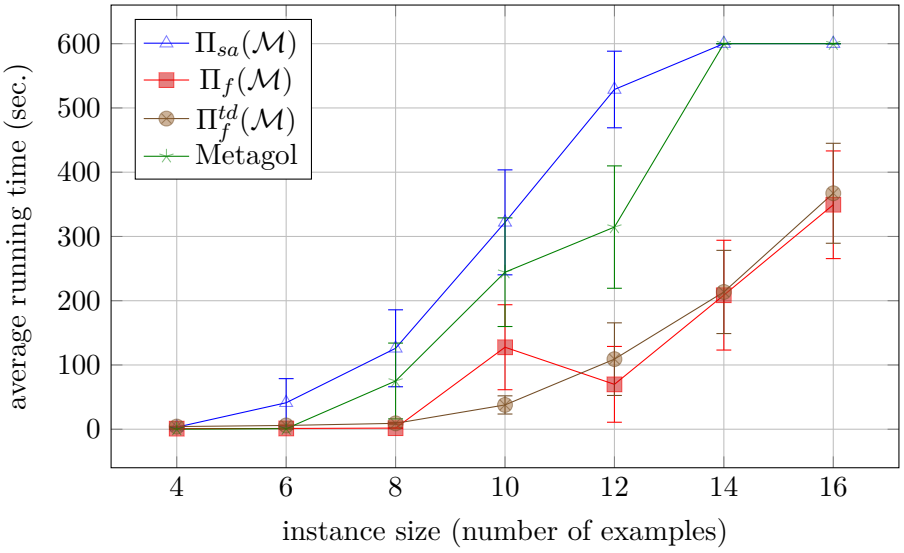

Figure 6.6: Average overall running times for *East-West Trains (BM2)*

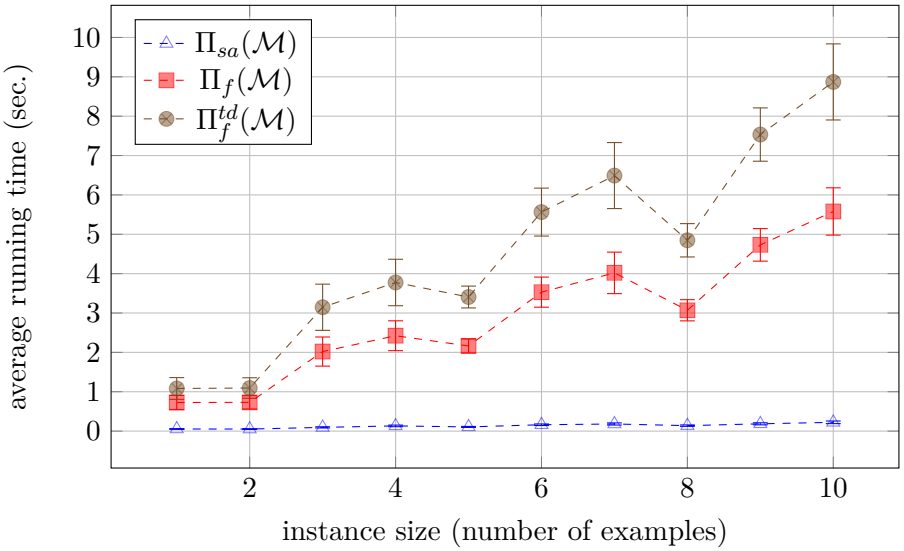

Figure 6.7: Average grounding running times for *East-West Trains (BM2)*

grounding times are shown in Figure 6.7.

Robot Waiter Strategies (BM3)

For our third experiment, we used a problem by Cropper and Muggleton (2016a) that consists in learning robot strategies: customers sit at a table in a row, and a waiter robot serves each customer her desired drink, which is either tea or coffee. Initially, the

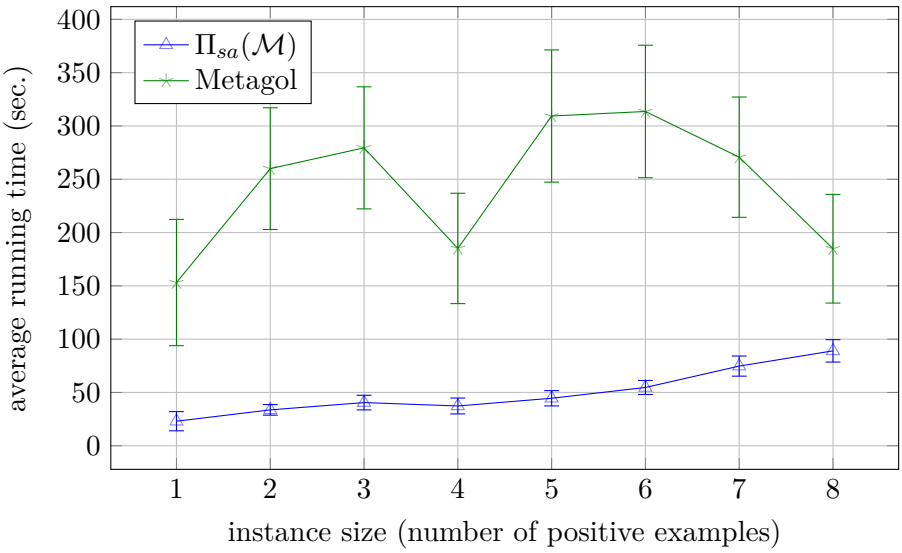

Figure 6.8: Average overall running times for *Robot Waiter Strategies (BM3)*

robot is located at the left end of the table and each customer has an empty cup. In the goal state, each cup contains the desired drink and the robot is at the right end of the table. States are represented using lists, and positive examples map an initial state to a goal state considering different numbers of customers and preferences for drinks. The actions are defined by binary background knowledge predicates *move_right*, *pour_coffee* and *pour_tea*, and the fluents by unary background knowledge predicates *wants_coffee*, *wants_tea* and *at_end*.[4] A solution constitutes a planning strategy by generalizing a plan for each positive example.

For this benchmark, solutions are constrained to be *functional*, i.e. to map an initial state only to the unique respective goal state and not to any non-goal state. Accordingly, negative examples are implicitly given by all binary atoms that map an initial state to a non-goal state. In Metagol, solutions can be restricted to functional theories by means of a property declaration, and we also integrated a corresponding check in the implementation for the external atom *&failNeg*[*meta*]().

We generated random instances similar to Cropper and Muggleton (2016a), where each positive example has a random number of $i \in [1, 10]$ customers with random drink preferences, and the instance size is measured in terms of the number of positive examples ranging from 1 to 8. For each instance size we averaged the running times of 20 problem instances. Figure 6.8 shows the average overall running times, and Figure 6.9 the average amounts of time required for grounding.

[4] In contrast to Cropper and Muggleton (2016a), we omitted the action *turn_cup_over*, as otherwise we obtained timeouts for the majority of instances and all conditions, as it is also the case for Metagol in (Cropper & Muggleton, 2016a).

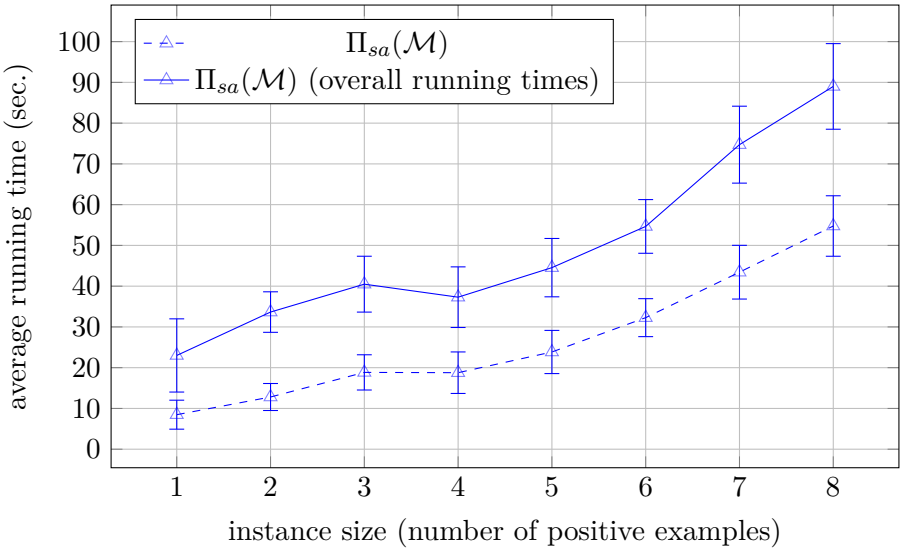

Figure 6.9: Average grounding running times for *Robot Waiter Strategies (BM3)*. The solid line shows the overall running times for comparison; grounding the encodings $\Pi_f(\mathcal{M})$ and $\Pi_f^{td}(\mathcal{M})$ was infeasible for this benchmark.

Drop Lasts (B4)

Our final benchmark problem consists in learning a definite logic program that, given a list of lists, removes the last element from each sublist. For instance, the binary atom $drop([[a,b,a],[b,a,b],e],[e,[a,b],[b,a]])$ is a positive example of the learning task, where the list element e marks the end of the list[5]. The second argument of the positive example is a list constituting the correct output for the input list in the first argument since each sublist is reduced by one element from the right side. A similar benchmark was also employed by Cropper and Muggleton (2016a), who used it to evaluate the learning of solution programs containing higher-order predicates. Since we do not consider learning of higher-order programs here, we defined background knowledge predicates different from the ones used by Cropper and Muggleton (2016a) in order to make the task learnable without higher-order definitions. We used the binary background knowledge predicates *tail*, *reverse* and *shift*, which replace the first sublist in a list by its tail, reverse the first sublist, and shift the elements in the outer list, respectively, as long as the end of the outer list is not reached. In addition, we defined the unary background knowledge atom *end*, which is true for lists that have e as first element.

A peculiarity of this benchmark is that instances only contain positive examples such that the search space is not constrained by negative ones. This means, on the one hand, that conflict propagation performed by the ASP-solver is not expected to yield a significant advantage since conflicts due to the violation of negative examples cannot

[5]Here, it is necessary to mark the end of the list since otherwise grounding of our HEX-encodings was not feasible.

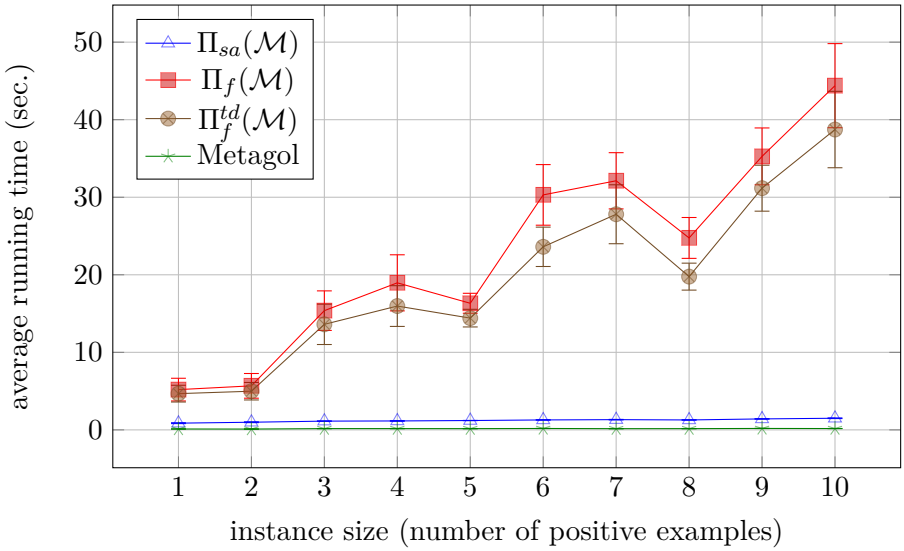

Figure 6.10: Average overall running times for *Drop Lasts (BM4)*

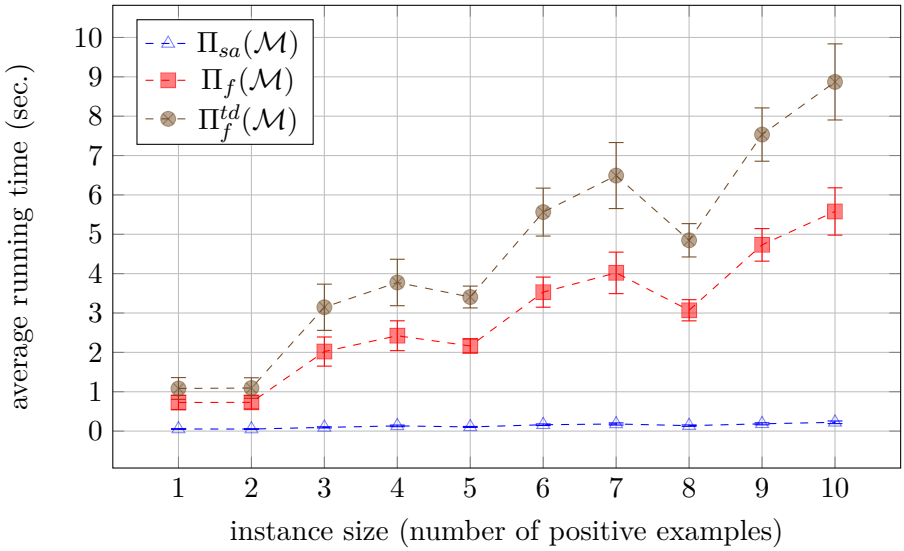

Figure 6.11: Average grounding running times for *Drop Lasts (B4)*

occur. On the other hand, the top-down HEX-MIL-encoding $\Pi_f^{td}(\mathcal{M})$ might be better suited in this case because it models a direct derivation of positive examples similar to the Prolog meta-interpreter exploited by Metagol.

We generated instances containing $n \in [1, 10]$ positive examples, where each list has two sublists containing random sequences of the letters a and b of length $2 \leq l \leq 4$. Due to scaling issues of the encodings $\Pi_f(\mathcal{M})$ and $\Pi_f^{td}(\mathcal{M})$, we could only use lists of short lengths because otherwise, combining the different background knowledge predicates that

all derive new terms resulted in an explosion of the grounding. We averaged the running times of 10 instances for each n, which are shown in Figure 6.10. The average grounding times for each instance size are shown in Figure 6.11.

6.4.4 Discussion of Results

Regarding the benchmarks (B1) and (B2), we found that instances can be solved significantly faster by employing either the $\Pi_f(\mathcal{M})$-encoding or the $\Pi_f^{td}(\mathcal{M})$-encoding than by using Metagol due to conflict propagation in ASP. However, the running times for $\Pi_f(\mathcal{M})$ and $\Pi_f^{td}(\mathcal{M})$ are very similar, which indicates that simulating a top-down derivation of positive examples yields no significant advantage when there are also negative examples that need to be taken into account during search. The encoding $\Pi_{sa}(\mathcal{M})$ performed similar to Metagol for these benchmarks since only two binary predicates, resp. one, are defined by the background knowledge such that solving the planning problem externally does not yield a significant advantage, and the advantage of efficient conflict propagation in ASP is outweighed by the overhead that goes along with outsourcing constraints for negative examples in $\Pi_{sa}(\mathcal{M})$. The $\Pi_{sa}(\mathcal{M})$-encoding performed slightly better in (B1), where two actions are available instead of only one in (B2). Accordingly, the results support hypothesis (H6.1).

For benchmark (B3), we did not obtain results by using the encoding $\Pi_f(\mathcal{M})$ or $\Pi_f^{td}(\mathcal{M})$ for many instances as the grounding was too large due to the imported background knowledge. For instance size 5, the import from the background knowledge already consumed around 100 MB of memory due to the high number of states, and the grounding of the encoding exceeded the available memory. However, the grounding problem could effectively be avoided by using state abstractions with $\Pi_{sa}(\mathcal{M})$, which yielded a significant speed-up compared to Metagol, which provides evidence for the correctness of hypothesis (H6.2). This is due to the fact that by using the $\Pi_{sa}(\mathcal{M})$-encoding, the planning problem is split from the generalization problem such that only one precomputed plan per positive example is considered, which largely reduced the search space. Overall, the performance could be improved by one of our encodings w.r.t. Metagol in all benchmarks where negative examples are propagated by the ASP-solver, whereby state abstraction was crucial when many different actions are defined by the background knowledge, but may decrease efficiency otherwise.

Regarding benchmark (B4), where the MIL-problems contain only positive examples, we found that Metagol is much faster than our encodings since using ASP does not provide any advantage w.r.t. the propagation of negative examples here. The results also show that the $\Pi_f^{td}(\mathcal{M})$-encoding can have an advantage over $\Pi_f(\mathcal{M})$, supporting our hypothesis (H6.3). However, the difference between $\Pi_f^{td}(\mathcal{M})$ and $\Pi_f(\mathcal{M})$ is relatively small. As mentioned before, benchmark (B3) uses three binary background knowledge atoms, which can be applied in different orders to produce a large number of new terms that need to be imported by $\Pi_f^{td}(\mathcal{M})$ and $\Pi_f(\mathcal{M})$. For this reason, the $\Pi_{sa}(\mathcal{M})$-encoding, which does not need to import constants from the background knowledge, performs much better here and nearly matches the performance of Metagol in this experiment. Hence,

the results are also in line with hypothesis (H6.2).

With respect to the grounding step, we found that grounding the $\Pi_f(\mathcal{M})$-encoding required significantly more resources in terms of running time as well as the size of the grounding than grounding $\Pi_{sa}(\mathcal{M})$, in the case of both benchmarks (B1) and (B2). The reason is that only states need to be considered which occur in a sequence of binary background knowledge atoms that derives a positive example for grounding the encoding $\Pi_{sa}(\mathcal{M})$, while $\Pi_f(\mathcal{M})$ also imports all constants that are potentially relevant for deriving some positive or negative example. However, the advantage of $\Pi_{sa}(\mathcal{M})$ w.r.t. the grounding step is canceled out for benchmarks (B1) and (B2) due to the overhead that goes along with outsourcing the check for negative examples and the small advantage in terms of search space pruning. Moreover, the grounding time required for (B1) and (B2) in general is negligible compared to the solving time.

In contrast, we observed that the running time required for grounding the encoding $\Pi_{sa}(\mathcal{M})$ in the case of (B3) makes up more than half of the overall running time. This is explained by the fact that the external atoms $\&checkPos[](X_{id}, X, Y, Z)$, $\&saUnary[](X, Y)$ and $\&saBinary[](X, Y, Z)$ need to be evaluated during grounding due to value invention, which accounts for the major fraction of the overall grounding time.

In general, we observe that the grounding time is less for the $\Pi_f(\mathcal{M})$ than for $\Pi_f^{td}(\mathcal{M})$, while the overall solving time is similar, or slightly better for $\Pi_f^{td}(\mathcal{M})$ in the case of benchmark (B4). This indicates that there is a tradeoff between the resources required for solving and grounding, respectively, and the $\Pi_f^{td}(\mathcal{M})$-encoding requires significantly less time after the grounding step.

We also tested the effect of fixing the number of invented predicates, and obtained timeouts for many instances which could be solved otherwise. The reason is that the availability of additional predicate symbols blows up the search space at the last iteration of the iterative deepening search. Consequently, the advantage of finding solutions with fewer invented predicates faster compensates for the time invested in restarting the solver many times during iterative deepening.

6.5 Further Discussion

In this section, we first discuss the practical implications of constraining the shapes of meta-rules that can be employed for learning, and the limitations of our state abstraction technique. Additionally, we discuss some possible mitigations w.r.t. these limitations, which are the subject of future work.

6.5.1 Meta-Rules

In this chapter, we focused on meta-rules according to Equation 6.1, and restricted the form of meta-rules and of the background knowledge for the encodings $\Pi_f(\mathcal{M})$, $\Pi_f^{td}(\mathcal{M})$ and $\Pi_{sa}(\mathcal{M})$. At this, the fragment of *dyadic Datalog*, i.e. the class of Datalog-programs with predicates of arity at most two, is extremely important in practice as it is suitable whenever input data is given in the form of a graph. Moreover, the seminal paper on

MIL mainly focused on the program class H^2_2 and shows that it has *Universal Turing Machine* expressivity (Muggleton et al., 2015). Accordingly, considering only hypotheses from the class H^2_m does not constitute a severe restriction.

On the other hand, the set of hypotheses that can be learned by the encodings based on forward-chained meta-rules and extensional background knowledge is restricted compared to the solutions that can be obtained by using the general encoding. In particular, as e.g. the meta-rule $P(X,Y) \leftarrow Q(Y,X)$ is not forward-chained, it is not possible to learn the inverse of binary predicates from the background knowledge. For instance, when a predicate *move_right* is contained in the background knowledge, it is not possible to learn a predicate $move\_left$. In this case, the inverses of binary predicates could be added to the background knowledge explicitly. Furthermore, the restriction to extensional background knowledge prevents dependencies of the background knowledge on the induced hypothesis, i.e. predicates in the background knowledge cannot be defined in terms of predicates defined by a solution. However, the majority of MIL-problems considered in the literature are forward-chained, and they do not employ background knowledge that depends on the respective hypothesis as usually only invented predicates are used for rule heads in a hypothesis.

Intuitively, forward-chained meta-rules are natural for applications where binary examples represent a mapping from their first to their second argument, e.g. of an initial state to a goal state in a planning scenario or a string transformation, and where sequences of operations need to be applied to obtain the second from the first argument. Many MIL-problems resulting from practical applications fall into this class. Previous applications of MIL have been mainly considered in three different areas: *Robot Planning*, e.g. by Cropper and Muggleton (2014, 2015, 2016a); *String/Language Processing*, e.g. by Lin et al. (2014) and by Cropper et al. (2015); and *Computer Vision*, e.g. by Dai et al. (2017). We found that most of the MIL-problems considered in the first two areas solely employ forward-chained rules and extensional background knowledge, or in some cases can easily be transformed to forward-chained rules. However, there are also some applications of MIL in the literature where a mapping to forward-chained meta-rules is not (easily) possible (Tamaddoni-Nezhad et al., 2014; Farquhar et al., 2015; Dai et al., 2017).

6.5.2 Limitations of State Abstraction

With respect to the degree of nondeterminism of the planning problems associated with a forward-chained MIL-problem, we can distinguish two factors that impact the size of the search space. First, many different (potentially nondeterministic) actions may be applicable in the different states of a planning problem while there are only few valid plans. Second, there may also be many different action sequences constituting solutions to the respective planning problem.

In the first case, the size of the search space can be reduced compared to Metagol by precomputing correct plans in the encoding $\Pi_{sa}(\mathcal{M})$, which also reduces the size of the grounding. While Metagol generates meta-substitutions based on all applicable actions, $\Pi_{sa}(\mathcal{M})$ only considers actions and states that are part of a correct plan. In the second

case, the size of the search space generated by $\Pi_{sa}(\mathcal{M})$ is closer to the size of the search space explored by Metagol because all possible plans need to be computed and considered during induction. This is necessary since it cannot be decided beforehand which selection of plans allows for a minimal solution w.r.t. the number of rules. Note that, for the same reason, the search space of Metagol also must contain all possible plans w.r.t. positive examples.

Accordingly, the effectiveness of the encoding $\Pi_{sa}(\mathcal{M})$ depends on a tradeoff between the number of actions applicable to states and the number of plans that can be generated for positive examples, and it has an advantage when there are many possible actions but only few plans. Due to the grounding bottleneck, the encoding $\Pi_{sa}(\mathcal{M})$ is likely to be less efficient than Metagol when there are many possible plans and according states that need to be imported. It is an open challenge to tackle problems of this type efficiently by using state abstraction.

As noted in Section 6.3, our approach could be extended by filtering techniques to preselect plans by the external atom in order to avoid the import of all possible plans for positive examples. At the same time, this would be difficult to realize in Metagol where planning and generalization are performed simultaneously. For instance, the number of plans could be restricted by analyzing them, and filtering those that are redundant for obtaining a minimal solution. Furthermore, only a limited number of plans could be imported and the impact on the accuracy investigated to find a good balance between efficiency and compactness of the hypothesis.

When computing sequences that derive positive examples according to Definition 6.9, cyclic sequences need to be avoided. At the same time, cyclic sequences potentially allow to induce a smaller hypothesis for a given MIL-problem, such that part (ii) of Theorem 6.4 is restricted to acyclic sequences as well. However, note that in general, a shorter sequence that derives the second argument of a positive examples from its first argument is obtained by removing cycles. Hence, in practice, the prevention of cyclic plans is often reasonable as, e.g. considering a robot planning scenario, one does not want the robot to loop many times between the same states. Furthermore, one is usually interested in learning a strategy that generalizes minimal (or at least reasonably short) plans. Consequently, there is a tradeoff between the lengths of plans that are considered for learning a strategy, and the size of a hypothesis that generalizes them. Potentially more compact hypotheses can be obtained by allowing cyclic plans, but infinite loops must be prevented.

One way to relax the acyclicity condition would be to allow for a fixed number of cycles in Definition 6.9, which may enable the induction of a smaller hypothesis. To empirically investigate the effect of allowing different numbers of cycles in sequences w.r.t. the accuracy that can be achieved is subject to future work. Moreover, the possibility of cyclic sequences also poses a problem for termination in Metagol, where a different approach is taken to avoid infinite loops. It relies on ordering constraints over predicate arguments of meta-rules w.r.t. a total ordering over terms, which constrain the hypothesis space as well. Similar ordering information could alternatively be employed in our approach to prevent the generation of infinitely many sequences.

6.6 Related Work

As already discussed at the start of this chapter, our approach is most closely related to the work by Muggleton et al. (2014) which also applies ASP to MIL. However, the ASP-encoding there is tailored to the induction of grammars, and grounding issues or modeling a procedural bias are not considered.

In addition, several other ILP systems based on ASP have been developed, e.g. (Otero, 2001; Ray, 2009; Law et al., 2014), which also mainly rely on an ASP-solver for computing solutions. Different from our approach, *default negation* is allowed in the background knowledge and hypotheses, and induced programs are interpreted under the *stable model semantics*. Moreover, examples are partial interpretations in the approach by Law et al. (2014). The declarative bias is defined by *mode declarations* instead of meta-rules in the mentioned approaches, which enables a more fine-grained specification of the hypothesis space; but, to the best of our knowledge, none of them models a procedural bias w.r.t. rule introduction in ASP itself. The *XHAIL* system bounds the search space by splitting the learning process into phases, where a *Kernel set* of ground rules is computed deductively and generalized in an *induction phase*. However, in contrast to the integration of object-level deduction and meta-level induction in our encoding, the phases are executed sequentially.

Compared to ASP-based systems, the MIL-approach has the advantage that meta-rules effectively limit the search space and, in particular, can guide the process of predicate invention, which is regarded as a very hard problem due to its high combinatorial complexity (Dietterich et al., 2008). In addition, intensional background knowledge that manipulates complex terms is difficult to integrate in ASP, while the query-driven procedure exploited by Metagol is well-suited for this.

6.7 Conclusion and Outlook

In this chapter, we presented a general HEX-encoding for solving MIL-problems that interacts with the background knowledge via external atoms and restricts the search space by interleaving derivations on the object and the meta level. In addition, we introduced modifications of the encoding for certain types of MIL-problems and a state abstraction technique to mitigate grounding issues that are hard to tackle otherwise.

Our approach combines several advantages of Metagol and ASP-based approaches, and it is very flexible as it allows to plug in arbitrary (monotonic) theories as background knowledge. Moreover, our encodings can easily be adjusted, e.g. by adding further constraints to limit the import of background knowledge. For instance, we also tried to delay the import of background knowledge by restricting the initial import to chains of a limited length. This resulted in a significant speed-up for many MIL-problems, but minimality of solutions is not guaranteed. Nevertheless, in our tests, solutions that were not considerably larger than solutions of other instances could be obtained instead of timeouts. To investigate how this and similar modifications affect the accuracy w.r.t. a test data set remains for future work.

The potential of an ASP-based approach for MIL is supported by our empirical evaluation; and our techniques could also be exploited in future implementations. Here, we use the HEX-formalism because it is very convenient for prototype implementations. Other formalisms could be used as well, e.g. the theory interface of *Clingo 5* (Gebser et al., 2016), which would potentially increase performance. In particular, employing optimization of *weak constraints* is expected to be beneficial for efficiency as currently the solver needs to be restarted many times during iterative deepening.

In the further development of our implementation, our goal is also to employ more sophisticated planning algorithms for computing the sequences used by our state abstraction technique, and to interface a Prolog-interpreter for processing the background knowledge and for checking negative examples.

CHAPTER 7

Hybrid Classification

In this chapter, we present a second application of HEX-programs in the area of machine learning, based on the paper (Eiter & Kaminski, 2016). While the approach developed in Chapter 6 implements a method for logic-based learning itself by utilizing the HEX-formalism, an external statistical machine learning model is integrated by means of external atoms in the approach described here.

For several decades, huge efforts have been devoted to automate logical reasoning and to develop methods for statistical learning and inference in different research areas. While these areas are rather mature, it became evident that many real-world domains require both logical and statistical reasoning as they comprise complex relational as well as uncertain information. Consequently, *statistical relational learning* (*SRL*) has gained momentum, and many approaches which combine statistical and logical methods have been developed (see (Getoor, 2007) for an overview).

One of the basic tasks in SRL is *collective classification*, which is simultaneously finding correct labels for a number of interrelated objects; this has applications in many concrete domains, e.g. classification of interlinked documents, part-of-speech tagging and optical character recognition (Sen et al., 2010). A further such application is to predict the labels (i.e., class memberships) of objects in a complex visual scene that contains many objects of different classes. Even if advanced and robust algorithms for object recognition have been developed, e.g. *SIFT* descriptors (Lowe, 1999) and the *bag of keypoints* approach (Csurka et al., 2004), they may fail unavoidably and yield ambiguous results due to few training data, noisy inputs, or inherent ambiguity of visual appearance (e.g. a lemon and a tennis ball might be indistinguishable in a low resolution image (Rabinovich et al., 2007)). It is then still possible to draw on further information from the scene in which an object occurs to disambiguate its label.

For an example, consider the street scene in Figure 7.1, where object 2 is wrongly labeled as 'building' in the center image. This misclassification could be resolved by considering all object labels simultaneously and drawing on background knowledge that

Figure 7.1: Objects in a scene from the LabelMe dataset (Russell et al., 2008) with predicted labels

wheels normally appear at the bottom of a car. When taking such background knowledge into account, the probability for labeling object 2 as 'car' could be increased.

Using logical reasoning, a natural approach to formalize admissible labelings of objects respecting their interrelations would consist in imposing logical constraints on labelings and using constraint programming techniques to compute *possible worlds* represented by complete labelings. While this approach yields all consistent labelings, it neglects (hidden) features of the concrete classification problem. Hence, it is desirable to combine constraints over label assignments with the output of a statistical classifier processing (low-level) object features.

A naive way to integrate the output of a classifier with a set of logical constraints would be to use a ranking over all labels for each object induced by the probability distributions given by the classifier, and to compute the labeling that maximizes the rank of the assigned labels while satisfying all constraints. However, in real-world domains this approach turns out to be too restrictive. First, real data necessarily has exceptions that cannot all be modeled, which may prevent that a consistent solution is found. Second, this approach retains no information about the metric distance between label probabilities, which is essential for deciding whether a label should be changed to a less likely one in order to satisfy some constraint.

In this chapter, we bridge the gap between combinatorial and statistical object classification by encoding the context of a concrete *collective classification problem* (*CCP*) in a set of ASP-rules and constraints that formalize restrictions over admissible label assignments w.r.t. the given relational structure; and we assign a probabilistic semantics to our encoding by employing the LP^{MLN}-formalism (Lee & Wang, 2016). Using ASP to formalize context knowledge, we can combine multiple context relations in even complex constraints and utilize *closed world reasoning* to express e.g. that objects not containing car parts should not be labeled as cars. Moreover, by employing HEX-programs[1] instead of ordinary ASP for computing solutions of the resulting LP^{MLN}-encoding, we are able to integrate a statistical object classifier and modules for computing relations between objects in a scene image using external atoms.

[1]In the first version of our work on hybrid classification (Eiter & Kaminski, 2016), ordinary ASP was used instead of the HEX-formalism, where the integration of external sources such as a statistical classifier and a spatial reasoning module was created ad-hoc, relying on an additional pre-processing step. Here, we present a further developed approach based on HEX-programs.

The content of this chapter is structured as follows:

- In Section 7.1, we start by providing the necessary background on LP^{MLN}.
- In Section 7.2, we define a general framework for solving CCPs that combines a generic local classifier and context constraints into a *hybrid classifier*, which is given semantics via an embedding into LP^{MLN}.
- In Section 7.3, we then show how solutions can be obtained efficiently with a back-translation from LP^{MLN} into HEX-programs with weak constraints (Buccafurri, Leone, & Rullo, 2000), and by leveraging combinatorial optimization capabilities of state-of-the-art ASP-solvers.
- In Section 7.4, we describe a methodology for constructing a hybrid classifier for a specific domain by designing and tuning a context encoding. To the best of our knowledge, this has not been considered before.
- In Section 7.5, we examine the usefulness of our methodology with an extensive empirical evaluation in the domain of visual object classification in indoor as well as outdoor scenes. The results provide evidence that hybrid classifiers can significantly improve accuracy, provided that the local classifier works reasonably well, given the outset of few training data, noisy data or ambigous data. Furthermore, they show that constraint selection and the use of a validation set are important elements for increasing accuracy gains.
- In Section 7.6, we discuss related work, before we conclude this chapter in Section 7.7.

Notably, in our approach, logical knowledge representation and reasoning is a first-class citizen, while SRL-approaches often rely on statistical formulations and probabilistic solving methods; this seems less geared towards combinatorial problem solving. Moreover, our encoding can be easily extended by spatial reasoning via rules over extracted facts, as well as by a component for taxonomical reasoning over label categories.

7.1 Background on LP^{MLN}

We assign a probabilistic semantics to the encoding developed in this chapter by utilizing the LP^{MLN}-formalism (Lee & Wang, 2016), which employs weighted rules for combining ASP with probabilistic graphical models based on *Markov Logic Networks* (*MLNs*) (Richardson & Domingos, 2006). LP^{MLN}-programs generalize answer set programs by assigning a weight w to each rule in a program. The weight w is either a real number or α, representing the infinite weight. When grounding an LP^{MLN}-program, every ground weighted rule $w : grnd(r)$ is mapped to the same weight w as its non-ground counterpart $w : r$. A probabilistic semantics is defined for LP^{MLN}-programs as follows.

Definition 7.1 (Unnormalized Weight, adapted from (Lee & Wang, 2016)). *For an LP^{MLN}-program $\mathcal{P}$ and an assignment $\mathbf{A}$, the unnormalized weight of $\mathbf{A}$ under $\mathcal{P}$ is given by*

$$W_{\mathcal{P}}(\mathbf{A}) = \begin{cases} exp\Big(\sum\limits_{w:r\in\mathcal{P}_{\mathbf{A}}} w\Big) & \text{if } \mathbf{A} \in SM[\mathcal{P}], \\ 0 & \text{otherwise,} \end{cases} \tag{7.1}$$

where $\mathcal{P}_{\mathbf{A}}$ represents all weighted rules $w : r$ in $\mathcal{P}$ s.t. $\mathbf{A} \models r$, and $SM[\mathcal{P}]$ contains all assignments $\mathbf{A}$ s.t. $\mathbf{A}$ is a classical answer set of $\mathcal{P}_{\mathbf{A}}$, omitting the weights.

In order to obtain a probability distribution over all assignments w.r.t. an LP^{MLN}-program, the corresponding weights have to be normalized.

Definition 7.2 (Normalized Weight (Lee & Wang, 2016)). *For an LP^{MLN}-program $\mathcal{P}$ and an assignment $\mathbf{A}$, the normalized weight of $\mathbf{A}$ under $\mathcal{P}$ is given by*

$$P_{\mathcal{P}}(\mathbf{A}) = \lim_{\alpha\to\infty} \frac{W_{\mathcal{P}}(\mathbf{A})}{\sum_{\mathbf{A}'\in SM[\mathcal{P}]} W_{\mathcal{P}}(\mathbf{A}')}. \tag{7.2}$$

Lee and Wang (2016) define a (probabilistic) stable model of an LP^{MLN}-program Π to be an assignment $\mathbf{A}$ s.t. $P_{\mathcal{P}}(\mathbf{A}) \neq 0$. Our goal is to use LP^{MLN}-programs for finding a global best labeling for a set of objects, i.e. an answer set encoding a label assignment with maximal probability; we do not discuss conditional probability queries here.

Lee and Wang show a close relationship between ASP with *weak constraints* (Buccafurri et al., 2000) and LP^{MLN}-programs, such that under certain conditions the answer set with the highest normalized probability can be computed directly by an ordinary ASP-solver that exhibits optimization capabilities. We exploit this relationship by utilizing weak constraints of ASP to obtain solutions of our encoding for solving CCPs developed in this chapter. A *weak constraint* is of the form

$$:\sim b_1, ..., b_m, \mathbf{not}\, b_{m+1}, ..., \mathbf{not}\, b_n\, [w], \tag{7.3}$$

where all b_j, $1 \leq j \leq n$, are atoms, and the weight w of a weak constraint c, denoted $weight(c)$, is either an integer constant or a variable occurring in the positive body of the constraint. The *positive* respectively *negative body* of a weak constraint c is defined by $B^+(c) = \{b_1, \ldots, b_m\}$ and $B^-(c) = \{b_{m+1}, \ldots, b_n\}$ as for ordinary ASP constraints. A ground weak constraint has the same weight as the constraint it originates from. For an assignment $\mathbf{A}$ and a set C of weak constraints, the violation cost of C w.r.t. $\mathbf{A}$ is $cost_{\mathbf{A}}(C) = \sum_{c'\in C'} weight(c')$, where $C' = \{c' \mid c' \in C, \mathbf{A} \models a \text{ for all } a \in B^+(c), \mathbf{A} \not\models a \text{ for all } a \in B^-(c)\}$. Given an answer set program P, the answer sets of $P \cup C$ are all those answer sets $\mathbf{A}$ of P such that no answer set $\mathbf{A}'$ of P with $cost_{\mathbf{A}'}(C) < cost_{\mathbf{A}}(C)$ exists. The semantics of weak constraints can be extended straightforwardly to HEX-programs, and the *hexlite*-solver is able to solve HEX-programs with weak constraints by leveraging the optimization capabilities of CLASP (Gebser, Kaminski, et al., 2015).

Lee and Wang define a translation from an answer set program with weak constraints P to an LP^{MLN}-program $\mathcal{P}$, which we denote here by $\tau(P)$ $(= \mathcal{P})$. The translated program

$\tau(P)$ can be obtained from an answer set program with weak constraints P by assigning the infinite weight α to all standard ASP-rules in P, and by transforming weak constraints as in Equation 7.3 to two LP^{MLN}-rules $-\alpha : H \leftarrow \text{not}\, b_1, ..., b_m, \text{not}\, b_{m+1}, ..., \text{not}\, b_n$ and $-w :\leftarrow \text{not}\, H$, where H is a fresh atom not occurring elsewhere.[2] Then, the following correspondence holds.

Proposition 7.1 (adapted from (Lee & Wang, 2016))**.** *For an answer set program with weak constraints P that has an answer set, its answer sets are the assignments $\{\mathbf{A} \mid \nexists \mathbf{A}' : P_{\mathcal{P}}(\mathbf{A}') > P_{\mathcal{P}}(\mathbf{A})\}$, where $\mathcal{P} = \tau(P)$.*

Proof. The statement follows directly from Proposition 3 in (Lee & Wang, 2016). □

For translating an LP^{MLN}-program into an answer set program with weak constraints, we apply τ^{-1}, which is only applicable to LP^{MLN}-programs in which all rules are assigned the infinite weight α and only constraints of the form (2) are assigned arbitrary weights. The translation τ^{-1} works by omitting the weight of rules with non-empty head and by replacing constraints of the form $w : \leftarrow b_1, ..., b_m, \text{not}\, b_{m+1}, ..., \text{not}\, b_n$ by a rule $H \leftarrow b_1, ..., b_m, \text{not}\, b_{m+1}, ..., \text{not}\, b_n$ together with the weak constraint $:\sim \text{not}\, H\, [-w]$, where H is a fresh atom.[3] Under the mentioned restrictions, Proposition 7.1 still holds.

Although we employ HEX-programs instead of ordinary answer set programs for computing the answer sets of an LP^{MLN}-encoding $\mathcal{P}$ with the highest normalized weights in Section 7.3, it is not necessary here to lift the translation τ^{-1} from answer set programs to HEX-programs. The reason is that for computing such answer sets for $\mathcal{P}$, we will first employ the translation τ^{-1}, and then modify $\tau^{-1}(\mathcal{P})$ by integrating external atoms for interfacing an external classifier and a module for computing relations between objects. Importantly, we will show that the resulting HEX-program $\pi(\tau^{-1}(\mathcal{P}))$ has the same answer sets as $\tau^{-1}(\mathcal{P})$ and thus, can be used instead of $\tau^{-1}(\mathcal{P})$.

7.2 LP^{MLN}-Encoding for Hybrid Classification

We aim at applying LP^{MLN}-programs for simultaneously classifying all objects in a visual scene. In order to obtain a complete labeling that is as close as possible to the ground truth, we exploit two sources of probabilistic information regarding the most likely label for a given object. On the one hand, we use a *classifier* that is trained on vectors of object features and predicts the probability of each local label given the features of a single new object. On the other hand, we exploit the *relational context* defined by relations between several objects, by learning the probability of certain label combinations for sets of objects that are related in specific ways, e.g. some objects in an image may be contained in some other objects more or less frequently. In this way,

[2] The logic program rules here are more restricted than in (Lee & Wang, 2016), where rule bodies may also contain *negative formulas*, such that a weak constraint can be translated to a single LP^{MLN}-rule $-w :\leftarrow \neg(b_1, ..., b_m, \text{not}\, b_{m+1}, ..., \text{not}\, b_n)$. Accordingly, we adapt the translation defined by Lee and Wang using two rules instead.

[3] Real-valued weights can be approximated by integers in weak constraints.

the notion of the best label for some object is probabilistically constrained from two sides, and we strive for an optimal label on the basis of all probabilistic information available. We refer to our combination of local classifier and relational context as *hybrid classifier* (*HC*); notably, the relational component has a richer structure than in most related approaches on collective classification.

In this section, we define an HC in form of an LP^{MLN}-encoding that combines a local classifier with a set of weighted context constraints over label assignments w.r.t. the relational structure. We start by defining *collective classification problems* (*CCPs*) based on the definition in (Sen et al., 2010), but we generalize the neighborhood function used there to arbitrary relations between objects. First, we introduce a schema, on the basis of which a group of CCPs can be defined.

Definition 7.3 (Collective Classification Schema)**.** *A* collective classification schema *(*CCS*) is represented by a pair* $S = \langle L, R \rangle$ *consisting of*

- *a set* $L = \{l_1, ..., l_m\}$ *of possible object labels,*
- *a family* $R = \{R_0, ..., R_k\}$ *of sets of* 0- *to* k*-ary* context relation names.

The sets $R_i \in R$ contain names for i-ary relations between objects that can, for instance, be extracted from a scene image, e.g. a binary relation entailing all pairs of objects where the first object is contained in the second object, or a ternary relation stating that an object is located in-between two other objects. The set L contains possible object labels, e.g. 'car' and 'tree' for objects in a street scene.

A CCS is instantiated by a CCP, by fixing the set of objects that need to be classified together with their local object features as well as the concrete relations occurring between them, as follows.

Definition 7.4 (Collective Classification Problem)**.** *A* collective classification problem *(*CCP*) is represented by a triple* $\mathcal{C} = \langle S, O, e \rangle$ *consisting of*

- *a CCS* $S = \langle L, R \rangle$*,*
- *a set* $O = \{o_1, ..., o_n\}$ *of objects with associated features* $f(o_i)$ *for each* $o_i \in O$*,*
- *a function* $e : \bigcup_{R_i \in R} \rightarrow O^k$ *that maps each* i*-ary relation name to a concrete* i*-ary relation over objects.*

A solution for a CCP is a complete labeling *represented by a mapping* $\lambda : O \rightarrow L$*, assigning a label from* L *to each object in* O*.*

Next, we introduce *local classifiers*, where we abstract from the level of particular object features and assume a classifier that is able to return a probability distribution over all labels for each object by processing their corresponding features. Subsequently, we draw on the information provided by a local classifier c for hybrid classification by integrating c with a context encoding into an HC.

Definition 7.5 (Local Classifier)**.** *Given a CCS* $S = \langle L, R \rangle$*, a* local classifier c *is a function that maps the feature vector* $f(o)$ *of an object* o *to a discrete probability distribution* P_o^c *over all labels in* L *(on the basis of their associated feature vectors).*

Due to the generality of the approach, different kinds of classifiers, e.g. *Logistic Regression* or *Neural Networks*, can be utilized to instantiate the local classifier c.

Example 7.1. For the scene in Figure 7.1, we construct a corresponding CCP $\mathcal{C} = \langle S, O, e\rangle$ with $S = \langle\{car, building, wheel\}, \{\emptyset, \emptyset, \{contains\}\}\rangle$, $O = \{o_1, o_2\}$ (omitting 'object 3') and $e(contains) = \{\langle o1, o2\rangle\}$. We assume that the classifier c for $\mathcal{C}$ yields, based on the object features $f(o_i)$ extracted from the image, $P^c_{o_1}(car) = 0.4$, $P^c_{o_1}(building) = 0.5$, $P^c_{o_1}(wheel) = 0.1$, $P^c_{o_2}(car) = 0.1$, $P^c_{o_2}(building) = 0.1$ and $P^c_{o_2}(wheel) = 0.8$. △

In other approaches (Galleguillos et al., 2008; Rabinovich et al., 2007; Angin & Bhargava, 2013), relations between objects are often used to conditionalize the probability distribution of label combinations of the involved objects. As we use ASP-constraints to describe the relational context, we use the relations in the sets R_i differently, i.e. to state restrictions over expected label assignments via relations that may be derived from further relations together with other supposed label assignments.

Following Richardson and Domingos (2006), the weight of a context constraint in LP^{MLN} can be interpreted as the logarithm of the odds between a possible world where it is satisfied and one where it is not (called the *log odds*), other things being equal. In general, context constraints are not independent from each other, thus changing their truth value also changes the value of other constraints. However, as we consider cases with only few training data (such that the classifier output can still be improved by considering the context), it is unfeasible to learn all interactions between constraints from it. Thus, we assume *bona fide* independence of context constraints and straight use the *log odds* for the constraints calculated from the training instances as weights.

The restrictions over label assignments in terms of the relational context are formalized by a *context encoding* as follows:

Definition 7.6 (Context Encoding)**.** *Given a CCS $S = \langle L, R\rangle$, we use the following designated predicates:* context relation predicates $\mathcal{R}{=}\bigcup_{R_i\in R} R_i$ *and* helper predicates $\mathcal{H}$ *ranging over tuples of objects, and the* label assignment predicate *a_label ranging over pairs of objects and labels. A* context encoding *E for S is an LP^{MLN}-program that consists of rules of the form*

$$\alpha : h(\vec{X}) \leftarrow b_1(\vec{X}), \ldots, b_k(\vec{X}), \mathbf{not}\, b_{k+1}(\vec{X}), \ldots, \mathbf{not}\, b_m(\vec{X}), \tag{7.4}$$

where $h \in \mathcal{H}$ and $b_1, \ldots, b_m \in \mathcal{R} \cup \mathcal{H} \cup \{a\_label\}$; and constraints of the form

$$w : \leftarrow b_1(\vec{X}), \ldots, b_k(\vec{X}), \mathbf{not}\, b_{k+1}(\vec{X}), \ldots, \mathbf{not}\, b_m(\vec{X}), \tag{7.5}$$

where $b_1, \ldots, b_m \in \mathcal{R} \cup \mathcal{H} \cup \{a\_label\}$, and w is the log odds for the constraint being satisfied given the extensions of the predicates in $\mathcal{R} \cup \mathcal{H}$ (as learned from training data).

The helper predicates in $\mathcal{H}$ are used to recursively aggregate relations and label assignments into new relations, which can be utilized to restrict permissible assignments.

Example 7.2 (cont'd). We define a simple context encoding E for S from Example 7.1, using the context relation predicate *contains* and the helper predicate *has_car_part*:

$$\alpha : \mathit{has\_car\_part}(X) \leftarrow \mathit{contains}(X, Y), \mathit{a\_label}(Y, \mathit{wheel}) \tag{7.6}$$

$$1.95 : \leftarrow \mathbf{not}\, \mathit{a\_label}(X, \mathit{car}), \mathit{has\_car\_part}(X) \tag{7.7}$$

The particular weight is chosen for the context constraint because we suppose here that we have observed 28 cases in our training data where an object that has a car part is actually a car, and four cases where it is not, i.e. the log odds for the constraint being true given the extension of the predicate *has_car_part* are $ln(28/4) \approx 1.95$.

Taxonomic reasoning can easily be added by introducing, e.g., a rule that derives all labels representing car parts. Likewise, spatial reasoning can be implemented by inferring further relations from the given relations (e.g., an object overlaps with another object if one contains the other). △

We combine a local classifier for a CCS $S = \langle L, R\rangle$ and a context encoding for S into an HC that yields a solution for a CCP $\mathcal{C} = \langle S, O, e\rangle$ as follows:

Definition 7.7 (Hybrid Classifier Encoding)**.** *Given a CCP $\mathcal{C} = \langle S, O, e\rangle$ for a CCS $S = \langle L, R\rangle$, a local classifier c, and a context encoding E for S, the* hybrid classifier (HC) *for $\mathcal{C}$ is represented by an LP^{MLN}-program $\Pi_{\mathcal{C}}(c, E) = E \cup A(c, O) \cup I(\mathcal{C})$ where the* classifier assignment encoding $A(c, O)$ *contains*

(1) the weighted facts

$$\alpha : \mathit{label}(l) \text{ for each label } l \in L,$$

$$\alpha : \mathit{clf}(o, l, p) \text{ for each } o \in O \text{ and } l \in L, \text{ where } p = ln\left(\tfrac{P_o^c(l)}{1-P_o^c(l)}\right),$$

(2) the two guessing rules

$$\alpha : \mathit{a\_label}(O, L) \leftarrow \mathit{object}(O), \mathit{label}(L), \mathit{not}\, \mathit{n\_a\_label}(O, L),$$

$$\alpha : \mathit{n\_a\_label}(O, L) \leftarrow \mathit{object}(O), \mathit{label}(L), \mathit{not}\, \mathit{a\_label}(O, L),$$

(3) the unique assignment constraint

$$\alpha : \leftarrow \#\mathit{count}\{L : \mathit{a\_label\_prob}(X, L, P)\} \neq 1, \mathit{object}(X),$$

(4) the weighted classifier constraint

$$P : \leftarrow \mathit{not}\, \mathit{a\_label\_prob}(O, L, P), \mathit{clf}(O, L, P)$$

and the rule

$$\alpha : \mathit{a\_label\_prob}(O, L, P) \leftarrow \mathit{a\_label}(O, L), \mathit{clf}(O, L, P),$$

and the CCP instance encoding $I(\mathcal{C})$ *contains*

(5) the weighted facts

$\alpha : object(o_i)$ *for each object* $o_i \in O$, *and*

$\alpha : r_i(o_1, \ldots, o_i)$ *for each* $r_i \in R_i$ *and each* $\langle o_1, \ldots, o_i \rangle \in e(r_i)$.

Here, item (5) represents the input part of the HC, while items (2) to (4) are fixed; items (2) and (3) ensure that each object gets exactly one label, and item (4) assigns the weights obtained from the local classifier to the separate label assignments. Again, we use the log odds between a complete label assignment where a label is assigned vs. not assigned as computed from the output of the local classifier as weight.

Intuitively, a solution of an HC should minimize the violation costs of context constraints, but at the same time maximize the joint classifier probability of the label assignment. As the two optimization criteria may be opposite in general, the goal is a good compromise that yields a better label assignment than the one of the local classifier alone. As it is not clear a priori how much influence the classifier and the context constraints should have on a solution, the probabilities returned by the local classifier in item (4) could be scaled by an influence factor such that its impact on a solution can be varied for tuning an HC.

A solution for a CCP w.r.t. an HC is defined as follows.

Definition 7.8 (HC-Solution)**.** *A solution for a CCP* $\mathcal{C}$ *provided by an HC* $\Pi_{\mathcal{C}}(c, E)$ *is a solution* λ *for* $\mathcal{C}$ *such that, for some assignment* $\mathbf{A}$*, no assignment* $\mathbf{A}'$ *with* $P_{\Pi_{\mathcal{C}}(c,E)}(\mathbf{A}') > P_{\Pi_{\mathcal{C}}(c,E)}(\mathbf{A})$ *exists and* $a\_label(o_i, l_i) \in \mathbf{A}$ *iff* $\lambda(o_i) = l_i$.

Definition 7.7 encodes the optimization problem by an LP^{MLN}-program that can be translated into an ordinary answer set program with weak constraints, such that a solution according to Definition 7.8 can be extracted from any answer set (cf. Proposition 7.1).

Example 7.3 (cont'd). The LP^{MLN}-program $\Pi_{\mathcal{C}}(c, E)$ representing the HC for $\mathcal{C}$, c and E from the previous examples contains E, weighted facts $\alpha{:}obj(o1)$, $\alpha{:}obj(o2)$, $\alpha{:}label(c)$, $\alpha{:}label(b)$, $\alpha{:}label(w)$, $\alpha{:}clf(o1, c, -0.41)$, $\alpha{:}clf(o1, b, 0)$, $\alpha{:}clf(o1, w, -2.2)$, $\alpha{:}clf(o2, c, -2.2)$, $\alpha{:}clf(o2, b, -2.2)$, $\alpha{:}clf(o2, w, 1.39)$ and $\alpha{:}contains(o1, o2)$ (abbreviating the labels), and items (2) to (4) from Definition 7.7.

Without the context encoding E, the single stable model of the program with the highest normalized weight would contain $a\_label(o1, b)$ and $a\_label(o2, w)$; this does not correspond to the correct labeling of the scene shown rightmost in Figure 7.1. The previous assignment would not satisfy the constraint in E. Hence, when considering E, there are three ways to satisfy it by changing the assigned labels: changing (a) the label of $o1$ to *car*; or the label of $o2$ to either (b) *building* or (c) *car*. As the constraint has weight 1.95 and the label adaptations result in a weight difference of -0.41 for (a) and -3.59 for (b) and (c) for the classifier constraints, only (a) would yield an overall weight improvement. Thus, labeling o_1 as *car* and o_2 as *wheel* is the only solution for $\mathcal{C}$ via $\Pi_{\mathcal{C}}(c, E)$ according to Definition 7.8; this is the correct labeling of the scene.

Note that if the difference between the probability that $o2$ is a *wheel* and e.g. a *building* would be small enough, satisfying the constraint (7) in E by changing the label of $o2$ could actually result in a higher overall weight. Thus, context constraints can

also decrease the accuracy of the resulting labeling, depending on the quality of the probabilities provided by the classifier. △

7.3 HEX-Program for Computing HC-Solutions

As mentioned above, we exploit Proposition 7.1 to obtain an answer set with maximum normalized weights of an LP^{MLN}-program constituting an HC-encoding, i.e. we apply the backtranslation τ^{-1}, whereby floating-point values are approximated by integers, and utilize an ASP-solver. In practice, given an HC $\Pi_{\mathcal{C}}(c, E)$, we transform $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$ into a HEX-program $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$, such that the local classifier and, e.g., a spatial reasoning module, can be interfaced directly from within the encoding, which yields a modular and highly configurable approach. At this, additional options can be provided as inputs to the corresponding external atoms. For instance, the specific type of classifier could be selected or the context relations that should be taken into account could be specified this way.

Furthermore, using external atoms allows information to flow from the program back to the classifier as well as restricted queries to the classifier. Hence, this lays the ground for orienting the classifier based on information derived in the ASP-part in the future, which could be of interest for abductive reasoning. For instance, if there are two rules stating that the object attached to a 'car' or a 'bicycle' is a 'wheel', and the label 'wheel' is assigned to some object, abduction could be used to find that the object is either attached to a 'car' or a 'bicycle'. Subsequently, the classifier could be queried to find the most likely explanation for discovering a 'wheel'.

We define two kinds of external atoms, which are used in the encoding $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$ introduced subsequently. The purpose of the first external atom is to interface the local classifier.

Definition 7.9 (External Classifier Atom)**.** *Given a CCP $\mathcal{C} = \langle S, O, e\rangle$ for a CCS $S = \langle L, R\rangle$ and a local classifier c, we call the external atom $\&classifier[File, X](Y, P)$ an* external classifier atom*, where $File$ is replaced by a resource locator, e.g. an URI or a file path, of a file that implements c, and $f_{\&classif}(\mathbf{A}, File, X, Y, P) = \mathbf{T}$ iff $X \in O$, $Y \in L$ and $P = ln\left(\frac{P^c_{o_i}(l_j)}{1-P^c_{o_i}(l_j)}\right)$.*

The file that implements the classifier can be an arbitrary machine learning model, and in our implementation is realized by a *pickle*-file "model.pkl" that stores a Logistic Regression model as produced by the Python library *scikit-learn*.

The second external atom definition constitutes a generic interface to context relations of different kinds and arities, and, e.g. in the setting of object classification in scene images, the external atom can compute spatial relations between objects based on the pixel data of a scene image.

Definition 7.10 (External Context Atom)**.** *Given a CCP $\mathcal{C} = \langle S, O, e\rangle$ for a CCS $S = \langle L, \{R_0, ..., R_k\}\rangle$, and $i \in \{0, ..., k\}$, we call the external atom $\&context[File](X, Y_0, ..., Y_i)$*

an i-ary external context atom, *where File is replaced by a resource locator, e.g. an URI or a file path, of a file containing information needed for computing e, and* $f_{\&context}(\mathbf{A}, File, X, Y_0, ..., Y_i) = \mathbf{T}$ *iff* $X \in R_i$ *and* $\langle Y_0, ..., Y_n \rangle \in e(X)$.

A concrete instance of a 2-ary external context atom for importing binary spatial relations for object classification is $\&context[File](contains, o1, o2)$, which would evaluate to true w.r.t. the CCP from Example 7.1. However, the same external atom could also be used to retrieve other kinds of relations between objects, e.g. links between websites that should be classified according to their topics.

Given an HC $\Pi_{\mathcal{C}}(c, E)$, we obtain a HEX-program $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$ from the ordinary answer set program $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$ by removing all facts of the forms $clf(o, l, p)$ and $r_i(o_1, ..., o_i)$, and adding the rule $clf(X, Y, P) \leftarrow \&classifier[File, X](Y, P), object(X)$, as well as the rule $r_i(Y_0, ..., Y_i) \leftarrow \&context[File](r_i, Y_0, ..., Y_i)$ for each $r_i \in R_i$.

Proposition 7.2. *Given a CCP* $\mathcal{C} = \langle S, O, e \rangle$ *for a CCS* $S = \langle L, R \rangle$*, a local classifier* c*, and a context encoding* E *for* S*, the answer set program* $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$ *and the* HEX*-program* $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$ *have the same answer sets.*

Proof. It follows from Definition 7.7 and the definition of the oracle function for the external atom $\&classifier[File, X](Y, P)$ in Definition 7.9 that $clf(o, l, p) \in \mathbf{A}$ for every $\mathbf{A} \in \mathcal{AS}(\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E))))$ if and only if the weighted fact $\alpha : clf(o, l, p) \in \Pi_{\mathcal{C}}(c, E)$. The previous holds because $\alpha : clf(o, l, p) \in \Pi_{\mathcal{C}}(c, E)$ if and only if $clf(o, l, p)$ is derived by the rule $clf(X, Y, P) \leftarrow \&classifier[File, X](Y, P), object(X)$ and since there is no other rule that defines the predicate clf. In addition, according to the definition of the back-transformation τ^{-1} (cf. Section 7.1), $\alpha : clf(o, l, p) \in \Pi_{\mathcal{C}}(c, E)$ holds if and only if the fact $clf(o, l, p)$ is contained in $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$. The same reasoning applies to atoms of form $r_i(Y_0, ..., Y_i)$ for $r_i \in R_i$.

Hence, the answer sets of $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$ and $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$ contain the same atoms of the forms $clf(o, l, p)$ and $r_i(Y_0, ..., Y_i)$ for $r_i \in R_i$. Moreover, $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$ and $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$ coincide w.r.t. to all other rules that do not define atoms of the forms $clf(o, l, p)$ and $r_i(Y_0, ..., Y_i)$ for $r_i \in R_i$. Consequently, we obtain that the answer set program $\tau^{-1}(\Pi_{\mathcal{C}}(c, E))$ and the HEX-program $\pi(\tau^{-1}(\Pi_{\mathcal{C}}(c, E)))$ have the same answer sets. □

As a result, we have a means for obtaining HC-solutions by employing a HEX-solver and by interfacing a local classifier and modules for computing context relations via external atoms, without extending the underlying LP^{MLN}-semantics.

7.4 Hybrid Classifier Construction

After having defined HCs abstractly above, we now describe a methodology for constructing a concrete HC for a given CCP, which we also employ in our empirical evaluation. We suggest the following strategy for obtaining a good HC, where the objective is high accuracy of the corresponding solution.

1. *Data and local classifier preparation.* We assume that we are given a set of CCPs that are all defined over the same CCS for training the HC, together with a solution λ for each CCP representing the ground truth. Obviously, the concrete relations e are usually different in each CCP and first must be extracted from the raw data. For testing the influence of different context constraints, it is crucial to use part of the data for validation to avoid overfitting of the designed constraint encoding. Hence, we split the initial data set into a training set, a validation set and a test set. The local classifier is trained on the associated features of all objects in the CCPs in the training set separately.

2. *Designing the context encoding.* Although context constraints theoretically could be learned, e.g. by ILP techniques, the current approach assumes that a domain expert with background knowledge on the particular task for the HC has designed the context encoding. However, failure patterns in the output of the local classifier can be used to guide the design process. For this purpose, the local classifier is first used to classify all objects in the validation set and a *confusion matrix* is compiled, which reveals objects that are difficult to classify for the local classifier and the pairs of labels confused most often. This way, the constraint encoding can be tailored to counter the shortcomings of the local classifier that might result from few, noisy or ambiguous data.

   For a constraint c of the form (6) (see Definition 7.6), its weight w is computed as follows. Determine in the training set the number of ground instances where the label assignment specified by atoms in L is false (resp., true), denoted by f_c (resp., t_c), provided the context described by the atoms in $\mathcal{R} \cup \mathcal{H}$ of (6) is satisfied. If we would not fix these atoms for counting, e.g. in Example 7.2 for (8) each object not containing a car part would count as positive instance. However, in this case we are interested in the odds for an object being a car if it has a car part. The weight w of the constraint c is then $\ln(f_c/t_c)$.

3. *Constraint selection and influence tuning.* After having designed the constraint encoding, the resulting HC could be evaluated already on the test set and used on new CCP instances. However, as discussed in Example 7.3, context constraints may also decrease the overall accuracy of the results. Hence, the constraint encoding E should be evaluated on the validation set first. As constraints may interact, in general each subset C of constraints must be tested to single out the optimal one w.r.t. the validation set. As there are exponentially many C, a heuristics is to assess the influence of each constraint c separately and keep it if the accuracy does not decrease if c is applied alone resp. increases if c is dropped from the set of all constraints, whereby constraints can also be dropped incrementally. In addition, the validation set can be used to tune the influence of the local classifier and the context encoding on the solution, by testing different influence factors.

Example 7.4 (HC in visual scenes). In the context of visual object classification in scene images, a CCS $S = \langle L, R \rangle$ is created by defining the set L of possible labels for the objects

in a class of scenes, e.g. 'car', 'building' and 'tree' for outdoor scenes, and 'table', 'chair' and 'shelf' for indoor scenes, and by fixing the considered set R of relations between objects. Spatial relations such as 'contains', 'intersects' and 'touches' are arguably most prevalent in visual scenes, but R may include also other relations, even relating local features of different objects, such as the binary relation 'has_same_color'.

To turn a set of scene images into a set of CCPs, the image first must be segmented into regions containing single objects. Many procedures for image segmentation exist (see e.g. the survey in (Zhang et al., 2008)); we assume here that the image is already segmented. The visual features of the separate segments represent the input to the local classifier, which needs to be trained on a training set of segments representing objects O from training CCPs $\langle S, O, e\rangle$ and the corresponding set L of labels. The extension e of the relations R must be extracted for each CCP separately from the information provided by the scene image and its segmentation, e.g. by computing spatial relations w.r.t. their bounding boxes or polygon coordinates. Further, implicitly entailed spatial relations can be derived e.g. by employing a spatial reasoning calculus such as RCC8 (Randell et al., 1992).

Suppose we examine the confusion matrix for the local classifier on the validation set for indoor scenes, and we observe that doors are often misclassified as tables (their surfaces look nearly identical). We then could add a constraint c to the encoding E which states that tables are not contained in walls. We compute f_c and t_c by counting the objects in the training set contained in a wall that are non-tables resp. tables; presumably the resulting weight $\ln(f_c/t_c)$ is quite high. After having added several constraints to E, we test how removing single constraints (or sets of them) affects the accuracy on the validation set. In that, we might observe that the constraint prohibiting tables in walls actually decreases the overall accuracy, even if it is mostly satisfied on the training data. Indeed, possibly some doors are still wrongly labeled as 'table' while the correct label 'wall' of a wall is changed to an incorrect one. This might have further implications; e.g. if a constraint states that windows only occur in walls, many windows are misclassified too. This illustrates the importance of constraint selection for HC construction. △

7.5 Empirical Evaluation

In this section, we evaluate two concrete HCs for two different sets of benchmark instances in order to empirically investigate the effect of applying context constraints.

7.5.1 Experimental Setup

For experimentation, we implemented an HC framework in *Python* that enables construction and evaluation of HCs for object classification in scene images. As local classifier, we used a *Logistic Regression* classifier implemented in the *scikit-learn* library (Pedregosa et al., 2011), which is interfaced by means of the external atom $\&classifier[File, X](Y, P)$. We trained the local classifier on features extracted from image segments obtained by the *bag of keypoints* approach (Csurka et al., 2004) using vector quantization of invariant

Figure 7.2: Example of a typical indoor and outdoor scene from the LabelMe dataset (Russell et al., 2008)

image descriptors. For creating the visual vocabulary, we employed *k-means clustering* and *Scale Invariant Feature Transform* (*SIFT*) descriptors (Lowe, 1999); they are suited for our purpose as they are invariant w.r.t. transformations, varying illumination and overlapping objects. To detect and compute SIFT descriptors, we used the *OpenCV*[4] library.

Furthermore, we used the *Shapely*[5] package for Python to calculate concrete spatial relations between object-polygons by an external atom $\&context[File](X, Y_0, Y_1)$ for each scene, based on the *DE-9IM* model (Strobl, 2008). Moreover, for computing the optimal solution of an HC encoding, we utilized *hexlite*[6] 0.3.20, and used CLINGO 5.1.0 as back-end solver.

7.5.2 Hypotheses

Our goal was to ascertain the following hypotheses.

(H7.1) HCs improve the accuracy provided that the local classifier yields sufficiently many correct labels as a basis to correct other labels.

(H7.2) The accuracy gain achieved by employing an HC compared to only using a local classifier can be increased by tuning an HC on a validation set.

(H7.3) HCs perform worse than local classifiers in case the latter classify most objects incorrectly because usually accurate labels cannot be inferred from incorrect labels using context constraints.

7.5.3 Experiments on Hybrid Classification

The experiments have been conducted on two sets of scene images from the *LabelMe* dataset (Russell et al., 2008). We used a custom segmentation obtained manually, as for testing the impact of context constraints the quality of the available segmentations as well as the user-defined labels varied considerably. The data sets used in our experiments,

[4] http://opencv.org/
[5] https://pypi.python.org/pypi/Shapely
[6] https://github.com/hexhex/hexlite/

Table 7.1: Results for local classifier and HC with (+sel) or without (-sel) constraint selection

data set	local classifier	HC -sel	HC +sel
(E1) validation	51.8 %	63.3 %	64.5 %
(E1) test	48.3 %	61.1 %	61.8 %
(E2) validation	56.9 %	71.3 %	72.7 %
(E2) test	56.0 %	72.2 %	72.8 %

the segmentation data, the constraint encodings and all results are available at `http://www.kr.tuwien.ac.at/staff/kaminski/thesis/hc-experiments.zip`.

We used (E1) a set of *indoor office scences* and (E2) a set of *outdoor street scenes*, each containing 120 images, which we split into a training set and validation set of 30 images each, and a test set of 60 images. A typical scene from each data set is shown in Figure 2. For both types, we defined 12 labels for the objects that occur most frequently:

- indoor: 'chair' (c), 'monitor' (mn), 'keyboard' (k), 'mouse' (ms), 'table' (t), 'book' (bk), 'shelf' (s), 'wall' (wl), 'board' (br), 'person' (p), 'door' (d) and 'window' (wi),
- outdoor: 'sign' (sg), 'person' (p), 'tree' (tr), 'window' (wi), 'door' (d), 'street' (st), 'car' (c), 'sky' (sk), 'building' (b), 'sidewalk' (si), 'wheel' (wh) and 'trunk' (trn).

Indoor scenes contain 7 to 23 objects, outdoor scenes 7 to 28. In total, (E1) contains 2046 objects, and (E2) has 2276 objects. We extracted the binary spatial relations 'contains', 'close_to', 'above', 'under', 'overlaps', 'contains_in_bottom_part' and 'higher' from the images for use in our constraint encodings, from which we created an HC for each dataset.

7.5.4 Discussion of Results

After training the local classifiers on all objects in the training sets, we applied them to the validation sets; for indoor scenes, the average accuracy was 51.8 % and for outdoor scenes 56.9 %. We then constructed HCs, following the methodology from Section 7.4, by setting up 20 constraints in each case and selecting a subset of 13 constraints using the validation sets. The accuracy increased for the indoor validation set to 64.5 % and for the outdoor validation set to 72.7 %.

In addition, we tested different influence factors, viz. 0.1, 1, 10, and 100 for the classifier weights; for both data sets, factor 1, i.e. having a balanced impact of the local classifier and the context constraints, yielded the best results. We thus fixed the influence factors to this value.

We then applied the final HCs to the test sets. Overall, for indoor scenes the accuracy increased from 48.3 % to 61.8 %, and the HC was better than the local classifier on 48 scenes and worse on 3 out of 60 scenes. For outdoor scenes, the accuracy increased from 56.0 % to 72.8 %, and the HC was better than the local classifier in 56 cases and worse

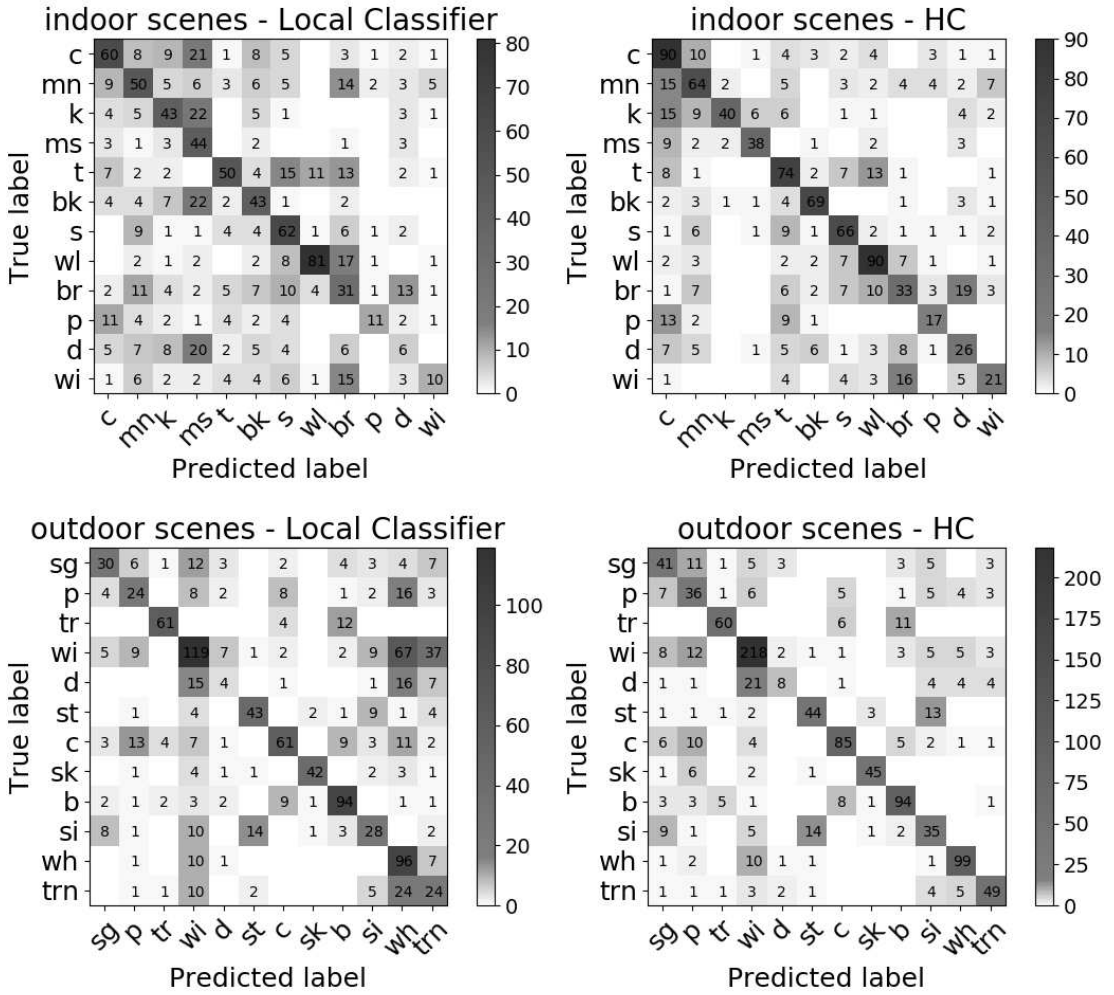

Figure 7.3: Confusion matrices of local classifier and HC test results for indoor and outdoor scenes

in 1 case. Thus our hypothesis (H7.1) could be confirmed w.r.t. our use cases. The test results are summarized in Table 7.1[7].

Regarding (H7.2), simply adding all 20 constraints increased the accuracy for indoor scenes from 48.3 % to 61.1 % and for outdoor scenes from 56.0 % to already 72.2 %;

[7]The results presented here differ from the results in (Eiter & Kaminski, 2016) because we used an improved version of our implementation, which is now based on HEX-programs, and re-trained the local classifier using a more recent version of the Python-library *scikit-learn*. This improves the accuracy of the local classifier for (E1) and slightly worsens it for (E2). Moreover, in (Eiter & Kaminski, 2016), an influence factor of 10 instead of 1 yielded the best results for both data sets. This difference is due to the fact that the new classifier used for the experiments in this chapter returns probabilities that are more concentrated on a single label.

this confirms that constraint selection can further improve accuracy, even though, the difference is relatively small for our use cases (cf. Table 7.1). Influence tuning did not make a difference in our tests since the standard influence factor of 1 proved to work best on the validation set. However, higher or lower influence factors might be more suited in other cases.

To provide more details on the effect of the context constraints on the particular labels, Figure 7.3 shows the confusion matrices of the local classifiers and the final HCs w.r.t. both test sets. As can be seen e.g. from the rows for books and shelves in the matrix for the indoor local classifier, it misclassifies nearly half of the books and many shelves. By adding a constraint that books are contained in shelves (weight 5.067) and that shelves contain books (weight 3.967), the number of correctly classified shelves increased from 62 to 66, and for books from 43 to 69. Similarly, considering the matrix for the outdoor local classifier, adding a constraint that windows are contained in buildings or in the upper parts of cars (weight 3.863) decreases wrong window classifications from 139 to 40.

For testing hypothesis (H7.3), we artificially decreased the quality of the local classifier by training it on a gradually shrunken training set. Notably the benefit of adding context constraints decreased with the accuracy of the local classifier, and when it was below ≈35 % for indoor resp. ≈45 % for outdoor scenes, the local classifier outperformed the HC.

Finding the optimal solution for an HC encoding for a given scene by *hexlite* usually took just a few seconds, on a Linux machine with an 3.2 GHz Intel Core i7 CPU and 8 GB RAM.

7.6 Related Work

As context information is valuable for simultaneous classification of visual objects, many approaches—mainly in *computer vision*—exploit scene information and provide either a statistical summary of the image (also called *gist*) as additional input to the classifier, or exploit relationships between particular objects in a scene (often called the *semantic context*) (Rabinovich & Belongie, 2009). Rabinovich and Belongie (2009) argue that by considering semantic context, stronger contextual constraints can be imposed (e.g. also spatial relations), and show empirically that they can greatly improve recognition performance. Most approaches using semantic context for label prediction employ some kind of *graphical model*, e.g. *conditional random fields* (Rabinovich et al., 2007) or *Markov logic networks* (Chechetka et al., 2010; Tran & Davis, 2008; Marton et al., 2009) in which the mutual influence of labelings is directly encoded by conditional probabilities. Another approach that is very effective for object classification in complex scenes (Angin & Bhargava, 2013) and for collective classification in general (Sen et al., 2008) is the *iterative classification algorithm* (*ICA*) (Sen et al., 2010), which iteratively predicts and updates the label of each object based on the current labeling.

Clearly, our approach is related to approaches that consider semantic context or use graphical models. Those above are different from ours as they usually employ

probabilistic inference methods such as *Markov chain monte carlo* and do not use combinatorial optimization techniques. In addition, often only simple relations such as the co-occurrence frequency of objects were addressed (Rabinovich et al., 2007; Angin & Bhargava, 2013). In contrast, we consider diverse relations between objects extracted from an image (e.g. their position, height and spatial relation to other objects) and they can be combined into more complex relations. Notably, *closed world reasoning* can directly be employed in our approach. In *Markov logic*, this is not straightforward since in the worst case an exponential number of *loop formulas* has to be computed in order to translate LP^{MLN} to Markov logic (Lee & Wang, 2016).

An approach similar to ours is presented in (Saathoff & Staab, 2008), where spatial context is also formalized as constraints to increase collective classification accuracy. However, *fuzzy CSPs* and *branch and bound* are used instead of probabilistic semantics, and only basic relation types are considered. From a bird's eye view, our probabilistic approach achieves a higher accuracy gain with considerably less training data, but further research (requiring an implementation of (Saathoff & Staab, 2008) and suitable benchmarks) is needed for a clear picture.

Regarding semantic representations of scene images, *scene graphs* have recently been introduced as a means for encoding the context of objects in scenes (Johnson et al., 2015). Formally, a scene graph consists of three sets that capture the semantic contents of a scene image: a set of objects, a set of object attributes, and a set of (binary) relations between objects. The objects hereby represent the vertices of a scene graph, their relations are represented by its edges and vertices are labelled with attributes. Scene graphs have attracted a lot of interest in the areas of computer vision as well as *natural language processing* in the last years and are used in a large number of different approaches as witnessed by two recent surveys (P. Xu et al., EasyChair, 2020; Agarwal, Mangal, & Vipul, 2020). They have been exploited for a variety of tasks, such as *semantic image retrieval* (Johnson et al., 2015), *visual question answering* (Tang, Zhang, Wu, Luo, & Liu, 2019; Ghosh, Burachas, Ray, & Ziskind, 2019), *image captioning* (Y. Li, Ouyang, Zhou, Wang, & Wang, 2017; X. Li & Jiang, 2019) and *image generation* (Johnson, Gupta, & Fei-Fei, 2018).

Besides the many applications of scene graphs for different downstream tasks, the foundational task explored in the literature consists in producing a visually-grounded scene graph for a given image, which is referred to as *scene graph generation* (*SGG*) (P. Xu et al., EasyChair, 2020). Many approaches for SGG jointly predict object and relation labels by passing messages between the different prediction modules along the edges of a *graph neural network*. For instance, D. Xu, Zhu, Choy, and Fei-Fei (2017) generate a scene graph using *recurrent neural networks* by iteratively passing messages between object and relation nodes to refine the respective label predictions. Y. Li et al. (2017) also exploit message passing between three semantic levels of their novel *multi-level scene description network* model, which are used for object detection, detecting relations between objects and captioning of image regions, respectively.

Similar to our approach, recognizing object and relation labels in orchestration has the advantage that detected relations can act as contextual cues to improve object recognition.

However, the primary focus of SGG is not to improve object recognition since object and relation labels are inferred simultaneously, while in our case, relation groundings are part of the input. Accordingly, the performance w.r.t. object recognition alone is often not reported for SGG, and instead of predictive accuracy, different metrics such as specific recall measures are commonly reported in this area (Agarwal et al., 2020). Moreover, usually the complete SGG-pipeline is evaluated monolithically, starting from a *region proposal network* for detecting regions likely to contain objects and resulting in a visually-grounded scene graph. This makes a direct comparison with our approach difficult.

Since approaches for SGG tackle a more difficult task than we consider in our work, i.e. detecting objects as well as relations from raw images, they normally require huge amounts of labelled training data (Agarwal et al., 2020). For instance, the *Visual Genome* dataset (Krishna et al., 2017), which is widely used for measuring performance in the literature on scene graphs, contains over 108K images of which commonly about 70 % are used for training. In contrast, one of our central goals was to improve object recognition in view of very few training data being available; and our training sets only contain 30 scene images each. In this scenario, a hybrid approach can exploit complex context constraints to correct wrong object labels, while it would be very hard to predict a complete scene graph from only 30 examples. At the same time, our context constraints can leverage the complete reasoning power of HEX-programs.

7.7 Conclusion and Outlook

In this chapter, we introduced a general framework for solving CCPs and a methodology for its application. Our tests show that classification of objects can be significantly improved by considering their semantic context. At the same time, the achievable improvement highly depends on the selected constraints and their interaction, as well as on the quality of the local classifier. If the latter labels most objects incorrectly, the context constraints intuitively lack a reasonable base for correction as wrong labels do not help to infer correct labels of other objects. Overall, we found that best HC results can be obtained when the local classifier performs reasonably well but there is still room for improvement, and when the right set of constraints is selected using a validation set.

Even though we address a specific application, our framework is applicable to a wide range of tasks. For instance, linked data (e.g. social networks or citation graphs) has been considered as a domain for collective classification (London & Getoor, 2014). Previous work in this area mostly focuses on uniform neighborhood relations, where complex reasoning over context relations is not required. However, there is much room for exploiting rich relational structures in these domains as well, similar to the one provided by spatial relations in the visual domain. For example, an external ontology that defines a number of properties of people and relations between them could be interfaced, and the transitive closure of the friend-of relation could be computed in an ASP-program.

Regarding future work, a promising direction would be to improve the learning side of our approach by utilizing more sophisticated methods. While we do not take interactions

between different context constraints into account and create context constraints manually, such constraints and their weights could also be obtained by employing advanced learning methods from the literature. On the one hand, for settings where the amount of training data is sufficiently large in order to also learn interactions between context constraints, it would be desirable to, e.g., employ the *MC-ASP* sampling method by Lee and Wang (2018), instead of using the log odds. On the other hand, the *meta-interpretive learning* approach discussed in Chapter 6 could be utilized to learn context constraints from training examples.

Moreover, as mentioned in Section 7.4, the expressiveness of HEX-programs, also allowing cyclic dependencies over external atoms, could be leveraged to a larger extent. For instance, labels which have already been fixed during the computation of an HC-solution could be used to constrain possible labels of other objects, where the set of possible labels for a respective object could be provided as input to the external classifier atom, such that only a subset of all labels is assigned non-zero probability. However, for this, questions such as how a cyclic interaction could be realized efficiently and how it would influence the probabilistic semantics need to be investigated.

Beyond feeding back context information to the local classifier for filtering possible labels, another promising future direction would be a deeper integration of the symbolic and the sub-symbolic components. Yang, Ishay, and Lee (2020) recently showed that a tight coupling of ASP with neural networks is possible by utilizing the outputs of a neural network as probability distribution over part of the atoms in an answer set program and reversely, by backpropagating a loss based on the answer set program for training the neural network.

To adopt a similar method for training the local classifier with the aid of semantic constraints in the context of our approach, however, different changes would be necessary. Firstly, a specific model type for the local classifier, e.g. a neural network, needs to be assumed which allows for propagating a loss computed based on the HEX-program. Secondly, since we currently apply the local classifier only to single objects in isolation, and during training we have the ground truth for each label available, context knowledge cannot be used to improve training of the local classifier right away. However, by classifying all objects in a scene image simultanously using a single neural network, the classifier could be trained to also take the context of objects into account. Then, analogously to the approach by Yang et al. (2020), the probability of all stable models that satisfy our context constraints could be maximized by employing *gradient propagation* w.r.t. the probability distribution over the labels returned by the neural network.

Yet, since we also have the ground truth labels available, an important question is how the different loss functions could be combined in order to improve learning in the neural network. We expect a combination to be particularly useful when certain objects in a scene image are hidden, i.e. covered by other objects, such that the neural network could learn to classify objects correctly even without any relevant object features being available. This scenario would be akin to the application of solving Sudoku puzzles as described by Yang et al. (2020), where correct values for empty cells need to be inferred based on the given values.

CHAPTER 8

Conclusion

In this chapter, we conclude this thesis by going back to our initial research questions from Section 1.3, with which we started our investigation of integrated algorithms for HEX and its application in machine learning; and we summarize the related progress that has been achieved during the doctoral research. In the second part of this chapter, we consider possible directions for future research.

8.1 Summary

In Part I, we developed several new techniques that tightly integrate different parts of HEX-evaluation, and the new methods have been incorporated into novel integrated HEX-algorithms. This research has thus advanced the state of HEX-evaluation in a number of ways.

Regarding our initial research question (RQ1), in Chapter 3, we achieved a tight integration of the search procedure employed during HEX-solving and external evaluation by extending the previous two-valued semantics of external atoms to three truth values. This enables the evaluation of external sources at any point during search under partial assignments; as a result, conflict detection and learning can be improved. Moreover, partial evaluation proved to have further benefits in that it allows to minimize learned io-nogoods by calling external sources repeatedly w.r.t. reduced input assignments, whereby our new minimization technique in fact yielded the best results in terms of solving efficiency. Subsequently, the methods for partial evaluation have also been extended to the search for unfounded sets employed by the external minimality check of HEX, as described in Chapter 4. Thus, our techniques based on partial evaluation represent considerable progress compared to traditional methods for HEX-evaluation and simultaneously laid the ground for other techniques that require on-the-fly evaluation of external sources.

With respect to research question (RQ2), i.e. the effective integration of information about external source behavior into the special minimality check of HEX, we have

demonstrated in Chapter 4 that the external source semantics can be approximated more closely by taking semantic dependencies over external atoms into account. We showed that additional dependency information can in turn be leveraged to avoid unnecessary external minimality checks and thus to reduce the running time required for checking minimality of candidate answer sets.

We can also positively answer research question (RQ3) since our experimental evaluation in Chapter 5 exhibits a significant benefit of lazy-grounding HEX-solving for grounding-intense problems. In order to answer question (RQ3), we developed a novel HEX-algorithm that exploits techniques from lazy grounding. At this, it became clear during the work on the new algorithm that further adaptations needed to be made to the previous evaluation framework, i.e. a dedicated program transformation and a new interface to external sources had to be designed. As a result, the lazy-grounding ASP-solver ALPHA can be exploited as backend solver for evaluating HEX-programs; and HEX can profit from future developments of lazy-grounding methods.

In Part II, we worked out two new use cases of the HEX-formalism in the context of machine learning. Their respective relation to machine learning differs in that the first application presented in Chapter 6 implements a logic-based machine learning approach, while the application developed in Chapter 7 uses external atoms to interface an external classification algorithm. In relation to our final research question (RQ4), we conclude that HEX-programs can be fruitfully employed also for approaches that require non-logical and sub-symbolic computations, and for implementing existing machine learning methods. However, special attention needs to be devoted to the amount of imported information that has to be considered during grounding.

While different parts of HEX-evaluation have been integrated by our new techniques, we also found that the different challenges encountered on the way were often interrelated. For instance, our results regarding the evaluation of external sources using partial assignments constituted a necessary foundation for the development of our lazy-grounding HEX-algorithm; and in order to improve the efficiency of the unfounded set search employed for the external minimality check, partial evaluation proved to be beneficial as well.

8.2 Future Work

Since the overarching theme of this work was to tightly integrate different mechanisms employed during HEX-solving, a number of new evaluation techniques have been developed for this purpose. However, there are several further ways in which these techniques could be combined, extended and exploited for different parts of HEX-evaluation in the future.

First, while we have only employed simple heuristics for deciding the frequency of external evaluations during HEX-solving, dynamic heuristics could also be used, where the frequency is adjusted according to the amount of information gained from previous calls. Second, our HEX-algorithm that exploits lazy grounding could be combined with a pre-grounding algorithm, where the respective grounding mechanisms are applied for different modules of a HEX-program based on their properties. Moreover, additional semantic dependency information, which we used for deciding the necessity of the external

minimality check, is also valuable for reducing the number of external evaluations during the model search and grounding, and could be utilized there as well.

In addition, our developed use cases exhibit the potential of HEX for applications in machine learning and for combining ASP with different paradigms. Hence, there is a lot of room for exploring the integration of other machine learning methods by means of external atoms of HEX, as well as for advancing our existing applications in different directions.

Even though grounding turned out to be a major bottleneck for our MIL-encodings from Chapter 6, we have not made use of lazy grounding for this use case yet as current solvers still lack many optimizations that are, e.g., implemented in CLINGO; and grounding of the used external atoms themselves did not pose a problem in our approach. Nevertheless, ASP-based approaches for MIL could exploit current and future advancements in lazy grounding.

Finally, we have already conducted tests related to our HC-encoding from Chapter 7 where we provided additional information computed in the ASP-part back to the external classifier. However, we found that one needs to pay particular attention to the presence of cyclic dependencies, which may result in a large number of calls to the external classifier and unsatisfactory solving performance. Despite our new methods for HEX-evaluation, and in particular partial evaluation of external atoms, this often cannot be avoided without carefully designing an encoding due to the inherently high complexity of HEX.

Bibliography

Agarwal, A., Mangal, A., & Vipul. (2020). Visual relationship detection using scene graphs: A survey. *CoRR*, *abs/2005.08045*.

Alviano, M., Dodaro, C., Leone, N., & Ricca, F. (2015). Advances in WASP. In F. Calimeri, G. Ianni, & M. Truszczynski (Eds.), *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings* (Vol. 9345, pp. 40–54). Springer.

Alviano, M., Faber, W., & Gebser, M. (2015). Rewriting recursive aggregates in answer set programming: back to monotonicity. *TPLP*, *15*(4-5), 559–573.

Angin, P., & Bhargava, B. (2013). A confidence ranked co-occurrence approach for accurate object recognition in highly complex scenes. *Journal of Internet Technology*, *14*(1), 13–19.

Antic, C., Eiter, T., & Fink, M. (2013). Hex semantics via approximation fixpoint theory. In P. Cabalar & T. C. Son (Eds.), *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings* (Vol. 8148, pp. 102–115). Springer.

Apt, K. R. (2003). *Principles of constraint programming*. Cambridge University Press.

Balduccini, M. (2009). Representing constraint satisfaction problems in answer set programming. In *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP) at ICLP*.

Balduccini, M., & Lierler, Y. (2013a). Hybrid automated reasoning tools: from black-box to clear-box integration. *CoRR*, *abs/1312.6105*.

Balduccini, M., & Lierler, Y. (2013b). Integration schemas for constraint answer set programming: a case study. *TPLP*, *13*(4-5-Online-Supplement).

Barrett, C. W., Sebastiani, R., Seshia, S. A., & Tinelli, C. (2009). Satisfiability modulo theories. In A. Biere, M. Heule, H. van Maaren, & T. Walsh (Eds.), *Handbook of Satisfiability* (Vol. 185, pp. 825–885). IOS Press.

Basol, S., Erdem, O., Fink, M., & Ianni, G. (2010). HEX programs with action atoms. In M. V. Hermenegildo & T. Schaub (Eds.), *Technical Communications of the 26th International Conference on Logic Programming, ICLP 2010, July 16-19, 2010, Edinburgh, Scotland, UK* (Vol. 7, pp. 24–33). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Brewka, G., Eiter, T., & Truszczynski, M. (2011). Answer set programming at a glance. *Commun. ACM*, *54*(12), 92–103.

Buccafurri, F., Leone, N., & Rullo, P. (2000). Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowl. Data Eng.*, *12*(5), 845–860.

Cabalar, P., Kaminski, R., Ostrowski, M., & Schaub, T. (2016). An ASP semantics for default reasoning with constraints. In S. Kambhampati (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016* (pp. 1015–1021). IJCAI/AAAI Press.

Cadoli, M., Eiter, T., & Gottlob, G. (1997). Default logic as a query language. *IEEE Trans. Knowl. Data Eng.*, *9*(3), 448–463.

Calì, A., Gottlob, G., & Pieris, A. (2012). Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, *193*, 87–128.

Calimeri, F., Cozza, S., & Ianni, G. (2007). External sources of knowledge and value invention in logic programming. *Ann. Math. Artif. Intell.*, *50*(3-4), 333–361.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Roland Kaminski, T. K., Leone, N., ... Schaub, T. (2013). *ASP-Core-2 Input Language Format.* Retrieved from `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf`

Calimeri, F., Fink, M., Germano, S., Humenberger, A., Ianni, G., Redl, C., ... Wimmer, A. (2016). Angry-hex: An artificial player for angry birds based on declarative knowledge bases. *IEEE Trans. Comput. Intellig. and AI in Games*, *8*(2), 128–139.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, *39*(3), 385–429.

Chechetka, A., Dash, D., & Philipose, M. (2010). Relational learning for collective classification of entities in images. In *Statistical Relational Artificial Intelligence, Papers from the 2010 AAAI Workshop, Atlanta, Georgia, USA, July 12, 2010* (Vol. WS-10-06). AAAI.

Cropper, A., & Muggleton, S. H. (2014). Logical minimisation of meta-rules within meta-interpretive learning. In J. Davis & J. Ramon (Eds.), *Inductive Logic Programming - 24th International Conference, ILP 2014, Nancy, France, September 14-16, 2014, Revised Selected Papers* (Vol. 9046, pp. 62–75). Springer.

Cropper, A., & Muggleton, S. H. (2015). Learning efficient logical robot strategies involving composable objects. In Q. Yang & M. J. Wooldridge (Eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (pp. 3423–3429). AAAI Press.

Cropper, A., & Muggleton, S. H. (2016a). Learning higher-order logic programs through abstraction and invention. In S. Kambhampati (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016* (pp. 1418–1424). IJCAI/AAAI Press.

Cropper, A., & Muggleton, S. H. (2016b). *Metagol system.* https://github.com/metagol/metagol. Retrieved from `https://github.com/metagol/metagol`

Cropper, A., Tamaddoni-Nezhad, A., & Muggleton, S. H. (2015). Meta-interpretive learning of data transformation programs. In K. Inoue, H. Ohwada, & A. Yamamoto

(Eds.), *Inductive Logic Programming - 25th International Conference, ILP 2015, Kyoto, Japan, August 20-22, 2015, Revised Selected Papers* (Vol. 9575, pp. 46–59). Springer.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., & Bray, C. (2004). Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV 2004* (pp. 1–22).

Dai, W., Muggleton, S., Wen, J., Tamaddoni-Nezhad, A., & Zhou, Z. (2017). Logical vision: One-shot meta-interpretive learning from real images. In N. Lachiche & C. Vrain (Eds.), *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers* (Vol. 10759, pp. 46–62). Springer.

Dao-Tran, M., Eiter, T., Fink, M., Weidinger, G., & Weinzierl, A. (2012). Omiga : An open minded grounding on-the-fly answer set solver. In L. F. del Cerro, A. Herzig, & J. Mengin (Eds.), *Logics in Artificial Intelligence - 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings* (Vol. 7519, pp. 480–483). Springer.

Dao-Tran, M., Eiter, T., & Krennwallner, T. (2009). Realizing default logic over description logic knowledge bases. In C. Sossai & G. Chemello (Eds.), *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 10th European Conference, ECSQARU 2009, Verona, Italy, July 1-3, 2009. Proceedings* (Vol. 5590, pp. 602–613). Springer.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *J. Artif. Intell. Res.*, *17*, 229–264.

Denecker, M., Marek, V., & Truszczyński, M. (2000). Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In J. Minker (Ed.), *Logic-Based Artificial Intelligence,* volume 597 of *The Springer International Series in Engineering and Computer Science* (pp. 127–144). Norwell, Massachusetts: Kluwer Academic Publishers.

Denecker, M., Marek, V. W., & Truszczynski, M. (2004). Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Inf. Comput.*, *192*(1), 84–121.

Dietterich, T. G., Domingos, P. M., Getoor, L., Muggleton, S., & Tadepalli, P. (2008). Structured machine learning: the next ten years. *Machine Learning*, *73*(1), 3–23.

Dodaro, C., Ricca, F., & Schüller, P. (2016). External propagators in WASP: preliminary report. In S. Bistarelli, A. Formisano, & M. Maratea (Eds.), *Proceedings of the 23rd RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2016 (RCRA 2016) A workshop of the XV International Conference of the Italian Association for Artificial Intelligence (AI*IA 2016), Genova, Italy, November 28, 2016.* (Vol. 1745, pp. 1–9). CEUR-WS.org.

Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., & Schaub, T. (2008). Conflict-driven disjunctive answer set solving. In G. Brewka & J. Lang (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the*

Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008 (pp. 422–432). AAAI Press.

Duggan, J. (2007). A systematic approach to the construction of non-empty choice sets. *Social Choice and Welfare*, *28*(3), 491–506.

Dutertre, B., & de Moura, L. M. (2006). A fast linear-arithmetic solver for DPLL(T). In T. Ball & R. B. Jones (Eds.), *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings* (Vol. 4144, pp. 81–94). Springer.

Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Redl, C., & Schüller, P. (2016). A model building framework for answer set programming with external computations. *TPLP*, *16*(4), 418–464.

Eiter, T., Fink, M., & Krennwallner, T. (2009). Decomposition of declarative knowledge bases with external functions. In C. Boutilier (Ed.), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009* (pp. 752–758).

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2012). Conflict-driven ASP solving with external sources. *TPLP*, *12*(4-5), 659–679.

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2013). Hex-programs with existential quantification. In M. Hanus & R. Rocha (Eds.), *Declarative Programming and Knowledge Management - Declarative Programming Days, KDPD 2013, Unifying INAP, WFLP, and WLP, Kiel, Germany, September 11-13, 2013, Revised Selected Papers* (Vol. 8439, pp. 99–117). Springer.

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2016). Domain expansion for ASP-programs with external sources. *Artif. Intell.*, *233*, 84–121.

Eiter, T., Fink, M., Krennwallner, T., Redl, C., & Schüller, P. (2014). Efficient hex-program evaluation based on unfounded sets. *J. Artif. Intell. Res.*, *49*, 269–321.

Eiter, T., Fink, M., Redl, C., & Stepanova, D. (2014). Exploiting support sets for answer set programs with external evaluations. In C. E. Brodley & P. Stone (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.* (pp. 1041–1048). AAAI Press.

Eiter, T., Fink, M., & Stepanova, D. (2016). Data repair of inconsistent nonmonotonic description logic programs. *Artif. Intell.*, *239*, 7–53.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008). Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, *172*(12-13), 1495–1539.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005a). NLP-DL: A KR system for coupling nonmonotonic logic programs with description logics. In R. Mizoguchi (Ed.), *Poster & Demonstration Proceedings of the 4th International Semantic Web Conference (ISWC 2005)* (p. PID 67). (System poster)

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005b). A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In L. P. Kaelbling & A. Saffiotti (Eds.), *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK,*

July 30 - August 5, 2005 (pp. 90–96). Professional Book Center.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2006). Effective integration of declarative rules with external evaluations for semantic-web reasoning. In Y. Sure & J. Domingue (Eds.), *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings* (Vol. 4011, pp. 273–287). Springer.

Eiter, T., & Kaminski, T. (2016). Exploiting contextual knowledge for hybrid classification of visual objects. In L. Michael & A. C. Kakas (Eds.), *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings* (Vol. 10021, pp. 223–239).

Eiter, T., & Kaminski, T. (2019). Pruning external minimality checking for ASP using semantic dependencies. In M. Balduccini, Y. Lierler, & S. Woltran (Eds.), *Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings* (Vol. 11481, pp. 326–339). Springer.

Eiter, T., Kaminski, T., Redl, C., Schüller, P., & Weinzierl, A. (2017). Answer set programming with external source access. In G. Ianni et al. (Eds.), *Reasoning Web. Semantic Interoperability on the Web - 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures* (Vol. 10370, pp. 204–275). Springer.

Eiter, T., Kaminski, T., Redl, C., & Weinzierl, A. (2016). Exploiting partial assignments for efficient evaluation of answer set programs with external source access. In S. Kambhampati (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016* (pp. 1058–1065). IJCAI/AAAI Press.

Eiter, T., Kaminski, T., Redl, C., & Weinzierl, A. (2018). Exploiting partial assignments for efficient evaluation of answer set programs with external source access. *J. Artif. Intell. Res.*, *62*, 665–727.

Eiter, T., Kaminski, T., & Weinzierl, A. (2017). Lazy-grounding for answer set programs with external source access. In C. Sierra (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017* (pp. 1015–1022). ijcai.org.

Eiter, T., Krennwallner, T., & Redl, C. (2011). Hex-programs with nested program calls. In H. Tompits et al. (Eds.), *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers* (Vol. 7773, pp. 269–278). Springer.

Eiter, T., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2004). Combining answer set programming with description logics for the semantic web. In D. Dubois, C. A. Welty, & M. Williams (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004* (pp. 141–151). AAAI Press.

Eiter, T., Redl, C., & Schüller, P. (2016). Problem solving using the HEX family. In

C. Beierle, G. Brewka, & M. Thimm (Eds.), *Computational Models of Rationality, Essays dedicated to Gabriele Kern-Isberner on the occasion of her 60th birthday* (pp. 150–174). College Publications.

Erdem, E., Gelfond, M., & Leone, N. (2016). Applications of answer set programming. *AI Magazine*, *37*(3), 53–68.

Faber, W. (2005). Unfounded sets for disjunctive logic programs with arbitrary aggregates. In C. Baral, G. Greco, N. Leone, & G. Terracina (Eds.), *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings* (Vol. 3662, pp. 40–52). Springer.

Faber, W., Pfeifer, G., & Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, *175*(1), 278–298.

Farquhar, C., Grov, G., Cropper, A., Muggleton, S., & Bundy, A. (2015). Typed meta-interpretive learning for proof strategies. In K. Inoue, H. Ohwada, & A. Yamamoto (Eds.), *Late Breaking Papers of the 25th International Conference on Inductive Logic Programming, Kyoto University, Kyoto, Japan, August 20th to 22nd, 2015.* (Vol. 1636, pp. 17–32). CEUR-WS.org.

Galleguillos, C., Rabinovich, A., & Belongie, S. J. (2008). Object categorization using co-occurrence, location and appearance. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE Computer Society.

Gebser, M., Grote, T., Kaminski, R., & Schaub, T. (2011). Reactive answer set programming. In J. P. Delgrande & W. Faber (Eds.), *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings* (Vol. 6645, pp. 54–66). Springer.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Thiele, S. (2008). Engineering an incremental ASP solver. In M. G. de la Banda & E. Pontelli (Eds.), *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings* (Vol. 5366, pp. 190–205). Springer.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Wanko, P. (2016). Theory solving made easy with Clingo 5. In M. Carro, A. King, N. Saeedloei, & M. D. Vos (Eds.), *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA* (Vol. 52, pp. 2:1–2:15). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., & Schaub, T. (2015). Progress in clasp series 3. In F. Calimeri, G. Ianni, & M. Truszczynski (Eds.), *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings* (Vol. 9345, pp. 368–383). Springer.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). Clingo = ASP + control: Preliminary report. *CoRR*, *abs/1405.3694*.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2019). Multi-shot ASP solving with clingo. *TPLP*, *19*(1), 27–82.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. T.

(2011). Potassco: The potsdam answer set solving collection. *AI Commun.*, *24*(2), 107–124.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). Conflict-driven answer set enumeration. In C. Baral, G. Brewka, & J. S. Schlipf (Eds.), *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings* (Vol. 4483, pp. 136–148). Springer.

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, *187*, 52–89.

Gebser, M., Kaufmann, B., & Schaub, T. (2013). Advanced conflict-driven disjunctive answer set solving. In F. Rossi (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013* (pp. 912–918). IJCAI/AAAI.

Gebser, M., Ostrowski, M., & Schaub, T. (2009). Constraint answer set solving. In P. M. Hill & D. S. Warren (Eds.), *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings* (Vol. 5649, pp. 235–249). Springer.

Gebser, M., Ryabokon, A., & Schenner, G. (2015). Combining heuristics for configuration problems using answer set programming. In F. Calimeri, G. Ianni, & M. Truszczynski (Eds.), *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings* (Vol. 9345, pp. 384–397). Springer.

Gebser, M., & Schaub, T. (2016). Modeling and language extensions. *AI Magazine*, *37*(3), 33–44.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, *9*(3/4), 365–386.

Gent, I. P., Miguel, I., & Moore, N. C. A. (2010). Lazy explanations for constraint propagators. In M. Carro & R. Peña (Eds.), *Practical Aspects of Declarative Languages, 12th International Symposium, PADL 2010, Madrid, Spain, January 18-19, 2010. Proceedings* (Vol. 5937, pp. 217–233). Springer.

Getoor, L. (2007). *Introduction to statistical relational learning.* MIT press.

Ghosh, S., Burachas, G., Ray, A., & Ziskind, A. (2019). Generating natural language explanations for visual question answering using scene graphs and visual attention. *CoRR*, *abs/1902.05715*.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artif. Intell.*, *153*(1-2), 49–104.

Heflin, J., & Munoz-Avila, H. (2002). LCW-Based Agent Planning for the Semantic Web. In A. Pease (Ed.), *Ontologies and the Semantic Web* (pp. 63–70). Menlo Park, CA: AAAI Press.

Hoehndorf, R., Loebe, F., Kelso, J., & Herre, H. (2007). Representing default knowledge in biomedical ontologies: application to the integration of anatomy and phenotype ontologies. *BMC Bioinformatics*, *8*.

HTCondor Website. (2018). `http://research.cs.wisc.edu/htcondor`. (Accessed: 2018-06-27)

Janhunen, T., Liu, G., & Niemelä, I. (2011). Tight integration of non-ground answer set programming and satisfiability modulo theories. In P. Cabalar, D. Mitchell, D. Pearce, & E. Ternovska (Eds.), *Informal Proceedings of the 1st Workshop on Grounding and Transformations for Theories with Variables (GTTV'11), LPNMR, Vancouver, BC, Canada May 16th, 2011* (p. 1-14).

Johnson, J., Gupta, A., & Fei-Fei, L. (2018). Image generation from scene graphs. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (pp. 1219–1228). IEEE Computer Society.

Johnson, J., Krishna, R., Stark, M., Li, L., Shamma, D. A., Bernstein, M. S., & Li, F. (2015). Image retrieval using scene graphs. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015* (pp. 3668–3678). IEEE Computer Society.

Junker, U. (2004). QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In D. L. McGuinness & G. Ferguson (Eds.), *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA* (pp. 167–172). AAAI Press / The MIT Press.

Kakas, A. C., Kowalski, R. A., & Toni, F. (1992). Abductive logic programming. *J. Log. Comput.*, *2*(6), 719–770.

Kalinowski, T., Narodytska, N., Walsh, T., & Xia, L. (2013). Strategic behavior when allocating indivisible goods sequentially. In M. desJardins & M. L. Littman (Eds.), *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press.

Kaminski, T., Eiter, T., & Inoue, K. (2018a). *Efficiently encoding meta-interpretive learning by answer set programming (work in progress).* http://ilp2018.unife.it/wp-content/uploads/2018/08/Efficiently-Encoding-Meta-Interpretive-Learning-by-Answer-Set-Programming.pdf.

Kaminski, T., Eiter, T., & Inoue, K. (2018b). Exploiting answer set programming with external sources for meta-interpretive learning. *TPLP*, *18*(3-4), 571–588.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA* (pp. 85–103). Plenum Press, New York.

Kaufmann, B., Leone, N., Perri, S., & Schaub, T. (2016). Grounding and solving in answer set programming. *AI Magazine*, *37*(3), 25–32.

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., ... Fei-Fei, L. (2017). Visual genome: Connecting language and vision using crowdsourced dense image annotations. *Int. J. Comput. Vis.*, *123*(1), 32–73.

Lahiri, S. K., Nieuwenhuis, R., & Oliveras, A. (2006). SMT techniques for fast predicate abstraction. In T. Ball & R. B. Jones (Eds.), *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings* (Vol. 4144, pp. 424–437). Springer.

Larson, J., & Michalski, R. S. (1977). Inductive inference of VL decision rules. *SIGART Newsletter*, *63*, 38–44.

Law, M., Russo, A., & Broda, K. (2014). Inductive learning of answer set programs. In E. Fermé & J. Leite (Eds.), *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings* (Vol. 8761, pp. 311–325). Springer.

Lee, J., & Meng, Y. (2013). Answer set programming modulo theories and reasoning about continuous changes. In F. Rossi (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013* (pp. 990–996). IJCAI/AAAI.

Lee, J., & Wang, Y. (2016). Weighted rules under the stable model semantics. In C. Baral, J. P. Delgrande, & F. Wolter (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.* (pp. 145–154). AAAI Press.

Lee, J., & Wang, Y. (2018). Weight learning in a probabilistic extension of answer set programs. In M. Thielscher, F. Toni, & F. Wolter (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018.* (pp. 22–31). AAAI Press.

Lefèvre, C., Béatrix, C., Stéphan, I., & Garcia, L. (2017). Asperix, a first-order forward chaining approach for answer set computing. *TPLP*, *17*(3), 266–310.

Lefèvre, C., & Nicolas, P. (2009a). A first order forward chaining approach for answer set computing. In E. Erdem, F. Lin, & T. Schaub (Eds.), *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings* (Vol. 5753, pp. 196–208). Springer.

Lefèvre, C., & Nicolas, P. (2009b). The first version of a new ASP solver : Asperix. In E. Erdem, F. Lin, & T. Schaub (Eds.), *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings* (Vol. 5753, pp. 522–527). Springer.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, *7*(3), 499–562.

Leone, N., Rullo, P., & Scarcello, F. (1997). Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Inf. Comput.*, *135*(2), 69–112.

Li, X., & Jiang, S. (2019). Know more say less: Image captioning based on scene graphs. *IEEE Trans. Multim.*, *21*(8), 2117–2130.

Li, Y., Ouyang, W., Zhou, B., Wang, K., & Wang, X. (2017). Scene graph generation from objects, phrases and region captions. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017* (pp. 1270–1279). IEEE Computer Society.

Lierler, Y. (2014). Relating constraint answer set programming languages and algorithms. *Artif. Intell.*, *207*, 1–22.

Lierler, Y., Maratea, M., & Ricca, F. (2016). Systems, engineering environments, and

competitions. *AI Magazine*, *37*(3), 45–52.

Lifschitz, V., & Turner, H. (1994). Splitting a logic program. In P. V. Hentenryck (Ed.), *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994* (pp. 23–37). MIT Press.

Lin, D., Dechter, E., Ellis, K., Tenenbaum, J. B., & Muggleton, S. (2014). Bias reformulation for one-shot function induction. In T. Schaub, G. Friedrich, & B. O'Sullivan (Eds.), *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)* (Vol. 263, pp. 525–530). IOS Press.

Liu, G., Janhunen, T., & Niemelä, I. (2012). Answer set programming via mixed integer programming. In G. Brewka, T. Eiter, & S. A. McIlraith (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012.* AAAI Press.

London, B., & Getoor, L. (2014). Collective classification of network data. In C. C. Aggarwal (Ed.), *Data Classification: Algorithms and Applications* (pp. 399–416). CRC Press.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision, Kerkyra, Corfu, Greece, September 20-25, 1999* (pp. 1150–1157). IEEE Computer Society.

Manquinho, V. M., & Silva, J. P. M. (2005). Effective lower bounding techniques for pseudo-boolean optimization. In *2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005), 7-11 March 2005, Munich, Germany* (pp. 660–665). IEEE Computer Society.

Marques-Silva, J. P., Lynce, I., & Malik, S. (2009). Conflict-driven clause learning SAT solvers. In A. Biere, M. Heule, H. van Maaren, & T. Walsh (Eds.), *Handbook of Satisfiability* (Vol. 185, pp. 131–153). IOS Press.

Marton, Z. C., Rusu, R. B., Jain, D., Klank, U., & Beetz, M. (2009). Probabilistic categorization of kitchen objects in table settings with a composite sensor. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 11-15, 2009, St. Louis, MO, USA* (pp. 4777–4784). IEEE.

Mellarkod, V. S., Gelfond, M., & Zhang, Y. (2008). Integrating answer set programming and constraint logic programming. *Ann. Math. Artif. Intell.*, *53*(1-4), 251–287.

Merriam-Webster Website. (2018). `https://www.merriam-webster.com/thesaurus`. (Accessed: 2018-06-27)

Michie, D., Muggleton, S., Page, D., & Srinivasan, A. (1994). *To the international computing community: A new east-west challenge* (Tech. Rep.). Oxford University Computing laboratory, UK.

Muggleton, S. H., Lin, D., Pahlavi, N., & Tamaddoni-Nezhad, A. (2014). Meta-interpretive learning: application to grammatical inference. *Machine Learning*, *94*(1), 25–49.

Muggleton, S. H., Lin, D., & Tamaddoni-Nezhad, A. (2015). Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning*, *100*(1), 49–73.

Nieuwenhuis, R., & Oliveras, A. (2005). DPLL(T) with exhaustive theory propagation and its application to difference logic. In K. Etessami & S. K. Rajamani (Eds.), *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings* (Vol. 3576, pp. 321–334). Springer.

Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2006). Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, *53*(6), 937–977.

Oikarinen, E., & Janhunen, T. (2006). Modular equivalence for normal logic programs. In G. Brewka, S. Coradeschi, A. Perini, & P. Traverso (Eds.), *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings* (Vol. 141, pp. 412–416). IOS Press.

Ostrowski, M., & Schaub, T. (2012). ASP modulo CSP: the clingcon system. *TPLP*, *12*(4-5), 485–503.

Otero, R. P. (2001). Induction of stable models. In C. Rouveirol & M. Sebag (Eds.), *Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings, series = Lecture Notes in Computer Science* (Vol. 2157, pp. 193–205). Springer.

Palù, A. D., Dovier, A., Pontelli, E., & Rossi, G. (2009). GASP: answer set programming with lazy grounding. *Fundam. Inform.*, *96*(3), 297–322.

Papadimitriou, C., & Yannakakis, M. (1985). A note on succinct representations of graphs. *Inform. Comput.*, *71*, 181–185.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, *12*, 2825–2830.

Pelov, N., Denecker, M., & Bruynooghe, M. (2004). Partial stable models for logic programs with aggregates. In V. Lifschitz & I. Niemelä (Eds.), *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings* (Vol. 2923, pp. 207–219). Springer.

Rabinovich, A., & Belongie, S. J. (2009). Scenes vs. objects: A comparative study of two approaches to context based recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2009, Miami, FL, USA, 20-25 June, 2009* (pp. 92–99). IEEE Computer Society.

Rabinovich, A., Vedaldi, A., Galleguillos, C., Wiewiora, E., & Belongie, S. J. (2007). Objects in context. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007* (pp. 1–8). IEEE Computer Society.

Randell, D. A., Cui, Z., & Cohn, A. G. (1992). A spatial logic based on regions and connection. In B. Nebel, C. Rich, & W. R. Swartout (Eds.), *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992.* (pp. 165–176). Morgan Kaufmann.

Ray, O. (2009). Nonmonotonic abductive inductive learning. *J. Applied Logic*, *7*(3), 329–340.

Redl, C. (2014). *Answer set programming with external sources: Algorithms and efficient evaluation* (Unpublished doctoral dissertation). Vienna University of Technology.

Redl, C. (2016). The DLVHEX system for knowledge representation: Recent advances (system description). *CoRR*, *abs/1607.08864*.

Redl, C. (2017a). Conflict-driven ASP solving with external sources and program splits. In C. Sierra (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017* (pp. 1239–1246). ijcai.org.

Redl, C. (2017b). Efficient evaluation of answer set programs with external sources based on external source inlining. In S. P. Singh & S. Markovitch (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.* (pp. 1222–1228). AAAI Press.

Redl, C., Eiter, T., & Krennwallner, T. (2011). Declarative belief set merging using merging plans. In R. Rocha & J. Launchbury (Eds.), *Practical Aspects of Declarative Languages - 13th International Symposium, PADL 2011, Austin, TX, USA, January 24-25, 2011. Proceedings* (Vol. 6539, pp. 99–114). Springer.

Reiter, R., & de Kleer, J. (1987). Foundations of assumption-based truth maintenance systems: Preliminary report. In K. D. Forbus & H. E. Shrobe (Eds.), *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, USA, July 1987.* (pp. 183–189). Morgan Kaufmann.

Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., & Leone, N. (2009). Ontodlv: An asp-based system for enterprise ontologies. *J. Log. Comput.*, *19*(4), 643–670.

Richardson, M., & Domingos, P. M. (2006). Markov logic networks. *Machine Learning*, *62*(1-2), 107–136.

Rosis, A. D., Eiter, T., Redl, C., & Ricca, F. (n.d., August). Constraint answer set programming based on HEX-programs. In *Eighth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2015), August 31, 2015, Cork, Ireland.*

Roussel, O., & Manquinho, V. M. (2009). Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, & T. Walsh (Eds.), *Handbook of Satisfiability* (Vol. 185, pp. 695–733). IOS Press.

Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, *77*(1-3), 157–173.

Saathoff, C., & Staab, S. (2008). Exploiting spatial context in image region labelling using fuzzy constraint reasoning. In *Ninth International Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS 2008, Klagenfurt, Austria, May 7-9, 2008* (pp. 16–19). IEEE Computer Society.

Schüller, P. (2012). *Inconsistency in multi-context systems: Analysis and efficient evaluation* (Unpublished doctoral dissertation). Vienna University of Technology,

Vienna, Austria.

Schüller, P. (2019). The hexlite solver - lightweight and efficient evaluation of HEX programs. In F. Calimeri, N. Leone, & M. Manna (Eds.), *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings* (Vol. 11468, pp. 593–607). Springer.

Sen, P., Namata, G., Bilgic, M., & Getoor, L. (2010). Collective classification. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 189–193). Springer.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., & Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, *29*(3), 93–106.

Shchekotykhin, K. M., Jannach, D., & Schmitz, T. (2015). MergeXplain: Fast computation of multiple conflicts for diagnosis. In Q. Yang & M. J. Wooldridge (Eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (pp. 3221–3228). AAAI Press.

Shen, Y., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., & Deng, J. (2014). FLP answer set semantics without circular justifications for general logic programs. *Artif. Intell.*, *213*, 1–41.

Simons, P., Niemelä, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artif. Intell.*, *138*(1-2), 181–234.

Soininen, T., Niemelä, I., Tiihonen, J., & Sulonen, R. (2001). Representing configuration knowledge with weight constraint rules. In A. Provetti & T. C. Son (Eds.), *Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning, Proceedings of the 1st Intl. ASP'01 Workshop, Stanford, CA, USA, March 26-28, 2001.*

Sörensson, N., & Biere, A. (2009). Minimizing learned clauses. In O. Kullmann (Ed.), *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings* (Vol. 5584, pp. 237–243). Springer.

Strobl, C. (2008). Dimensionally extended nine-intersection model (DE-9IM). In S. Shekhar & H. Xiong (Eds.), *Encyclopedia of GIS.* (pp. 240–245). Springer.

Susman, B., & Lierler, Y. (2016). Smt-based constraint answer set solver EZSMT (system description). In M. Carro, A. King, N. Saeedloei, & M. D. Vos (Eds.), *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA* (Vol. 52, pp. 1:1–1:15). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Tamaddoni-Nezhad, A., Bohan, D., Raybould, A., & Muggleton, S. (2014). Towards machine learning of predictive models from ecological data. In J. Davis & J. Ramon (Eds.), *Inductive Logic Programming - 24th International Conference, ILP 2014, Nancy, France, September 14-16, 2014, Revised Selected Papers* (Vol. 9046, pp. 154–167). Springer.

Tang, K., Zhang, H., Wu, B., Luo, W., & Liu, W. (2019). Learning to compose dynamic tree structures for visual contexts. In *IEEE Conference on Computer Vision and*

Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019 (pp. 6619–6628). Computer Vision Foundation / IEEE.

Taupe, R., Weinzierl, A., & Friedrich, G. (2019). Degrees of laziness in grounding - effects of lazy-grounding strategies on ASP solving. In M. Balduccini, Y. Lierler, & S. Woltran (Eds.), *Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings* (Vol. 11481, pp. 298–311). Springer.

Terracina, G., Francesco, E. D., Panetta, C., & Leone, N. (2008). Enhancing a DLP system for advanced database applications. In D. Calvanese & G. Lausen (Eds.), *Web Reasoning and Rule Systems, Second International Conference, RR 2008, Karlsruhe, Germany, October 31-November 1, 2008. Proceedings* (Vol. 5341, pp. 119–134). Springer.

Terracina, G., Leone, N., Lio, V., & Panetta, C. (2008). Experimenting with recursive queries in database and logic programming systems. *TPLP*, *8*(2), 129–165.

Tran, S. D., & Davis, L. S. (2008). Event modeling and recognition using markov logic networks. In D. A. Forsyth, P. H. S. Torr, & A. Zisserman (Eds.), *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part II* (Vol. 5303, pp. 610–623). Springer.

Valiant, L. G. (1984). A theory of the learnable. *Commun. ACM*, *27*(11), 1134–1142.

Weinzierl, A. (2017). Blending lazy-grounding and CDNL search for answer-set solving. In M. Balduccini & T. Janhunen (Eds.), *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings* (Vol. 10377, pp. 191–204). Springer.

Xu, D., Zhu, Y., Choy, C. B., & Fei-Fei, L. (2017). Scene graph generation by iterative message passing. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017* (pp. 3097–3106). IEEE Computer Society.

Xu, P., Chang, X., Guo, L., Huang, P.-Y., Chen, X., & Hauptmann, A. (EasyChair, 2020). *A survey of scene graph: Generation and application.* EasyChair Preprint no. 3385.

Yang, Z., Ishay, A., & Lee, J. (2020). NeurASP: Embracing neural networks into answer set programming. In C. Bessiere (Ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020* (pp. 1755–1762). ijcai.org.

Zhang, H., Fritts, J. E., & Goldman, S. A. (2008). Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, *110*(2), 260–280.

Zirtiloglu, H., & Yolum, P. (2008). Ranking semantic information for e-government: complaints management. In A. Duke, M. Hepp, K. Bontcheva, & M. B. Vilain (Eds.), *Proceedings of the First International Workshop on Ontology-supported Business Intelligence, OBI 2008, Karlsruhe, Germany, October 27, 2008* (Vol. 308, p. 5). ACM.

APPENDIX A

Proofs

This appendix contains the proofs for the complexity results from Chapter 4 and the proofs for soundness and completeness of Algorithm 5.1 from Chapter 5.

A.1 Proofs for Complexity Results from Section 4.2

We thank Thomas Eiter for contributing the following complexity results to our joint work in (Eiter & Kaminski, 2019).

Proposition 4.8. Checking faithfulness of a given set $D \subseteq dep(\&g[\vec{p}])$ is co-NEXP-complete in general, and co-NP-complete for fixed predicate arities.

Proof. Membership in co-NEXP respectively co-NP can be shown by a guess and check algorithm for disproving faithfulness of D: to this end, we can guess assignments $\mathbf{A}$, $\mathbf{A}'$ to the input predicates $\vec{p}$, and an output tuple $\vec{c}$, such that $\mathbf{A}$, $\mathbf{A}'$ coincide on all atoms in $comp(D, \&g[\vec{p}](\vec{c}))$ and $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$.

The guess for $\mathbf{A}$, $\mathbf{A}'$ and $\vec{c}$ is in the general case of exponential size in the input, while it has polynomial size if the arity of the predicates p_i in $\vec{p}$ is bounded by a constant, as only a polynomial number of atoms in the size of the set of constants is possible.

In order to verify the guess, one first computes the set $comp(D, \&g[\vec{p}](\vec{c}))$; this is feasible in exponential (resp. polynomial) time in the size of the input. Checking whether $\mathbf{A}$, $\mathbf{A}'$ coincide on $comp(D, \&g[\vec{p}](\vec{c}))$ and computing $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$ is both feasible in exponential (resp. polynomial) time in the size of the input; overall, this means that disproving faithfulness of D is in NEXP, from which the claimed upper bound follows.

For the hardness parts, we provide a reduction from the complement of Graph 3-Colorability, which is a canonical NP-complete problem. For succinct input representation, this problem is well-known to be NEXP-complete (Papadimitriou & Yannakakis, 1985); that is, the graph $G = (V, E)$ is not given in the usual form (say, by the adjacency matrix

of its vertices), but by a Boolean circuit C_G with $2n$ input bits $b_1, \ldots, b_{2n}$ such that $v = b_1, \ldots, b_n$ and $v' = b_{n+1}, \ldots, b_{2n}$ represent nodes (in binary coding) and the circuit C_G outputs 1 for v, v' if and only if there is an edge between v and v'. (Note that C_G can be exponentially more succinct than the usual representation of G, which intuitively explains the exponential complexity blowup.)

We reduce 3-Colorability to non-faithfulness checking as follows. We use an external predicate $\&col_G[r, g, b]/1$ that has three n-ary input predicates $\vec{p} = r, g, b$, where each ground atom $r(b_1, \ldots, b_n)$ with all $b_i \in \{0, 1\}$ means that the vertex $v = b_1, \ldots, b_n$ is colored red (analogous for b and g). The function $f_{\&col_G}(\mathbf{A}, \vec{p}, 1)$ evaluates to $\mathbf{T}$ iff $\vec{p} = r, g, b$ as given in $\mathbf{A}$ constitutes a legal 3-coloring for the graph G (this property can be easily checked in polynomial time in the size of $\mathbf{A}$ and C_G[1]), and $f_{\&col_G}(\mathbf{A}, \vec{p}, 0)$ takes the opposite value.

The set of io-dependencies is $D = \{\delta_1, \delta_2, \delta_3\}$, where $\delta_i = \langle i, 1 : \{0\}, 1 : 0\rangle$, for $i = 1, 2, 3$. Intuitively for output 0, only the color assignment to the vertices in $V_0 = \{v = 0, b_2, \ldots, b_n \mid b_i \in \{0, 1\},\ 2 \leq i \leq n\}$ matters, i.e., those with a leading 0 in the binary representation, as $comp(D, \&col_G[r, g, b](0)) = \{r(v), g(v), b(v) \mid v \in V_0\}$. However, to be sure that any 3-coloring for these vertices (which might be feasible) can not be extended to a 3-coloring of all vertices, it must hold that the full graph is not 3-colorable.

Formally, we claim that D is faithful w.r.t. $\&col_G[r, g, b]$ iff the graph G is not 3-colorable.

($\Leftarrow$) Assume that G is not 3-colorable. Then, for every assignment $\mathbf{A}$, we have that $f_{\&col_G}(\mathbf{A}, r, g, b, 1) = \mathbf{F}$ and $f_{\&col_G}(\mathbf{A}, r, g, b, 0) = \mathbf{T}$; hence D is clearly faithful, as no counterexample to the faithfulness condition is possible.

($\Rightarrow$) Assume that G is 3-colorable. Then there exists an assignment $\mathbf{A}$ such that $f_{\&col_G}(\mathbf{A}, r, g, b, 1) = \mathbf{T}$ holds, which means $f_{\&col_G}(\mathbf{A}, r, g, b, 0) = \mathbf{F}$. However, for the assignment $\mathbf{A}'$ that coincides with $\mathbf{A}$ on V_0 and assigns no color to the remaining vertices $V \setminus V_0$ (where without loss of generality, some such vertex exists), we have $f_{\&col_G}(\mathbf{A}, r, g, b, 1) = \mathbf{F}$ and thus $f_{\&col_G}(\mathbf{A}, r, g, b, 0) = \mathbf{T}$; hence, D is not faithful.

This shows the co-NEXP-hardness of the problem in the general case. In the case of bounded predicate arities, we use the nodes V as constants, and $r(v)$ expresses that vertex v is colored red (analogous for b and g). Then, assuming that only 0 and 1 can be in the output of $\&col_G[r, g, b]$, i.e., $f_{\&col_G}(\mathbf{A}, r, g, b, c) = \mathbf{F}$ for every $c \neq 0, 1$, we similarly conclude that D is not faithful w.r.t. $\&col_G[r, g, b]$ iff G is 3-colorable; this proves co-NP-hardness.

We remark that the three predicates r, g, b in the above construction can be replaced by a single predicate p using reification (i.e., represent $r(\vec{d})$ by $p(r, \vec{d})$); moreover, in the case of bounded predicate arity, tuples (r, v), etc can be viewed as constants of the domain. Consequently, the hardness parts hold for a single input predicate, which moreover for bounded arities is unary. □

[1] Technically, the code for this check can be realized as a Turing machine M_G that cycles through all pairs v, v' of nodes and simulates for each pair the evaluation of C_G and checks whether v, v' are colored differently if an edge between them exists. The machine M_G is constructible in polynomial time from C_G, and it runs on input $\mathbf{A}$ in time polynomial in the size of $\mathbf{A}$ and C_G. As such, M_G constitutes the implementation of $f_{\&col_G}$.

Corollary 4.2. If all $p_i \in \vec{p}$ for $\&g[\vec{p}]$ are monotonic and $|comp(D, \&g[\vec{p}](\vec{c}))|$ is bounded, then checking faithfulness is polynomial for fixed predicate arities.

Proof. Indeed, if we have that $|comp(D, \&g[\vec{p}](\vec{c}))| \leq k$ for a constant k, then after computing $comp(D, \&g[\vec{p}](\vec{c}))$, which can for bounded predicate arities be done in polynomial time by cycling through all (polynomially many) atoms, we need to consider by Proposition 4.9 for each assignment on $comp(D, \&g[\vec{p}](\vec{c}))$ only the two specific assignments that set all other ground atoms to true resp. all to false; thus, we need to evaluate and compare $2{\cdot}2^k = 2^{k+1}$ function calls, which is feasible in polynomial time. □

A.2 Proofs for Soundness and Completeness of Algorithm 5.1

In this section, we work out the proof of Proposition 5.4, as well as the proof for the main result stated in Theorem 5.2, i.e. soundness and completeness of Algorithm 5.1. To this end, we proceed in several steps, introducing two auxiliary lemmas used in the proofs of Proposition 5.4 and Theorem 5.2.

The overall structure of the proof is as follows:

- First, we characterize assignments $b(\mathbf{A}, \mathcal{A})$ in Definition A.1 and show, in the proof of Lemma A.1, that all outputs of Algorithm 5.1 correspond to some $b(\mathbf{A}, \mathcal{A})$.
- Proposition 5.4 then represents the core of the soundness and completeness result, and we show in the corresponding proof that the sets $b(\mathbf{A}, \mathcal{A})$ encode exactly the answer sets of the respective HEX-program given as input to Algorithm 5.1. For this, we utilize the *Splitting Theorem* from (Eiter, Fink, Ianni, et al., 2016) and show both directions by an induction proof.
- Accordingly, Theorem 5.2 follows directly from Lemmas A.1 and A.2.
- Finally, based on the results from Lemmas A.1 and A.2, Proposition 5.4 can be proven by showing that given a complete assignment $\mathbf{A}$, $b(\mathbf{A}, \mathcal{A})$ is an answer set of $\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$ if and only if $\mathbf{A}$ is an answer set of Π.

Definition A.1. *Given a* HEX*-program* Π*, an input-safe domain* $\mathcal{A}$ *of* Π *and a complete assignment* $\mathbf{A}$ *over the set of atoms occurring in* $grnd(\Pi)$*, we define* $b(\mathbf{A}, \mathcal{A}) = \hat{\mathbf{D}}$*, i.e.* $b(\mathbf{A}, \mathcal{A})$ *is the completion of* $\mathbf{D}$ *w.r.t.* $\mathcal{HB}$*, where* $\mathbf{D} = \{\mathbf{T}p_d(\vec{X}) \mid p(\vec{X}) \in \mathcal{A}\} \cup \{\mathbf{T}\overline{p}(\vec{X}) \mid p(\vec{X}) \in \mathcal{A}, \mathbf{F}p(\vec{X}) \in \mathbf{A}\} \cup \{\mathbf{T}p(\vec{X}) \in \mathbf{A}\} \cup \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \mid f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}, \&g[\vec{p}](\vec{c})$ *is a ground external atom in* $grnd(\Pi)\}$.

Lemma A.1. *Let* Π *be a* HEX*-program,* $\mathcal{A}$ *an input-safe domain of* Π*, and* $\mathsf{res}(\alpha(\Pi, \mathcal{A}))$ *the set returned by Algorithm 5.1 for input* $\alpha(\Pi, \mathcal{A})$*. Then, every* $\mathbf{A} \in \mathsf{res}(\alpha(\Pi, \mathcal{A}))$ *is such that* $\mathbf{A} = b(\mathbf{A}', \mathcal{A})$ *for some complete assignment* $\mathbf{A}'$ *that assigns* $\mathbf{T}$ *only to atoms occurring in* $grnd(\Pi)$.

Proof. Let Π be a HEX-program, $\mathcal{A}$ an input-safe domain of Π, and $\mathsf{res}(\alpha(\Pi,\mathcal{A}))$ the set returned by Algorithm 5.1 for input $\alpha(\Pi,\mathcal{A})$. We need to show:

(*) every $\mathbf{A} \in \mathsf{res}(\alpha(\Pi,\mathcal{A}))$ is such that $\mathbf{A} = b(\mathbf{A}',\mathcal{A})$ for some complete assignment $\mathbf{A}'$ that assigns $\mathbf{T}$ only to atoms occurring in $grnd(\Pi)$.

Given any complete assignment $\mathbf{A}$, let $\mathbf{A}_{ord} = \{\mathbf{T}a \in \mathbf{A} \mid a \text{ is an ordinary atom in } grnd(\Pi)\}$. Then, $\hat{\mathbf{A}}_{ord}$ is a complete assignment that assigns $\mathbf{T}$ only to atoms occurring in $grnd(\Pi)$. We prove the statement (*) by showing that every $\mathbf{A} \in \mathsf{res}(\alpha(\Pi,\mathcal{A}))$ is such that $\mathbf{A} = b(\hat{\mathbf{A}}_{ord},\mathcal{A})$. For this, we take an arbitrary $\mathbf{A} \in \mathsf{res}(\alpha(\Pi,\mathcal{A}))$ and show that $\mathbf{A} = b(\hat{\mathbf{A}}_{ord},\mathcal{A})$.

It is easy to see that $\{\mathbf{T}a \in \mathbf{A} \mid a \text{ is an ordinary atom in } grnd(\Pi)\} = \{\mathbf{T}a \in b(\hat{\mathbf{A}}_{ord},\mathcal{A}) \mid a \text{ is an ordinary atom in } grnd(\Pi)\}$ since assignments of the value $\mathbf{T}$ to atoms in $\hat{\mathbf{A}}_{ord}$ are preserved in $b(\hat{\mathbf{A}}_{ord},\mathcal{A})$, according to Definition A.1. It is left to show that each of $\mathbf{T}p_d(\vec{X}) \in \mathbf{A}$, $\mathbf{T}\overline{p}(\vec{X}) \in \mathbf{A}$ and $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}$ if and only if it holds that $\mathbf{T}p_d(\vec{X}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$, $\mathbf{T}\overline{p}(\vec{X}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ and $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$, respectively. Note that it suffices to show that the same truth values are assinged to atoms of one of the forms $p_d(\vec{X})$, $\overline{p}(\vec{X})$ and $e_{\&g[\vec{p}]}(\vec{c})$ because $\mathbf{A}$ and $b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ do not assign $\mathbf{T}$ to any atom not occurring in $grnd(\alpha(\Pi,\mathcal{A}))$.

First, we derive that $\mathbf{T}p_d(\vec{X}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ implies $\mathbf{T}p_d(\vec{X}) \in \mathbf{A}$ since the fact $p_d(\vec{X}) \leftarrow$ is contained in $\alpha(\Pi,\mathcal{A})$ for all $p(\vec{X}) \in \mathcal{A}$. Moreover, we derive that $\mathbf{T}p_d(\vec{X}) \in \mathbf{A}$ implies $\mathbf{T}p_d(\vec{X}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ because atoms of the form $p_d(\vec{X})$ do not occur in the head of any rule in $\alpha(\Pi,\mathcal{A})$ apart from the fact $p_d(\vec{X}) \leftarrow$, and guessing and propagation as performed by Algorithm 5.1 does not assign $\mathbf{T}$ to ordinary atoms that are not defined by any rule.

In addition, we have that for all predicate symbols p of some atom in $\mathcal{A}$ the rule $\overline{p}(\mathbf{X}) \leftarrow p_d(\mathbf{X}), \text{not } p(\mathbf{X})$ is contained in $\alpha(\Pi,\mathcal{A})$. We infer that $\mathbf{T}\overline{p}(\mathbf{X}) \in \mathbf{A}$ iff $\mathbf{F}p(\mathbf{X}) \in \mathbf{A}$, for all $p(\vec{X}) \in \mathcal{A}$, since we know that the fact $p_d(\vec{X}) \leftarrow$ is contained in $\alpha(\Pi,\mathcal{A})$. Consequently, we also obtain that $\mathbf{T}\overline{p}(\vec{X}) \in \mathbf{A}$ iff $\mathbf{T}\overline{p}(\vec{X}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ because $\{\mathbf{T}\overline{p}(\vec{X}) | p(\vec{X}) \in \mathcal{A}, \mathbf{F}p(\vec{X}) \in \mathbf{A}\}$ is the projection of $b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ to signed literals of the form $\mathbf{T}\overline{p}(\vec{X})$, and since there are no other rules in $\alpha(\Pi,\mathcal{A})$ with $\overline{p}(\vec{X})$ in the head.

Finally, we have that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$ holds iff $f_{\&g}(\hat{\mathbf{A}}_{ord},\vec{p},\vec{c}) = \mathbf{T}$, for every ground external atom $\&g[\vec{p}](\vec{c})$ in $grnd(\Pi)$, according to Definition A.1. Moreover, we obtain that $f_{\&g}(\hat{\mathbf{A}}_{ord},\vec{p},\vec{c}) = \mathbf{T}$ holds iff it also holds that $f_{\&g}(\mathbf{A},\vec{p},\vec{c}) = \mathbf{T}$ as $\hat{\mathbf{A}}_{ord}$ and $\mathbf{A}$ coincide w.r.t. ordinary atoms in $grnd(\Pi)$. Hence, a rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{\mathbf{A},\vec{p}}$ is added during the computation of $\mathbf{A}$ by Algorithm 5.1 s.t. $\mathbf{A} \models \mathbf{B}_{\mathbf{A},\vec{p}}$ iff $f_{\&g}(\hat{\mathbf{A}}_{ord},\vec{p},\vec{c}) = \mathbf{T}$, and we derive that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}$ iff $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\hat{\mathbf{A}}_{ord},\mathcal{A})$.

We conclude that $\mathbf{A} = b(\hat{\mathbf{A}}_{ord},\mathcal{A})$. This finishes the proof and shows that the set $\mathsf{res}(\alpha(\Pi,\mathcal{A}))$ returned by Algorithm 5.1 for input $\alpha(\Pi,\mathcal{A})$ contains a complete assignment $\mathbf{A}$ only if $\mathbf{A} = b(\mathbf{A}',\mathcal{A})$ for some complete assignment $\mathbf{A}'$ that assigns $\mathbf{T}$ only to atoms occurring in $grnd(\Pi)$. □

Lemma A.2. *Let Π be a HEX-program, $\mathcal{A}$ be an input-safe domain of Π, and $\mathbf{A}$ be a complete assignment. The set $\mathsf{res}(\alpha(\Pi,\mathcal{A}))$ returned by Algorithm 5.1 for input $\alpha(\Pi,\mathcal{A})$ contains $b(\mathbf{A},\mathcal{A})$ if and only if $\mathbf{A}$ is an answer set of Π.*

Proof. Let Π be a HEX-program, $\mathcal{A}$ an input-safe domain of Π and $\mathsf{res}(\alpha(\Pi, \mathcal{A}))$ the set of complete assignments returned by Algorithm 5.1 for input $\alpha(\Pi, \mathcal{A})$. We need to prove that for every complete assignment $\mathbf{A}$, $b(\mathbf{A}, \mathcal{A}) \in \mathsf{res}(\alpha(\Pi, \mathcal{A}))$ if and only if $\mathbf{A}$ is an answer set of Π.

Here, we rely on the notions of *rule dependency* and *rule dependency graph* defined in Definitions 9 and 10 of (Eiter, Fink, Ianni, et al., 2016), respectively. The rule dependency graph of $grnd(\Pi)$ is called $DG(grnd(\Pi))$. In the following, let $\omega \geq 1$ be the number of strongly connected components of $DG(grnd(\Pi))$. We denote an arbitrary topological sorting of the strongly connected components of $DG(grnd(\Pi))$ by $R_1, ..., R_\omega$, where R_1 is the component that has no outgoing edges, i.e. no rule in R_1 depends on a rule outside of R_1.

We define that $S_j = R_1 \cup ... \cup R_j$, for $1 \leq j \leq \omega$. Then, according to Definition 11 of (Eiter, Fink, Ianni, et al., 2016), every S_j is a *rule splitting set* for S_k, with $1 \leq j \leq k \leq \omega$. Thus, due to the *Splitting Theorem* stated in Theorem 1 of (Eiter, Fink, Ianni, et al., 2016), we have that M is an answer set of $(S_k \setminus S_j) \cup facts(X)$ if and only if M is an answer set of S_k, where $facts(X)$ is a set of facts corresponding to the true atoms in some answer set X of S_j.

Due to the restriction on cyclic dependencies over external atoms stated above, a ground external atom in a rule contained in component R_m can only depend on rules in components $R_{m'}$, where $m' < m$, i.e. if the ground external atom $\&g[\vec{p}](\vec{c})$ occurs in a rule in R_m, then all rules with head $p(\vec{X})$, where $p \in \vec{p}$, are in some component $R_{m'}$ s.t. $m' < m$.

Moreover, when Algorithm 5.1 is called with input $\alpha(S_k, \mathcal{A}_k)$, for any $1 \leq k \leq \omega$ and arbitrary input-safe domain $\mathcal{A}_k$ of S_k, the algorithm can generate the same set of nogoods for rules corresponding to rules in S_j, with $1 \leq j \leq k \leq \omega$, in Part (c) as when it is called with input $\alpha(S_j, \mathcal{A}_j)$, with arbitrary input-safe domain $\mathcal{A}_j$ of S_j. In addition, it can derive the same truth values for replacement atoms representing external atoms in S_j in this case, and it can make the same guesses and propagations for ordinary atoms in S_j in Parts (e) and (a), respectively, because the latter do not depend on rules in $S_k \setminus S_j$. Consequently, the same set of intermediary assignments for ordinary atoms and replacement atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ for external atoms in S_j can be generated during the execution of Algorithm 5.1 with input $\alpha(S_k, \mathcal{A}_k)$ as when $\alpha(S_j, \mathcal{A}_j)$ is processed.

Correctness proof:

($\Rightarrow$) We need to prove that for every complete assignment $\mathbf{A}$ it holds that if $b(\mathbf{A}, \mathcal{A}) \in \mathsf{res}(\alpha(\Pi, \mathcal{A}))$, then $\mathbf{A}$ is an answer set of Π.

We show by induction on n that (*) for every complete assignment $\mathbf{A}^{S_n}$ it holds that if $b(\mathbf{A}^{S_n}, \mathcal{A}_n) \in \mathcal{AS}(\alpha(S_n, \mathcal{A}_n))$, where $\mathcal{A}_n$ is an arbitrary input-safe domain of S_n, then $\mathbf{A}^{S_n}$ is an answer set of S_n, for all $1 \leq n \leq \omega$.

Base case:

For the base case, consider $n = 1$, i.e. we need to show that the proposition (*) holds w.r.t. S_1. If there are no external atoms in S_1, it follows directly from the correctness of the ALPHA algorithm (cf. (Weinzierl, 2017)) that for every complete assignment $\mathbf{A}^{S_1}$ and input-safe domain $\mathcal{A}_1$ of S_1, it holds that if $b(\mathbf{A}^{S_1}, \mathcal{A}_1) \in \mathcal{AS}(\alpha(S_1, \mathcal{A}_1))$, then $\mathbf{A}^{S_1}$

is an answer set of S_1 since no nogoods are generated in Part (d) of Algorithm 5.1 and no replacement atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ occur in $\alpha(S_1, \mathcal{A}_1)$ in this case.

So, consider the case that S_1 contains external atoms. Let $\mathbf{A}^{S_1}$ be an arbitrary complete assignment s.t. $b(\mathbf{A}^{S_1}, \mathcal{A}_1) \in \mathcal{AS}(\alpha(S_1, \mathcal{A}_1))$.

We show that, for each ground external atom $\&g[\vec{p}](\vec{c})$ occurring in S_1, it holds that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ if and only if $\mathbf{A}^{S_1} \models \&g[\vec{p}](\vec{c})$. From this, it follows that $\mathbf{A}^{S_1}$ is an answer set of S_1 because then, $b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ determines exactly the truth value of $\&g[\vec{p}](\vec{c})$ under $\mathbf{A}^{S_1}$ for each ground external atom $\&g[\vec{p}](\vec{c})$ occurring in S_1; all external atoms of the form $\&g[\vec{p}](\vec{c})$ in S_1 are replaced by ordinary atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ in $\alpha(S_1, \mathcal{A}_1)$; and the alpha algorithm is correct w.r.t. ordinary answer set programs.

Let $\&g[\vec{p}](\vec{c})$ be an arbitrary ground external atom in S_1. We need to show that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ if and only if $\mathbf{A}^{S_1} \models \&g[\vec{p}](\vec{c})$.

First, consider the case that $\mathbf{A}^{S_1} \models \&g[\vec{p}](\vec{c})$, which implies that $f_{\&g}(\mathbf{A}^{S_1}, \vec{p}, \vec{c}) = \mathbf{T}$. From this, it follows that $f_{\&g}(i(b(\mathbf{A}^{S_1}, \mathcal{A}_1), \mathcal{A}_1), \vec{p}, \vec{c}) = \mathbf{T}$ as we have that $\mathbf{T}\overline{p}(\vec{X}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ iff $\mathbf{F}p(\vec{X}) \in \mathbf{A}^{S_1}$, for all $p(\vec{X}) \in \mathcal{A}_1$, due to the program transformation defined in Definition 5.5. Consequently, the rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{b(\mathbf{A}^{S_1}, \mathcal{A}_1), \vec{p}}$ is added in Part (d) of Algorithm 5.1. Moreover, we have that $b(\mathbf{A}^{S_1}, \mathcal{A}_1) \models \mathbf{B}_{b(\mathbf{A}^{S_1}, \mathcal{A}_1), \vec{p}}$ and thus, it must be the case that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$.

Second, consider the case that $\mathbf{A}^{S_1} \not\models \&g[\vec{p}](\vec{c})$, and suppose towards a contradiction that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$. From $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$, we infer that, for some $C \subseteq b(\mathbf{A}^{S_1}, \mathcal{A}_1)$, it holds that $f_{\&g}(i(C, \mathcal{A}_1), \vec{p}, \vec{c}) = \mathbf{T}$, relying on the fact that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ can only be the case if a rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{C, \vec{p}}$ is added during the execution of Algorithm 5.1 in Part (d) and it holds that $b(\mathbf{A}^{S_1}, \mathcal{A}_1) \models \mathbf{B}_{C, \vec{p}}$. Now, it cannot be the case that a rule with head $p(\vec{X})$, s.t. $p \in \vec{p}$, is contained in S_1 since R_1 is a strongly connected component of $DG(grnd(\Pi))$ where no rule in R_1 depends on a rule outside of R_1 (recall that cyclic dependencies over external atoms are not allowed). Hence, we have that for all $p(\vec{X})$, s.t. $p \in \vec{p}$, the signed literal $\mathbf{F}p(\vec{X})$ must be contained in $b(\mathbf{A}^{S_1}, \mathcal{A}_1)$. As during the evaluation of $f_{\&g}(i(C, \mathcal{A}_1), \vec{p}, \vec{c})$ all atoms $p(\vec{X})$, s.t. $p \in \vec{p}$ and $Xp(\vec{X}) \notin i(C, \mathcal{A}_1)$ for $X \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$, are treated as having the truth value $\mathbf{F}$, we derive that $f_{\&g}(i(b(\mathbf{A}^{S_1}, \mathcal{A}_1), \mathcal{A}_1), \vec{p}, \vec{c}) = \mathbf{T}$ and hence, that also $f_{\&g}(\mathbf{A}^{S_1}, \vec{p}, \vec{c}) = \mathbf{T}$ holds.

However, the fact that $\mathbf{A}^{S_1} \not\models \&g[\vec{p}](\vec{c})$ implies that $f_{\&g}(\mathbf{A}^{S_1}, \vec{p}, \vec{c}) \neq \mathbf{T}$ and hence, we obtain a contradiction. We conclude that $\mathbf{A}^{S_1}$ is an answer set of S_1 as the truth value of every replacement atom $e_{\&g[\vec{p}]}(\vec{c})$ in $b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ corresponds exactly to the truth value of $\&g[\vec{p}](\vec{c})$ under $\mathbf{A}^{S_1}$, and the alpha algorithm is correct w.r.t. ordinary answer set programs.

This finishes the base case; we continue with the induction step.

Induction step:

Next, take an arbitrary $1 \leq k < \omega$ and suppose that proposition (*) holds regarding every n with $1 \leq n \leq k$ (induction hypothesis). We show that (*) also holds for $n = k+1$.

Let $\mathbf{A}^{S_{k+1}}$ be an arbitrary complete assignment s.t. $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ is contained in $\mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$, where $\mathcal{A}_{k+1}$ is an input-safe domain of S_{k+1}. We need to prove that $\mathbf{A}^{S_{k+1}}$ is an answer set of S_{k+1}. Let $\mathbf{A}^{S_k}$ be a complete assignment s.t. $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ is

contained in $\mathcal{AS}(\alpha(S_k, \mathcal{A}_k))$ and $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ assigns the same truth values to atoms in $\alpha(S_k, \mathcal{A}_k)$ corresponding to ordinary and external atoms in S_k as $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$. The complete assignment $\mathbf{A}^{S_k}$ must exist because we have that $S_k \subset S_{k+1}$, and Algorithm 5.1 can generate the same set of nogoods for rules corresponding to rules in S_k when it is executed with input $\alpha(S_{k+1}, \mathcal{A}_{k+1})$ as when it is called with $\alpha(S_k, \mathcal{A}_k)$. Further, Algorithm 5.1 can make the same guesses and propagations for ordinary atoms in S_k, and can derive the same truth values for replacement atoms in $\alpha(S_k, \mathcal{A}_k)$ when it is executed with input $\alpha(S_{k+1}, \mathcal{A}_{k+1})$ as when it is called with $\alpha(S_k, \mathcal{A}_k)$. According to the induction hypothesis, $\mathbf{A}^{S_k}$ is an answer set of S_k.

Now, we proceed with the induction step analogously to the base case.

If there are no external atoms in $S_{k+1} \setminus S_k$, it follows from the correctness of the ALPHA algorithm (cf. (Weinzierl, 2017)) that if it holds for a complete assignment $\mathbf{A}^{S_{k+1}}$ that $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1}) \in \mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$, then $\mathbf{A}^{S_{k+1}}$ is an answer set of $(S_{k+1} \setminus S_k) \cup \mathit{facts}(\mathbf{A}^{S_k})$ because $\mathbf{A}^{S_{k+1}}$ assigns the same truth values to atoms in S_k as $\mathbf{A}^{S_k}$. According to the Splitting Theorem, we obtain that $\mathbf{A}^{S_{k+1}}$ is an answer set of S_{k+1} as $\mathbf{A}^{S_k}$ is an answer set of S_k.

Alternatively, consider the case that $S_{k+1} \setminus S_k$ contains external atoms. Similar to before, we proceed by showing that, for each ground external atom $\&g[\vec{p}](\vec{c})$ occurring in $S_{k+1} \setminus S_k$, it holds that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ if and only if $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$. Let $\&g[\vec{p}](\vec{c})$ be an arbitrary ground external atom in $S_{k+1} \setminus S_k$.

First, consider the case that $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$, which implies that $f_{\&g}(\mathbf{A}^{S_k}, \vec{p}, \vec{c}) = \mathbf{T}$ since $\&g[\vec{p}](\vec{c})$ only depends on rules in S_k. It follows that $f_{\&g}(i(b(\mathbf{A}^{S_k}, \mathcal{A}_k), \mathcal{A}_k), \vec{p}, \vec{c}) = \mathbf{T}$ as $\mathbf{T}\overline{p}(\vec{X}) \in b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ iff $\mathbf{F}p(\vec{X}) \in \mathbf{A}^{S_k}$, for all $p(\vec{X}) \in \mathcal{A}_k$. Thus, the rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{b(\mathbf{A}^{S_k}, \mathcal{A}_k), \vec{p}}$ is added in Part (d) of Algorithm 5.1, and we have that $b(\mathbf{A}^{S_k}, \mathcal{A}_k) \models \mathbf{B}_{b(\mathbf{A}^{S_k}, \mathcal{A}_k), \vec{p}}$. Consequently, it must be the case that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ since $\{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_k}, \mathcal{A}_k)\} \subseteq \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})\}$.

Second, consider the case that $\mathbf{A}^{S_{k+1}} \not\models \&g[\vec{p}](\vec{c})$, and suppose towards contradiction that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$. From $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$, we infer that, for some $C \subseteq b(\mathbf{A}^{S_k}, \mathcal{A}_k)$, it holds that $f_{\&g}(i(C, \mathcal{A}_k), \vec{p}, \vec{c}) = \mathbf{T}$. The previous holds since $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ can only be the case if a rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{C,\vec{p}}$ is added during the execution of Algorithm 5.1 in Part (d) and it holds that $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1}) \models \mathbf{B}_{C,\vec{p}}$. Because $i(C, \mathcal{A}_k)$ is input-complete w.r.t. S_k being over an input-safe domain of S_k, we have that $f_{\&g}(i(C, \mathcal{A}_k), \vec{p}, \vec{c}) = \mathbf{T}$ implies that $f_{\&g}(\mathbf{A}^{S_k}, \vec{p}, \vec{c}) = \mathbf{T}$. This is true according to Definition 5.2 because $\mathbf{A}^{S_k}$ is an answer set of S_k, due to the induction hypothesis, s.t. $\mathbf{A}^{S_k} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$, where $\mathbf{A}_{\vec{p},\mathcal{A}} = \{Xa \in i(C, \mathcal{A}_k) \mid a \text{ has predicate } p \in \vec{p}\}$. However, the fact that $\mathbf{A}^{S_k} \not\models \&g[\vec{p}](\vec{c})$ implies that $f_{\&g}(\mathbf{A}^{S_k}, \vec{p}, \vec{c}) \neq \mathbf{T}$ and hence, we obtain a contradiction.

We infer that $\mathbf{A}^{S_{k+1}}$ assigns the same truth values to atoms in S_k as $\mathbf{A}^{S_k}$ because $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ assigns the same truth values to atoms in $\alpha(S_k, \mathcal{A}_k)$ that correspond to ordinary and external atoms in S_k as $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$. In addition, due to the induction hypothesis, we have that $\mathbf{A}^{S_k}$ is an answer set of S_k. We infer that $\mathbf{A}^{S_{k+1}}$ is an answer set of $(S_{k+1} \setminus S_k) \cup \mathit{facts}(\mathbf{A}^{S_k})$, where $\mathit{facts}(\mathbf{A}^{S_k})$ is a set of facts corresponding to the true

atoms in $\mathbf{A}^{S_k}$, because, for each ground external atom $\&g[\vec{p}](\vec{c})$ occurring in $S_{k+1} \setminus S_k$, the truth values of $e_{\&g[\vec{p}]}(\vec{c})$ in $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ and $\&g[\vec{p}](\vec{c})$ under $\mathbf{A}^{S_{k+1}}$ coincide, and we know from (Weinzierl, 2017) that Algorithm 5.1 is correct for ordinary answer set solving. According to the Splitting Theorem, we obtain that $\mathbf{A}^{S_{k+1}}$ is an answer set of S_{k+1}.

This finishes the induction step, and we obtain, for all $1 \leq n \leq \omega$, that for every complete assignment $\mathbf{A}^{S_n}$ s.t. $b(\mathbf{A}^{S_n}, \mathcal{A}_n) \in \mathcal{AS}(\alpha(S_n, \mathcal{A}_n))$ it holds that $\mathbf{A}^{S_n}$ is an answer set of S_n, where $\mathcal{A}_n$ is an input-safe domain of S_n. We conclude that for every complete assignment $\mathbf{A}$ it holds that if $b(\mathbf{A}, \mathcal{A}) \in \mathsf{res}(\alpha(\Pi, \mathcal{A}))$, then $\mathbf{A}$ is an answer set of Π since $\Pi = S_\omega$ and $\mathcal{A}$ is an input-safe domain of Π. Hence, Algorithm 5.1 is correct.

Completeness proof:

($\Leftarrow$) We need to prove that, for every complete assignment $\mathbf{A}$, if $\mathbf{A}$ is an answer set of Π, then $b(\mathbf{A}, \mathcal{A})$ is contained in $\mathsf{res}(\alpha(\Pi, \mathcal{A}))$.

We show by induction on n that (**) for every complete assignment $\mathbf{A}^{S_n}$ it holds that if $\mathbf{A}^{S_n}$ is an answer set of S_n, then $b(\mathbf{A}^{S_n}, \mathcal{A}_n) \in \mathcal{AS}(\alpha(S_n, \mathcal{A}_n))$, where $\mathcal{A}_n$ is an arbitrary input-safe domain of S_n, for all $1 \leq n \leq \omega$.

Base case:

For the base case, consider $n = 1$, i.e. we need to show that proposition (**) holds w.r.t. S_1.

If there are no external atoms in S_1, it follows directly from the completeness of the ALPHA algorithm (cf. (Weinzierl, 2017)) that if $\mathbf{A}^{S_1}$ is an answer set of S_1 and $\mathcal{A}_1$ an input-safe domain of S_1, then the complete assignment $b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ is contained in $\mathcal{AS}(\alpha(S_1, \mathcal{A}_1))$ because we have that $S_1 \subseteq \alpha(S_1, \mathcal{A}_1)$ in this case, and S_1 does not contain any atom that is defined in $\alpha(S_1, \mathcal{A}_1) \setminus S_1$.

So, consider the case that S_1 contains external atoms. Let $\mathbf{A}^{S_1}$ be an arbitrary answer set of S_1. Let $\&g[\vec{p}](\vec{c})$ be an arbitrary ground external atom in S_1.

We need to show that $b(\mathbf{A}^{S_1}, \mathcal{A}_1) \in \mathcal{AS}(\alpha(S_1, \mathcal{A}_1))$. Note that, as stated before, there cannot be a rule with head $p(\vec{X})$, s.t. $p \in \vec{p}$, contained in $grnd(\Pi)$ since R_1 is a strongly connected component of $DG(grnd(\Pi))$ where no rule in R_1 depends on a rule outside of R_1 (there are no cyclic dependencies over external atoms).

Accordingly, for every atom $p(\vec{X})$ with $p \in \vec{p}$, it must be the case that $\mathbf{F}p(\vec{X}) \in \mathbf{A}^{S_1}$. Moreover, for every atom $p(\vec{X}) \in \mathcal{A}_1$ with $p \in \vec{p}$, it must be the case that $\mathbf{T}\overline{p}(\vec{X})$ is contained in every complete assignment in $\mathcal{AS}(\alpha(S_1, \mathcal{A}_1))$. Then, however, it holds that $\mathbf{A}^{S_1} \models \&g[\vec{p}](\vec{c})$ if and only if the rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{\{\mathbf{F}p(\vec{X}) | p(\vec{X}) \in \mathcal{A}_1, p \in \vec{p}\}, \vec{p}}$ is added in Part (d) when Algorithm 5.1 is executed with $\alpha(S_1, \mathcal{A}_1)$ as input. It follows that $b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ is contained in $\mathcal{AS}(\alpha(S_1, \mathcal{A}_1))$ because then, $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$ is derived by Algorithm 5.1 if and only if $\mathbf{A}^{S_1} \models \&g[\vec{p}](\vec{c})$, i.e. it holds that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_1}, \mathcal{A}_1)$ if and only if $\mathbf{A}^{S_1} \models \&g[\vec{p}](\vec{c})$; and the ALPHA algorithm is complete w.r.t. ordinary answer set programs.

This finishes the base case; we continue with the induction step.

Induction step:

Next, take an arbitrary $1 \leq k < \omega$ and suppose that statement (**) holds regarding every n with $1 \leq n \leq k$ (induction hypothesis). We show that (**) also holds for $n = k + 1$.

Let $\mathbf{A}^{S_{k+1}}$ be an arbitrary answer set of S_{k+1}, and $\mathcal{A}_{k+1}$ an input-safe domain of S_{k+1}. We need to show that $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ is contained in $\mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$. Let $\mathbf{A}^{S_k}$ be an answer set of S_k s.t. $\mathbf{A}^{S_{k+1}}$ is an answer set of $S_{k+1} \setminus S_k \cup \mathit{facts}(\hat{\mathbf{A}}^{S_k}_{ord})$, which must exist according to the Lifting Theorem. Then, due to the induction hypothesis, we have that $b(\mathbf{A}^{S_k}, \mathcal{A}_k) \in \mathcal{AS}(\alpha(S_k, \mathcal{A}_k))$.

If there are no external atoms in $S_{k+1} \setminus S_k$, it follows directly from the completeness of the ALPHA algorithm (cf. (Weinzierl, 2017)) that $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1}) \in \mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$ since $b(\mathbf{A}^{S_k}, \mathcal{A}_k) \in \mathcal{AS}(\alpha(S_k, \mathcal{A}_k))$; and Algorithm 5.1 can generate the same set of nogoods for rules corresponding to rules in S_k when it is executed with input $\alpha(S_{k+1}, \mathcal{A}_{k+1})$ as when it is called with $\alpha(S_k, \mathcal{A}_k)$, and it can make the same guesses and propagations for atoms corresponding to atoms in S_k. Consequently, during the execution of Algorithm 5.1 with input $\mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$, an intermediary assignment is obtained that contains all assignments to atoms corresponding to atoms in S_k from $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$, and it can be extended to $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$.

Now, consider the case that $S_{k+1} \setminus S_k$ contains external atoms.

We show that, for each ground external atom $\&g[\vec{p}](\vec{c})$ occurring in $S_{k+1} \setminus S_k$, it holds that $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$ if and only if a rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{C,\vec{p}}$, with $C \subseteq b(\mathbf{A}^{S_k}, \mathcal{A}_k)$, is added in Part (d) when Algorithm 5.1 is executed with $\alpha(S_{k+1}, \mathcal{A}_{k+1})$ as input. Then, $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$ is derived by Algorithm 5.1 when executed with input $\alpha(S_k, \mathcal{A}_k)$ based on an intermediary assignment C if and only if $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$. Thus, it holds that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ if and only if $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$ since an intermediary assignment is obtained that contains all assignments to atoms corresponding to atoms in S_k from $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ when Algorithm 5.1 is executed with input $\alpha(S_{k+1}, \mathcal{A}_{k+1})$. Furthermore, the ALPHA algorithm is complete w.r.t. ordinary answer set programs and hence, it follows that $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ is contained in $\mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$.

Let $\&g[\vec{p}](\vec{c})$ be an arbitrary ground external atom in $S_{k+1} \setminus S_k$.

First, consider the case that $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$, which implies that $f_{\&g}(\mathbf{A}^{S_k}, \vec{p}, \vec{c}) = \mathbf{T}$ since $\&g[\vec{p}](\vec{c})$ only depends on rules in S_k. It follows that $f_{\&g}(i(b(\mathbf{A}^{S_k}, \mathcal{A}_k), \mathcal{A}_k), \vec{p}, \vec{c}) = \mathbf{T}$ since $\mathbf{T}\overline{p}(\vec{X}) \in b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ iff $\mathbf{F}p(\vec{X}) \in \mathbf{A}^{S_k}$, for all $p(\vec{X}) \in \mathcal{A}_k$. In addition, the rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{b(\mathbf{A}^{S_k}, \mathcal{A}_k), \vec{p}}$ is added in Part (d) of Algorithm 5.1 because the assignments to atoms corresponding to atoms in S_k from $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ are contained in an intermediary assignment when the algorithm is executed with input $\alpha(S_{k+1}, \mathcal{A}_{k+1})$.

Second, consider the second case, namely that $\mathbf{A}^{S_{k+1}} \not\models \&g[\vec{p}](\vec{c})$, and suppose towards contradiction that a rule $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{C,\vec{p}}$, with $C \subseteq b(\mathbf{A}^{S_k}, \mathcal{A}_k)$, is added in Part (d) when Algorithm 5.1 is executed with $\alpha(S_{k+1}, \mathcal{A}_{k+1})$ as input. We infer that $f_{\&g}(i(C, \mathcal{A}_k), \vec{p}, \vec{c}) = \mathbf{T}$. Since the assignment $i(C, \mathcal{A}_k)$ is input-complete w.r.t. S_k as it is over an input-safe domain of S_k, we have that $f_{\&g}(i(C, \mathcal{A}_k), \vec{p}, \vec{c}) = \mathbf{T}$ implies that $f_{\&g}(\mathbf{A}^{S_k}, \vec{p}, \vec{c}) = \mathbf{T}$. The previous holds according to Definition 5.2 because $\mathbf{A}^{S_k}$ is an answer set of S_k, s.t. $\mathbf{A}^{S_k} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$, where $\mathbf{A}_{\vec{p},\mathcal{A}} = \{Xa \in i(C, \mathcal{A}_k) \mid a \text{ has predicate } p \in \vec{p}\}$. However, the fact that $\mathbf{A}^{S_{k+1}} \not\models \&g[\vec{p}](\vec{c})$ implies that $f_{\&g}(\mathbf{A}^{S_k}, \vec{p}, \vec{c}) \neq \mathbf{T}$ and hence, we obtain a contradiction.

We obtain that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$ is derived by Algorithm 5.1 based on an intermediary assignment C when executed with input $\alpha(S_k, \mathcal{A}_k)$ if and only if $\mathbf{A}^{S_{k+1}} \models \&g[\vec{p}](\vec{c})$.

In addition, recall that due to the induction hypothesis, we have that $b(\mathbf{A}^{S_k}, \mathcal{A}_k) \in \mathcal{AS}(\alpha(S_k, \mathcal{A}_k))$. We infer that $b(\mathbf{A}^{S_{k+1}}, \mathcal{A}_{k+1})$ is contained in $\mathcal{AS}(\alpha(S_{k+1}, \mathcal{A}_{k+1}))$ because Algorithm 5.1 derives an intermediary assignment that contains all assignments to atoms corresponding to atoms in S_k from $b(\mathbf{A}^{S_k}, \mathcal{A}_k)$ when it is executed with input $\alpha(S_{k+1}, \mathcal{A}_{k+1})$, and we know from (Weinzierl, 2017) that Algorithm 5.1 is complete for ordinary answer set solving.

This finishes the induction step, and we infer that, for all n, with $1 \leq n \leq \omega$, and every complete assignment $\mathbf{A}^{S_n}$, it holds that if $\mathbf{A}^{S_n}$ is an answer set of S_n, then $b(\mathbf{A}^{S_n}, \mathcal{A}_n) \in \mathcal{AS}(\alpha(S_n, \mathcal{A}_n))$, where $\mathcal{A}_n$ is an input-safe domain of S_n. We conclude that for every complete assignment $\mathbf{A}$, if $\mathbf{A}$ is an answer set of Π, then $b(\mathbf{A}, \mathcal{A})$ is contained in $\mathsf{res}(\alpha(\Pi, \mathcal{A}))$ since $\Pi = S_\omega$ and $\mathcal{A}$ is an input-safe domain of Π. Consequently, that Algorithm 5.1 is complete. □

Proposition 5.4. For HEX-program Π and input-safe domain $\mathcal{A}$ of Π, Algorithm 5.1 yields the answer sets of $\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$.

Proof. Let Π be a HEX-program and $\mathcal{A}$ an input-safe domain of Π. We know from Lemma A.2 that, given a complete assignment $\mathbf{A}$, the set $\mathsf{res}(\alpha(\Pi, \mathcal{A}))$ returned by Algorithm 5.1 for input $\alpha(\Pi, \mathcal{A})$ contains $b(\mathbf{A}, \mathcal{A})$ if and only if $\mathbf{A}$ is an answer set of Π. Moreover, we know from Lemma A.1 that $\mathsf{res}(\alpha(\Pi, \mathcal{A}))$ contains a complete assignment $\mathbf{A}$ only if $\mathbf{A} = b(\mathbf{A}', \mathcal{A})$ for some complete assignment $\mathbf{A}'$ that assigns $\mathbf{T}$ only to atoms occurring in $grnd(\Pi)$. Hence, we can prove the proposition by showing that, (*) given a complete assignment $\mathbf{A}$, $b(\mathbf{A}, \mathcal{A})$ is an answer set of $\alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$ if and only if $\mathbf{A}$ is an answer set of Π.

($\Rightarrow$) We need to show that, given a complete assignment $\mathbf{A}$, if $b(\mathbf{A}, \mathcal{A})$ is an answer set of $\Pi' = \alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$, then $\mathbf{A}$ is an answer set of Π. Let $\mathbf{A}$ be a complete assignment s.t. $b(\mathbf{A}, \mathcal{A})$ is an answer set of Π', i.e. it is a $\leq$-minimal model of the FLP-reduct $f\Pi'^{b(\mathbf{A}, \mathcal{A})}$, according to Definition 2.4. We need to show that $\mathbf{A}$ is a $\leq$-minimal model of $f\Pi^{\mathbf{A}}$.

First, we show that $\mathbf{A} \models f\Pi^{\mathbf{A}}$ by showing that $\mathbf{T}H \in \mathbf{A}$ for all rules r of the form $H \leftarrow B$ in $f\Pi^{\mathbf{A}}$. We have that for every $r \in \Pi$ there is a rule $r' \in \Pi'$, s.t. all external atoms of the form $\&g[\vec{p}](\vec{c})$ occurring in r are replaced by ordinary atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ in r', due to the construction of $\alpha(\Pi, \mathcal{A})$ according to Definition 5.5. In addition, we know, due to Definition A.1, that $b(\mathbf{A}, \mathcal{A}) \models \mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$ iff $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$, where $\&g[\vec{p}](\vec{c})$ is a ground external atom in $grnd(\Pi)$. As $\mathbf{A}$ satisfies all bodies of rules in $f\Pi^{\mathbf{A}}$, according to the definition of the FLP-reduct, and since external atoms only occur in the heads of rules in Π, it must be the case that for every rule $H \leftarrow B \in f\Pi^{\mathbf{A}}$ there is a rule $H \leftarrow B' \in f\Pi'^{b(\mathbf{A}, \mathcal{A})}$, s.t. all external atoms of the form $\&g[\vec{p}](\vec{c})$ occurring in $H \leftarrow B$ are replaced by ordinary atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ in $H \leftarrow B'$. Hence, from $b(\mathbf{A}, \mathcal{A}) \models f\Pi'^{b(\mathbf{A}, \mathcal{A})}$, we derive that $\mathbf{T}H \in b(\mathbf{A}, \mathcal{A})$ for all rules $H \leftarrow B' \in f\Pi'^{b(\mathbf{A}, \mathcal{A})}$ and consequently, we obtain also that $\mathbf{T}H \in \mathbf{A}$ for all rules $H \leftarrow B \in f\Pi^{\mathbf{A}}$.

Next, we show that there is no complete assignment $\mathbf{A}'$, with $\mathbf{A}' < \mathbf{A}$, s.t. $\mathbf{A}' \models f\Pi^{\mathbf{A}}$. Recall that for partial assignments $\mathbf{A}_1$ and $\mathbf{A}_2$, we denote by $\mathbf{A}_1 < \mathbf{A}_2$ that $\{\mathbf{T}a \in \mathbf{A}_1\} \subset \{\mathbf{T}a \in \mathbf{A}_2\}$. We assume towards a contradiction that there is a complete assignment $\mathbf{A}'$, with $\mathbf{A}' < \mathbf{A}$, s.t. $\mathbf{A}' \models f\Pi^{\mathbf{A}}$.

Since no external input-cycles (where an input predicate of an external atom depends on the atom itself) are allowed, the truth value of an external atom $\&g[\vec{p}](\vec{c})$ in the body of a rule $H \leftarrow B \in f\Pi^{\mathbf{A}}$ cannot depend on the atom H. Now, suppose that for some rule $H \leftarrow B \in f\Pi^{\mathbf{A}}$ and some external atom $\&g[\vec{p}](\vec{c})$ in B either $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{F}$, or $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$. Let $\mathbf{A}'' = (\mathbf{A}' \setminus \mathbf{F}H) \cup \{\mathbf{T}H\}$. Then it must hold that $\mathbf{A}'' < \mathbf{A}$ because

1. $\mathbf{T}H \in \mathbf{A}$ since $\mathbf{A} \models f\Pi^{\mathbf{A}}$ and due to the definition of the FLP-reduct, and
2. there must be some $\mathbf{T}p(\vec{c}) \in \mathbf{A}$ s.t. $\mathbf{F}p(\vec{c}) \in \mathbf{A}''$ which is the reason for the change of the value of $\&g[\vec{p}](\vec{c})$.

Hence, we can assume without loss of generality that our $\mathbf{A}'$, with $\mathbf{A}' < \mathbf{A}$, is such that for all rules $H \leftarrow B \in f\Pi^{\mathbf{A}}$, where for some external atom $\&g[\vec{p}](\vec{c})$ in B either $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{F}$, or $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$, it holds that $\mathbf{T}H \in \mathbf{A}'$.

We derive a contradiction by showing that $b(\mathbf{A}, \mathcal{A})$ is not a $\leq$-minimal model of $f\Pi'^{b(\mathbf{A},\mathcal{A})}$. Let $b(\mathbf{A}', \mathcal{A})'$ be the completion of $(b(\mathbf{A}', \mathcal{A}) \setminus \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}', \mathcal{A})\}) \cup \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A}, \mathcal{A})\}$ w.r.t. $\mathcal{HB}$, i.e. we keep the truth assignments regarding replacement atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ which correspond to the evaluation of $\&g[\vec{p}](\vec{c})$ under $\mathbf{A}$. By construction of $b(\mathbf{A}', \mathcal{A})'$, it holds that $b(\mathbf{A}', \mathcal{A})' < b(\mathbf{A}, \mathcal{A})$ as $b(\mathbf{A}', \mathcal{A})'$ differs from $b(\mathbf{A}, \mathcal{A})$ only in that some ordinary atoms occurring in Π which are assigned the truth value $\mathbf{T}$ by $b(\mathbf{A}, \mathcal{A})$ are mapped to $\mathbf{F}$ by $b(\mathbf{A}', \mathcal{A})'$. We show that $b(\mathbf{A}', \mathcal{A})'$ is a model of $f\Pi'^{b(\mathbf{A},\mathcal{A})}$.

First, all facts of the form $p_d(\vec{X})$ must still be satisfied under $b(\mathbf{A}', \mathcal{A})'$; and all heads of rules of the form $\overline{p}(\vec{X}) \leftarrow p_d(\vec{X}), \mathrm{not}\, p(\vec{X})$ in $f\Pi'^{b(\mathbf{A},\mathcal{A})}$ are satisfied under $b(\mathbf{A}', \mathcal{A})'$ because from $\mathbf{A}' < \mathbf{A}$ it follows that $\{\overline{p}(\vec{X}) \in b(\mathbf{A}, \mathcal{A})\} \subset \{\overline{p}(\vec{X}) \in b(\mathbf{A}', \mathcal{A})\}$. Additionally, we know that all rule heads of the form $e_{\&g[\vec{p}]}(\vec{c})$ in $f\Pi'^{b(\mathbf{A},\mathcal{A})}$ are satisfied by $b(\mathbf{A}', \mathcal{A})'$ because of the construction of $b(\mathbf{A}', \mathcal{A})'$ above. Finally, let $H \leftarrow B$ be an arbitrary rule in $f\Pi'^{b(\mathbf{A},\mathcal{A})}$ s.t. $H \leftarrow B$ corresponds to a rule $H \leftarrow B' \in f\Pi^{\mathbf{A}}$, where all external atoms of the form $\&g[\vec{p}](\vec{c})$ occurring in B' are replaced by ordinary atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ in B. It is left to show that if $b(\mathbf{A}', \mathcal{A})' \models B$, then $b(\mathbf{A}', \mathcal{A})' \models H$. So, suppose that $b(\mathbf{A}', \mathcal{A})' \models B$. Now, in case $\mathbf{A}' \models B$, we obtain that $b(\mathbf{A}', \mathcal{A})' \models H$ because it follows from Definition A.1 that $b(\mathbf{A}', \mathcal{A})' \supset \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A}'\}$, and as we have that $\mathbf{A}' \models f\Pi^{\mathbf{A}}$. In case $\mathbf{A}' \not\models B$, there must be some $\&g[\vec{p}](\vec{c})$ occurring in B s.t. either $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{F}$, or $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ and $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$, again because $b(\mathbf{A}', \mathcal{A})' \supset \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A}'\}$ and due to the fact that $\mathbf{A}' < \mathbf{A}$. However, then we have that $b(\mathbf{A}', \mathcal{A})' \models H$ due to our choice of $\mathbf{A}'$.

Consequently, $b(\mathbf{A}', \mathcal{A})'$ is a model of $f\Pi'^{b(\mathbf{A},\mathcal{A})}$, which contradicts $b(\mathbf{A}, \mathcal{A})$ being a $\leq$-minimal model of $f\Pi'^{b(\mathbf{A},\mathcal{A})}$. We infer that there is no complete assignment $\mathbf{A}'$, with $\mathbf{A}' < \mathbf{A}$, s.t. $\mathbf{A}' \models f\Pi^{\mathbf{A}}$.

($\Leftarrow$) Now, we need to show that, given a complete assignment $\mathbf{A}$, if $\mathbf{A}$ is an answer set of Π, then $b(\mathbf{A}, \mathcal{A})$ is an answer set of $\Pi' = \alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$. Let $\mathbf{A}$ be an answer set of Π. We need to show that $b(\mathbf{A}, \mathcal{A})$ is an answer set of $\Pi' = \alpha(\Pi, \mathcal{A}) \cup \eta(\Pi)$, i.e. that $b(\mathbf{A}, \mathcal{A})$ is a $\leq$-minimal model of $f\Pi'^{b(\mathbf{A},\mathcal{A})}$.

Similar to before, we first show that $b(\mathbf{A},\mathcal{A}) \models f\Pi'^{b(\mathbf{A},\mathcal{A})}$ by showing that $\mathbf{T}H(r) \in b(\mathbf{A},\mathcal{A})$ for all rules r in $f\Pi'^{b(\mathbf{A},\mathcal{A})}$. For each rule $r \in f\Pi'^{b(\mathbf{A},\mathcal{A})}$ corresponding to a rule $r' \in \Pi$, where all external atoms of the form $\&g[\vec{p}](\vec{c})$ occurring in r' are replaced by ordinary atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ in r, we have that $r' \in f\Pi\mathbf{A}$ because $b(\mathbf{A},\mathcal{A}) \models B(r)$ iff $\mathbf{A} \models B(r')$, which follows from Definition A.1. We obtain that $\mathbf{T}H(r) \in b(\mathbf{A},\mathcal{A})$ in this case since $H(r) = H(r')$ for all rules $r \in f\Pi'^{b(\mathbf{A},\mathcal{A})}$, and because we have that $\mathbf{A} \models f\Pi^{\mathbf{A}}$. Regarding rules $r \in f\Pi'^{b(\mathbf{A},\mathcal{A})}$ of the form $\overline{p}(\mathbf{X}) \leftarrow p_d(\mathbf{X}), \text{not } p(\mathbf{X})$ or $p_d(\mathbf{X}) \leftarrow$ we also derive that $\mathbf{T}H(r) \in b(\mathbf{A},\mathcal{A})$, due to the construction of $b(\mathbf{A},\mathcal{A})$ according to Definition A.1. Recall that $\eta(\Pi) = \{r \mid \exists \mathbf{A}' \text{ s.t. } r \in \eta(\&g[\vec{p}], i(\mathbf{A}',\mathcal{A}))$, $\&g[\vec{p}]$ occurs in $\Pi\}$. Accordingly, it is left to show that for all rules $r \in f\Pi'^{b(\mathbf{A},\mathcal{A})}$ of the form $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{\mathbf{A}',\vec{p}}$, i.e. $r \in \eta(\Pi)$, it holds that $\mathbf{T}H(r) \in b(\mathbf{A},\mathcal{A})$.

Let $r = e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{\mathbf{A}',\vec{p}}$ be an arbitrary rule in $f\Pi'^{b(\mathbf{A},\mathcal{A})} \cap \eta(\Pi)$. Due to the definition of the FLP-reduct, we know that $r \in f\Pi'^{b(\mathbf{A},\mathcal{A})}$ implies that $b(\mathbf{A},\mathcal{A}) \models \mathbf{B}_{\mathbf{A}',\vec{p}}$, i.e. that $\{\mathbf{T}p'(\vec{X}) \in \mathbf{A}' \mid p' \in \{\overline{p}, p\}, p \in \vec{p}\} \subseteq b(\mathbf{A},\mathcal{A})$. From this, we derive that $\mathbf{A} \succeq \mathbf{A}_{\vec{p},\mathcal{A}}$, where $\mathbf{A}_{\vec{p},\mathcal{A}} = \{Xp(\vec{X}) \in i(\mathbf{A}',\mathcal{A}) \mid p \in \vec{p}\}$, by Definition A.1 and the construction of $i(\mathbf{A}',\mathcal{A})$ according to Definition 5.5. Since $e_{\&g[\vec{p}]}(\vec{c}) \leftarrow \mathbf{B}_{\mathbf{A}',\vec{p}} \in \eta(\Pi)$, we know that $f_{\&g}(i(\mathbf{A}',\mathcal{A}),\vec{p},\vec{c}) = \mathbf{T}$. By Definition 5.2, we obtain that $f_{\&g}(\mathbf{A},\vec{p},\vec{c}) = \mathbf{T}$ because $i(\mathbf{A}',\mathcal{A})$ is input-complete w.r.t. Π being over an input-safe domain $\mathcal{A}$ of Π. By Definition A.1, we derive that $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in b(\mathbf{A},\mathcal{A})$ because we have that $f_{\&g}(\mathbf{A},\vec{p},\vec{c}) = \mathbf{T}$ and hence, $\mathbf{T}H(r) \in b(\mathbf{A},\mathcal{A})$.

Lastly, we show that there is no complete assignment $\mathbf{A}'$, with $\mathbf{A}' < b(\mathbf{A},\mathcal{A})$, s.t. $\mathbf{A}' \models f\Pi'^{b(\mathbf{A},\mathcal{A})}$. Assume again towards contradiction that $\mathbf{A}'$ is a complete assignment, with $\mathbf{A}' < b(\mathbf{A},\mathcal{A})$, s.t. $\mathbf{A}' \models f\Pi'^{b(\mathbf{A},\mathcal{A})}$. Then, there must also be a compete assignment $\mathbf{A}''$, with $\mathbf{A}'' < b(\mathbf{A},\mathcal{A})$ and $\mathbf{A}'' \models f\Pi'^{b(\mathbf{A},\mathcal{A})}$, s.t. $\{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}''\} = \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}\}$, due to the fact that no cyclic dependencies over external atoms are allowed. Without loss of generality, let $\mathbf{A}''$ be a compete assignment $\mathbf{A}''$, with $\mathbf{A}'' < b(\mathbf{A},\mathcal{A})$ and $\mathbf{A}'' \models f\Pi'^{b(\mathbf{A},\mathcal{A})}$, s.t. $\{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}''\} = \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}\}$. We derive a contradiction by showing that $\mathbf{A}''_{ord} = \{\mathbf{T}a \in \mathbf{A}'' \mid a \text{ is an ordinary atom in } grnd(\Pi)\}$ is a model of $f\Pi^{\mathbf{A}}$ with $\mathbf{A}''_{ord} < \mathbf{A}$.

Note that the truth values of atoms of the form $\overline{p}(\vec{X})$ or $p_d(\vec{X})$ cannot change from $\mathbf{T}$ under $\mathbf{A}$ to $\mathbf{F}$ under $\mathbf{A}''$ due to the construction of $b(\mathbf{A},\mathcal{A})$ and $\alpha(\Pi,\mathcal{A})$ according to Definitions A.1 and 5.5. Consequently, we have that $\mathbf{A}''_{ord} < \mathbf{A}$.

Now, we prove that $\mathbf{A}''_{ord} \models f\Pi^{\mathbf{A}}$ by showing that for every rule $H \leftarrow B \in f\Pi^{\mathbf{A}}$ and the corresponding rule $H \leftarrow B' \in f\Pi'^{b(\mathbf{A},\mathcal{A})}$, where all external atoms $\&g[\vec{p}](\vec{c})$ occurring in B are replaced by ordinary atoms of the form $e_{\&g[\vec{p}]}(\vec{c})$ in B', it holds that $\mathbf{A}''_{ord} \models B$ implies that $\mathbf{A}'' \models B'$. Let $r = H \leftarrow B$ be a rule in $f\Pi^{\mathbf{A}}$ and $r' = H \leftarrow B'$ be the corresponding rule in $\Pi'^{b(\mathbf{A},\mathcal{A})}$. If $\mathbf{A}''_{ord} \not\models B$ or B does not contain any external atoms, the implication follows straightforwardly since, in the latter case, it holds that $B = B'$ and $\mathbf{A}''_{ord}$ maps ordinary atoms to the same truth values as $\mathbf{A}''$. So, consider the case that $\mathbf{A}''_{ord} \models B$, and that an external atom $\&g[\vec{p}](\vec{c})$ occurs in B. Then, $\&g[\vec{p}](\vec{c})$ is replaced by $e_{\&g[\vec{p}]}(\vec{c})$ in B'. Since we know that $r \in f\Pi^{\mathbf{A}}$ iff $\mathbf{A} \models B$, we infer that $\mathbf{A}''_{ord} \models \&g[\vec{p}](\vec{c})$ iff $\mathbf{A} \models \&g[\vec{p}](\vec{c})$. As we know that $b(\mathbf{A},\mathcal{A}) \models \mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$ iff $f_{\&g}(\mathbf{A},\vec{p},\vec{c}) = \mathbf{T}$, where $\&g[\vec{p}](\vec{c})$ is a ground external atom in $grnd(\Pi)$, and we have chosen $\mathbf{A}''$ s.t. $\{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}''\} = \{\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}\}$, we obtain that $\mathbf{A}'' \models B'$.

Hence, we derive that there is no complete assignment $\mathbf{A}'$, with $\mathbf{A}' < b(\mathbf{A}, \mathcal{A})$, s.t. $\mathbf{A}' \models f\Pi'^{b(\mathbf{A},\mathcal{A})}$. □

Theorem 5.2. For a HEX-program Π and an input-safe domain $\mathcal{A}$ of Π, the answer sets returned by Algorithm 5.1 correspond exactly to the answer sets of Π.

Proof. The theorem follows directly from Lemmas A.1 and A.2. □

APPENDIX B

HEX-MIL-Encodings

To illustrate the concrete encodings and the instances employed for the empirical evaluation in Chapter 6, we present the encodings $\Pi_f(\mathcal{M})$ and $\Pi_{sa}(\mathcal{M})$ as well as the input to Metagol used for the *Robot Waiter Strategies* benchmark (BM3). A sample instance and a corresponding solution of benchmark (BM3) can be found at the end of this section.

Moreover, the encodings of all benchmark problems used in Section 6 and all instances used in the experiments are available at `http://www.kr.tuwien.ac.at/staff/kaminski/thesis/hexmil-experiments.zip`.

Forward-Chained HEX-MIL-Encoding

```
binary(pour_tea,X,Y) :- &pour_tea[X](Y), state(X).
binary(pour_coffee,X,Y) :- &pour_coffee[X](Y), state(X).
binary(move_right,X,Y) :- &move_right[X](Y), state(X).

unary(wants_tea,X) :- &wants_tea[X](), state(X).
unary(wants_coffee,X) :- &wants_coffee[X](), state(X).
unary(at_end,X) :- &at_end[X](), state(X).

order(X,Y) :- skolem(X), binary(Y,_,_).
order(X,Y) :- pos_ex(X,_,_), binary(Y,_,_).
order(X,Y) :- pos_ex(X,_,_), skolem(Y).
order(X,Y) :- skolem(X), skolem(Y), X < Y.

{meta(precon,P1,P2,P3)} :- order(P1,P3), unary(P2,X), deduced(P3,X,Y).
{meta(postcon,P1,P2,P3)} :- order(P1,P2), deduced(P2,X,Y), unary(P3,Y).
{meta(chain,P1,P2,P3)} :- order(P1,P2), order(P1,P3), deduced(P2,X,Z),
                                                      deduced(P3,Z,Y).
{meta(tailrec,P1,P2,n)} :- order(P1,P2), deduced(P2,X,Z), deduced(P1,Z,Y).

deduced(P1,X,Y) :- meta(precon,P1,P2,P3), unary(P2,X), deduced(P3,X,Y).
deduced(P1,X,Y) :- meta(postcon,P1,P2,P3), deduced(P2,X,Y), unary(P3,Y).
deduced(P1,X,Y) :- meta(chain,P1,P2,P3), deduced(P2,X,Z), deduced(P3,Z,Y).
deduced(P1,X,Y) :- meta(tailrec,P1,P2,n), deduced(P2,X,Z), deduced(P1,Z,Y).
```

```
state(X) :- pos_ex(_,X,_).
state(Y) :- deduced(_,_,Y).

deduced(P,X,Y) :- binary(P,X,Y).

:- pos_ex(P,X,Y), not deduced(P,X,Y).
:- pos_ex(P,X,Y1), deduced(P,X,Y2), Y1 != Y2.

:- #count{ M,P1,P2,P3 : meta(M,P1,P2,P3) } != N, size(N).
```

State Abstraction HEX-MIL-Encoding

Note that even though the syntax of the external atom used for importing binary and unary background knowledge as well as the positive examples in the encoding below differs from the external atoms used in Definition 6.13, identical extensions are imported for the atoms *binary*, *unary* and *pos* as described in Section 6.3. Hence, the encoding is equivalent to the encoding of Definition 6.13.

```
binary(A,N1,N2) :- &abduceSequence[ID,ExStart,ExEnd](X,N1,N2,A),
                                pos_ex(ID,_,ExStart,ExEnd), X = seq.
unary(A,N) :- &abduceSequence[ID,ExStart,ExEnd](X,N,N,A),
                                pos_ex(ID,_,ExStart,ExEnd), X = check.
pos(ID,A,N1,N2) v n_pos(ID,A,N1,N2) :-
                                &abduceSequence[ID,ExStart,ExEnd](X,N1,N2,A),
                                pos_ex(ID,_,ExStart,ExEnd), X = goal.

:- &failNeg[meta,pos_ex]().

pos1(ID) :- pos(ID,_,_,_).
:- pos_ex(ID,_,_,_), not pos1(ID).

order(X,Y) :- skolem(X), binary(Y,_,_).
order(X,Y) :- pos(X,_,_), binary(Y,_,_).
order(X,Y) :- pos(X,_,_), skolem(Y).
order(X,Y) :- skolem(X), skolem(Y), X < Y.

{meta(precon,P1,P2,P3)} :- order(P1,P3), unary(P2,X), deduced(P3,X,Y).
{meta(postcon,P1,P2,P3)} :- order(P1,P2), deduced(P2,X,Y), unary(P3,Y).
{meta(chain,P1,P2,P3)} :- order(P1,P2), order(P1,P3), deduced(P2,X,Z),
                                                      deduced(P3,Z,Y).
{meta(tailrec,P1,P2,n)} :- order(P1,P2), deduced(P2,X,Z), deduced(P1,Z,Y).

deduced(P1,X,Y) :- meta(precon,P1,P2,P3), unary(P2,X), deduced(P3,X,Y).
deduced(P1,X,Y) :- meta(postcon,P1,P2,P3), deduced(P2,X,Y), unary(P3,Y).
deduced(P1,X,Y) :- meta(chain,P1,P2,P3), deduced(P2,X,Z), deduced(P3,Z,Y).
deduced(P1,X,Y) :- meta(tailrec,P1,P2,n), deduced(P2,X,Z), deduced(P1,Z,Y).

deduced(P,X,Y) :- binary(P,X,Y).

:- not deduced(P,X,Y), pos(_,P,X,Y).
```

```
:- #count{ M,P1,P2,P3 : meta(M,P1,P2,P3) } != N, size(N).
```

Metagol Input Program

```
metagol:functional.

func_test(Atom,PS,G):-
  Atom = [P,A,B],
  Actual = [P,A,Z],
\+ (metagol:prove_deduce([Actual],PS,G),Z \= B).

metarule(precon,[P,Q,R],([P,A,B]:-[[Q,A],[R,A,B]])).
metarule(postcon,[P,Q,R],([P,A,B]:-[[Q,A,B],[R,B]])).
metarule(chain,[P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).
metarule(tailrec,[P,Q],([P,A,B]:-[[Q,A,C],[P,C,B]])).

prim(pour_tea/2).
prim(pour_coffee/2).
prim(move_right/2).

prim(wants_tea/1).
prim(wants_coffee/1).
prim(at_end/1).

a :-
  train_exs(Pos),
  Neg = [],
  learn(Pos,Neg).


pour_tea([robot_pos(X),end(Y),places([place(X,A,cup(up,empty))|R])],
                    [robot_pos(X),end(Y),places([place(X,A,cup(up,tea))|R])]).
pour_tea([robot_pos(X),end(Y),places([E|R1])],
                                    [robot_pos(X),end(Y),places([E|R2])]) :-
pour_tea([robot_pos(X),end(Y),places(R1)],[robot_pos(X),end(Y),places(R2)]).

pour_coffee([robot_pos(X),end(Y),places([place(X,A,cup(up,empty))|R])],
                  [robot_pos(X),end(Y),places([place(X,A,cup(up,coffee))|R])]).
pour_coffee([robot_pos(X),end(Y),places([E|R1])],
                                    [robot_pos(X),end(Y),places([E|R2])]) :-
pour_coffee([robot_pos(X),end(Y),places(R1)],[robot_pos(X),end(Y),places(R2)]).

move_right([robot_pos(X1),end(Y)|R],[robot_pos(X2),end(Y)|R]) :-
                                                        X1 < Y, X2 is X1 + 1.

wants_tea([robot_pos(X),end(_),places([place(X,tea,_)|_])]).
wants_tea([robot_pos(X),end(Y),places([_|R])]) :-
                                    wants_tea([robot_pos(X),end(Y),places(R)]).

wants_coffee([robot_pos(X),end(_),places([place(X,coffee,_)|_])]).
wants_coffee([robot_pos(X),end(Y),places([_|R])]) :-
                                  wants_coffee([robot_pos(X),end(Y),places(R)]).
```

```
at_end([robot_pos(X),end(X)|_]).
```

Sample Instance and Solution

```
Instance:

pos_ex([robot_pos(1),end(3),places([place(1,coffee,cup(up,empty)),
place(2,coffee,cup(up,empty))])],
[robot_pos(3),end(3),places([place(1,coffee,cup(up,coffee)),
place(2,coffee,cup(up,coffee))])]).

pos_ex([robot_pos(1),end(6),places([place(1,coffee,cup(up,empty)),
place(2,coffee,cup(up,empty)),place(3,coffee,cup(up,empty)),
place(4,tea,cup(up,empty)),place(5,coffee,cup(up,empty))])],
[robot_pos(6),end(6),places([place(1,coffee,cup(up,coffee)),
place(2,coffee,cup(up,coffee)),place(3,coffee,cup(up,coffee)),
place(4,tea,cup(up,tea)),place(5,coffee,cup(up,coffee))])]).

Solution:

robot(A,B):-robot_1(A,B),at_end(B).
robot(A,B):-robot_1(A,C),robot(C,B).
robot_1(A,B):-robot_2(A,C),move_right(C,B).
robot_2(A,B):-wants_tea(A),pour_tea(A,B).
robot_2(A,B):-wants_coffee(A),pour_coffee(A,B).
```