FAKULTÄT FÜR !INFORMATIK

Towards A Characterization Of Semi-Stable Models In The Logic Of Here-And-There

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science (Computational Logic) MSc

im Rahmen des Studiums

DDP Computational Logic

eingereicht von

João Manuel Gomes Moura

Matrikelnummer 0827802

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Michael Fink

Wien, 9. September 2009 _____ _____
(Unterschrift Verfasser) (Unterschrift Betreuer)

Towards A Characterization Of Semi-Stable Models In The Logic Of Here-And-There*

João Manuel Gomes Moura

September 9, 2009

*I would like to thank both Prof. Thomas Eiter and Prof. Michael Fink for all the fruitful discussions. Indeed, thanks to their supervision, I dare say that I have almost gone through my stay in Vienna without any major problems. Their observations and suggestions have guided me throughout the M.Sc. thesis and shed light into my doubts. Unquestionably, the best parts of this work have their fingerprints. They taught me how to do scientific research and have polished the roughness of my writings, specially the technical part.

Would also like to thank my EMCL colleagues back in Lisbon for lending me some motivation both for last and the present year. A special thank you goes to Professors Carlos Damasio and Luis Moniz Pereira for encouraging me to join the program in the first place.

Also to my family and friends across Europe for being there for me and for helping making this experience of living abroad a very good one.

This work was supported by the Austrian Science Fund (FWF) project P20841 and by the Vienna Science and Technology Fund (WWTF) under grant ICT 08-020

Abstract

The main motivation for any paraconsistent logic is the idea that reasoning with inconsistent information should be allowed and possible in a controlled and discriminating way. The principle of explosion makes this inviable, and as such must be abandoned. In non-paraconsistent logics, only one inconsistent theory exists: the trivial theory that contains every sentence as a theorem. Paraconsistent logic allows distinguishing between inconsistent theories and to reason with them. Sometimes it is possible to revise a theory to make it consistent, however in other cases (e.g., large software systems) it is currently impossible to attain consistency. Some philosophers and logicians take a radical approach, holding that some contradictions are true, and thus a theory being inconsistent is not something undesirable.

We investigate possible characterizations of existing semantics for paraconsistency using semantic structures that have been proposed in non-monotonic logic programming more recently, while seeking for possible ways of implementation by means of transformation to standard non-monotonic logic programming. In this way, we characterize and present a new way of calculating the semi-stable models of a program (which are paraconsistent in the presence of incoherence) without having to explicitly perform a syntactical transformation as the ones in the characterizations available in the literature. We do this by dealing with strong negation and then calculating the program's Routley models. Afterwards we only need to perform a selection according to some criteria.

Keywords Paraconsistency, inconsistency, incoherence, semi-stable models, Routley models, here-and-there models.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Introduction

This section is divided in a few small subsection from which, in subsection 1.2 we present a motivation for this work while in subsection 1.3 we list a few projects where the same sort of problems stated in subsection 1.4 were addressed in the past.

1.2 Motivation

On modular logic programs where multiple persons introduce different and possibly conflicting knowledge, inconsistencies or incoherences can show up.

Example 1.1 *A program P is incoherent if in some rule a literal is implied by its default negation:*

$$P = \{\ a \ \leftarrow \ not\ a.\ \}$$

While a program P is inconsistent if both an atom and its strongly negated version are implied:

$$P = \{\ \begin{array}{l} a. \\ \neg a.\ \} \end{array}$$

Some of the usual assumptions taken in traditional ASP must be withdrawn, by generalizing modular answer-set programming, if one wants to allow multiple programs exchanging data. Since multiple programs are involved, possibly built by different people, inconsistencies will most likely arise. In this case there are better ways than to declare the whole composition as inconsistent. Thus, some form of paraconsistent reasoning (see, e.g., [Ari02, DLMPea, Sak92, SI95]) should be introduced in such a framework allowing modular logic programming.

In [PT09], a good motivation for paraconsistency is presented pretty much as follows: The reasons for paraconsistency that have been suggested in the literature appear to be connected to the development of specific formal systems of paraconsistent forms of logic. There are, however, several reasons for considering that some logics should be paraconsistent. The primary motivation for such logics is the conviction that it should be possible to reason with inconsistent information in a controlled and discriminating way. The principle of explosion makes this impossible and as such must be abandoned. In standard non-paraconsistent logics, there is only one inconsistent theory: the trivial theory; where every sentence holds as a theorem. Paraconsistent logic makes it possible to distinguish between inconsistent theories and to reason with them. Sometimes it is possible to revise a theory and make it consistent (Belief revision). In other cases (e.g., large software systems) it is currently impossible, due to computational complexities involved, to achieve consistency.

We must highlight the fact that there are theories which are inconsistent but still non-trivial as one of the main reasons for paraconsistent logics. Once we admit the existence of such theories, their underlying logics must be paraconsistent. Examples of inconsistent but non-trivial theories are easy to produce and we present some interesting examples ahead. A very interesting example is presented by the authors, which can be derived from the history of science.

- Consider Bohr's theory of the atom. According to this, an electron orbits the nucleus of the atom without radiating energy. However, according to Maxwell's equations, which formed an integral part of the theory, an electron which is accelerating in orbit must radiate energy.

- Hence Bohr's account of the behavior of the atom was inconsistent. Yet, patently, not everything concerning the behavior of electrons was inferred from it, nor should it have been. Hence, whatever inference mechanism that was used, it must have been paraconsistent.

Paraconsistent logic is motivated by philosophical considerations as well as by its applications and implications. One of the applications is automated reasoning (information processing) for which another example is presented by the authors:

- Consider a computer which stores a large amount of information. While the computer stores the information, it is also used to operate on it, and, crucially, to infer from it. Now it is quite common for the computer to contain inconsistent information, because of mistakes by the data entry operators or because multiple sources are involved.

- This is certainly a problem for database operations with theorem-provers, and so has drawn much attention from computer scientists. Techniques for removing inconsistent information have been investigated. Yet all have limited applicability, and, in any case, are not guaranteed to produce consistency. (There is no algorithm for logical falsehood.)

Hence, even if we take some measures to deal with contradictions when we find them, an underlying paraconsistent logic is desirable if hidden contradictions are not to gen-

erate spurious answers to queries.

We will develop an approach for handling inconsistency in knowledge bases based in logic programs. To this aim, we will establish theoretical results on alternative semantics (compared to the exact answer-set semantics), where we plan to utilize techniques from paraconsistent reasoning in logic programming [DLMPea, Sak92, SI95] and epistemic logic.

1.3 Background and previous projects

In [OP05], Odintsov and Pearce introduce the notion of strong equivalence for programs with paraconsistent answer-set semantics (not only programs but propositional theories to be precise) and show that Routley models exactly capture strong equivalence for these paraconsistent answer-set (PAS) programs. They have also shown how both ordinary and paraconsistent answer-sets can be captured via possible worlds models, devised by Routley.

In [ADP05], the authors defined frames based on point sets to capture and characterize PAS semantics. They introduce point sets as $P = \langle Q, \leq \rangle$, with Q a set and $\leq$ a partial ordering on Q. Propositions on P are upwards closed subsets of Q. Frames are point sets together with accessibility relations.

1.4 Research issues and questions

If one aims at developing a modular, possibly distributed, paraconsistent semantics for dealing with incoherencies and inconsistencies that may arise from combining different programs, several issues need to be resolved that currently pose challenging research problems:

1. **How to deal with inconsistency?** Combining multiple programs may lead to inconsistent information. This is an important problem when it comes to how different modules are programmed in a modularized program composition. Usually, a team of programmers builds a program and the knowledge encoded this way is in some way coordinated, but taking a distributed approach it is possible and even sometimes desirable that some program module can be created by an a priori unknown user, which means that conflicting information may arise when combining multiple knowledge bases or programs. While this is tightly connected to the consistency checking reasoning task, it gives rise to the interesting question of whether it is possible to relax the definition of consistency, i.e., 'hiding' conflicting information to gain consistent program modules.

   The inconsistency issue can for instance be tackled by weakening the semantics, i.e., by using a semantic approximation of a program's answer-sets. For example, this could be done by using partial models, as defined by the well-founded semantics, instead of the usual answer-set semantics. We might also make a distinction between local program modules' models and the overall program semantics of the compound program. Here, the idea is that local models, rather

than global models, could be considered relevant for some application in some context. Additionally, techniques for paraconsistent reasoning, such as the ones that rely on multi-valued semantics, might help dealing with inconsistency (we refer the reader to [Ari02, DLMPea, Sak92, SI95]) from which we highlight Inoue and Sakama's semi-stable models as the ones obviously of greater relevance to us.

2. **Which models can capture incoherence and can thus be considered paraconsistent? Well founded semantics or answer-set semantics?** Due to the extension of the topic (close to twenty semantics have been described in [DLMPea]) we organize the following short discussion in a few general features that we consider being the most important:

   Semantical Concepts - We can pin the first studies of extended logic programming to the works of Blair and Subramanian on paraconsistent logic programs [BS87]. They have shown that their Generalized Horn Programs are equivalent to extended logic programs without default negated literals. This is a common basis on which almost all extended logic programming semantics agree, and has its roots in the works on paraconsistent constructive logics [Nel49]. Belnap's logic provides the underlying model theory for this semantics.

   With the introduction of default negation some problems, and different views on how to solve them appeared. Two main different approaches are identifiable, the coherence view and the weak negation view.

   - The first adopts the coherence principle [Mon92] relating the two forms of negation as basic and provides a localized explosion of consequences when faced with contradiction. $WFSX_p$ and its extensions are the amongst the only representative of this type of semantics.
   - The second view sustains more or less firmly the complete independence of an atom A from its explicit negation $\neg A$. The authors state that every other semantics that they are aware of embrace it.

   As for the basic inference relation for the weak negationist view, it can be attributed to Wagner's liberal reasoning, except for his conservative, credulous, and skeptical forms of reasoning. As in standard normal logic programming semantics, the differences mainly lie in the way we can take care of infinite negative recursions. The well-founded semantics like the ones in [PM93, Sak92] assign the logical value undefined to literals involved in such recursions. Przymusinski's semantics has an explosive behavior in face of contradiction, while Sakama's extended well-founded semantics does not.

   The followers of Weak negation have proposed several non-explosive semantics (e.g. [SI95]) which are variants of the answer-sets semantics [GL90]. To overcome the problem of non-existing models for some extended logic programs, Sakama and Inoue defined the semi-stable model semantics, which suffers from problems in the treatment of undefined literals manifested in the need of having different languages for the program itself and for it's models.

An important remark that we must make is that most of the paraconsistent semantics proposed for extended logic programs use fairly standard techniques of normal logic programming semantics. First, the semantics is defined for the default negation free case. Then, if the behavior of default negation is to be similar to the one in Well Founded Semantics, an alternating fixpoint definition can be defined. If the *not* is like the one in stable-models, then a corresponding fixpoint equation is used to define the semantics for the general case. The notable exception, and also the one of greater interest to us, is semi-stable semantics which is based on a semantics for disjunctive logic programs.

It is sometimes necessary to block the propagation of contradiction, besides detecting unsafe conclusions. An argumentation approach to this problem is adopted by Wagner in [Wag93] with his credulous, conservative and skeptical inference relations. The last two are in themselves consistent. However, all three inference relations do not obey to Modus Ponens and are not reflexive.

The family tree of the semantics surveyed in Damasio and Pereira's paper [DLMPea] can be found in Figure 1.1. The non-obvious abbreviations on that figure that are of relevance to us are as follows; PSM, WAS and SE stand for respectively paraconsistent stable models, weak answer-sets and stable environments. Suspicious well-founded and p-stable models are shortened to WFS^s and PSM respectively. The well founded and stable structures take the names WS and SS respectively.

Still dealing with the question of which models can capture incoherence and can be considered paraconsistent, a particular question must be posed:

- **Can Routley models capture incoherence?** Roughly speaking, incoherency arises when a literal is implied by its default negation in a program. Since incoherency is viewed as a kind of inconsistency, it is desirable to provide a framework which is paraconsistent for such incoherency. In the case of PAS, the underlying logic $N9$ introduced in [OP05] belongs to the lattice of logics studied by Odintsov [Odi05].

  In [SI95] the authors introduce, in order to present incoherent facts, five extra truth values bt, bf, $b\top$, tcb, and fcb which respectively denote believed true, believed false, believed contradictory[1], true with contradictory belief and false with contradictory belief. These values together with the values in their logic IV constitute a lattice of nine-valued logic IX such that $\bot \preceq bx \preceq x \preceq xcb \preceq \top$ and $bx \preceq b\top \preceq xcb$ for $x \in t, f$. The set of truth values of their four-valued logic is defined as $IV = t, f, \top, \bot$, in which t, f, $\top$, $\bot$ are propositions over the language of a program and respectively denote true, false, contradictory, and undefined as can be seen in Figure 1.2 on Page 24.

  So, apparently the two lattices are compatible even though at this point more study and some work must be put into it.

[1] Usually, the truth value contradictory is represented with $\bot$ but the authors chose to use $\top$.

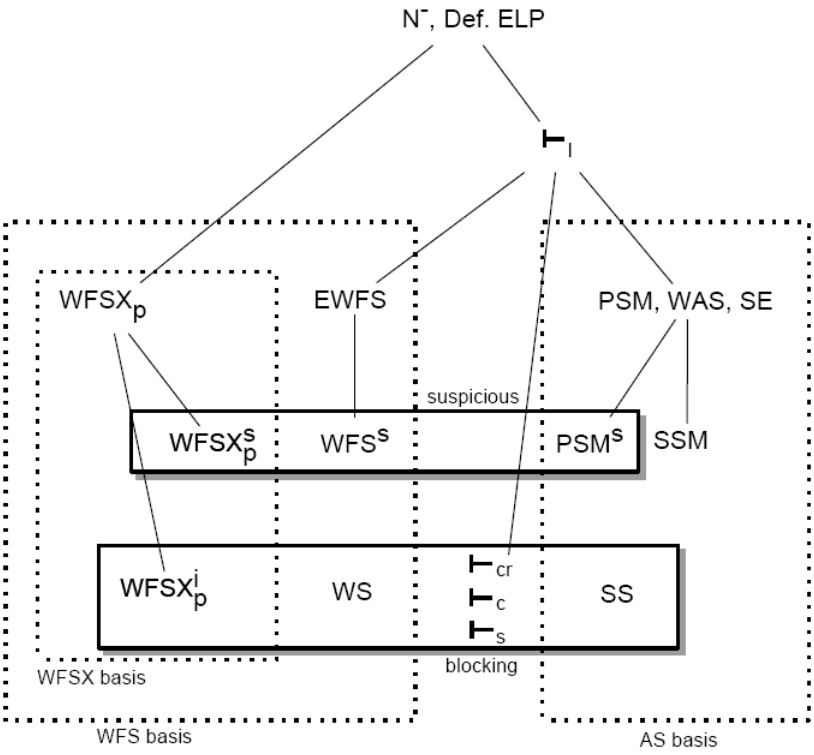

Figure 1.1: Paraconsistent semantics genealogy

3. **Is a Language transformation enough?** A language transformation might allow us to deal with incoherence through a 'syntactical transformation'. In fact, [SI95] introduces an epistemic transformation along the way, including an operator K similar to the modal operator K in epistemic logic that is used for capturing that a given literal is *believed*, for reducing programs with default negation to positive programs. Thus, the original set of literals is expanded with the set of literals that are believed to be true.

   This raises the following question:

   - **Can we calculate semi-stable models without having to perform an explicit program transformation?** Improving results presented in [SI95], the present work has the goal to provide a definition of semi-stable models grounded in logical terms. Unlike the original definition, this should be attained without employing any kind of logic syntactic transformation over the original program at hand.
   - **Can we have such a characterization of semi-stable models in terms of here-and-there models?** This question posed itself along the way as we were looking for a good way to characterize semi-stable models and

we were able to conjecturize the characterization the we present in Proposition 2.10 after looking at some examples. So, yes, if a program has a semi-stable model then there are always semi-stable here-and-there models that embed it and a characterization is possible.

1.5 Contributions

We provide several embeddings of here-and-there and Routley models and we prove their equivalences. Furthermore, towards a new characterization of semi-stable models in terms of here-and-there models, we provide in Definition 2.1 a simple way of dealing with strong negation that suits our purposes and we reuse the way Sakama and Inoue deal with default negation by means of the epistemic transformation described in Definition 1.9. The prime transformation and its modified version that we present in 2.1 will allow us to compute models containing both L and L' (instead of $\neg L$), that would otherwise be inconsistent. We define a seven-valued interpretation for the language that derives from these transformations in 2.2.

In 2.3, to an interpretation I'^{κ} of a transformed program P'^{κ} we define its equivalent here-and-there model.

Then we state a fixpoint equivalence in Lemma 2.1. The fixpoint semantics that we will be using to calculate both semi-stable models and answer-sets reuses the definition presented in [SI95] and characterizes operational aspects of logic programs. This is also implemented using bottom-up model generation techniques as presented in [Sak92].

Having all the auxiliary definitions in place, we moved to stating an equality between a transformed program's minimal sets calculated with the fixpoint operator $\mu(\tau_{P'^{\kappa}} \uparrow w)$ and its answer-sets. More formally: $AS(P'^{\kappa}) = min(\mu(\tau_{P'^{\kappa}} \uparrow w))$.

Having this, we can generalize the previous lemma. Then, having a transformed program, its answer-sets will coincide with its semi-stable models after selecting the maximally canonical interpretations obj'^{κ}_{mc}. As such, $obj^{\kappa'}_{mc}(AS(P^{\kappa'})) = SST_{P'^{\kappa}}$. Proposition 2.3 follows trivially from Lemma 2.2 and from our definition of semi-stable (SST) models as $SST_{P'^{\kappa}} = obj^{\kappa'}_{mc}(min(\mu(\tau_{P^{\kappa'}} \uparrow \omega)))$. This is our first major result - A characterization of semi-stable models of a modified program as its maximally canonical answer-sets.

In Subsection 2.2 we provide several embeddings, correspondences and definitions revolving around here-and-there and Routley models of transformed programs before we present our second major result - A fully declarative new characterization of semi-stable models in the logic of here-and-there.

In Section 2.4 we will also describe two prototypical implementations of our two characterizations of semi-stable models.

1.6 Road map

Still in Part 1, in Section 1.8 we present a comprehensive introduction and establish a framework for the research task at hand.

In Part 2 we resent our contributions, divided in the following sections: Section 2.1 where we, besides introducing the problem of dealing with both default and strong negation, present our contribution towards dealing with them; In Section 2.2 we provide several embeddings for the different models we have at hand as well as preparatory results that we need for Section 2.3 where we finally present a new characterization of semi-stable models in terms of here-and-there models.

Part 3 is divided in Section 3.2 where we will present some complexity results for paraconsistent logics and a complexity study for our contributions. In Section 3.2 we discuss the work presented and we outline possible guidelines for future work. Finally, in Section 3.1 we provide a summary of relevant related work.

1.7 Constraint types, inconsistency and incoherence

In answer-set programming, we can write constraints both in the canonical way e.g.,

$$\{p \leftarrow a, b, not\ p.\}$$

as well as in a simplified way:

$$\{\leftarrow a, b.\}$$

Under the usual stable model semantics they yield the same result - eliminating models containing all the positive literals in the constraint's body. If we evaluate the same programs under the semi-stable semantics, the classical constraints are transformed by the epistemic transformation but not its simplified version. This way, under the semi-stable semantics, the two types of constraints must be distinguished. We can clearly see this in a good motivating example - Example 1.2:

Example 1.2 *(For checking incoherence without 'simplified constraints')*

$$P_1 = \{a. \quad p \leftarrow a, not\ p.\}$$

- (a, ap) SST *model* $\{a, Ka, Kp\}$
- (ap, ap)

This program, having a classic constraint, has no stable-models but still allows one semi-stable model. The program with the simplified version of the same constraint has no stable nor semi-stable models:

$$P_2 = \{a. \quad \leftarrow a.\}$$

As a result, the simplified writing of a constraint does not yield the same models as the extended one. As for its Routley models, for $T \vDash P$ to hold, the *there* part of the model must contain $\{a\}$ but this is disallowed by the constraint. Therefore, there are no candidate Routley interpretations and as such no Routley models.

Roughly speaking, incoherency arises when a literal is implied by its default negation in a program. Since incoherency is viewed as a kind of inconsistency, it is desirable to provide a framework which is paraconsistent for such incoherency. In Section 1.8, we described the notion of semi-stable models which are paraconsistent for incoherent programs [SI95].

Despite the fact that dialetheism and paraconsistency needs to be distinguished, dialetheism can be a motivation for paraconsistent logic. If there are true contradictions (dialetheias), i.e. there are sentences such that both A and $\neg A$ are true, then some inferences of the form $\{A, \neg A\} \models B$ must fail. Because only true and not arbitrary conclusions follow validly from true premises, logic has to be paraconsistent. One candidate for a dialetheia is the liar paradox. Consider the sentence: 'This sentence is not true'. There are two options: either the sentence is true or it is not. Suppose it is true. Then what it says is the case. Hence the sentence is not true. Suppose, on the other hand, it is not true. This is what it says. Hence the sentence is true. In either case it is both true and not true making it so that non paraconsistent semantics will trivialize the program.

1.8 Preliminaries

Lets start with a short foreword about *Model Theory*. Generally speaking, a set of sentences in some logic is called a theory. A theory T is satisfiable if it has a model $M \models T$, i.e. a structure (of the appropriate signature) which satisfies all the sentences in the set T. Consistency of a theory is usually defined in a syntactical way, but in first-order logic by the completeness theorem there is no need to distinguish between satisfiability and consistency. Therefore model theorists often use 'consistent' as a synonym for 'satisfiable'.

Logic of here-and-there

Linear, rooted frames with two nodes are also called here-and-there frames. They characterize a super-intuitionistic logic called here-and-there, in short HT, and sometimes referred to as Gödel's 3-valued logic.

In the semantics for intermediate or super-intuitionistic logics, the so-called logic of here-and-there can be captured by rooted frames with two elements, commonly denoted by h and t and called 'here' and 'there', with h $\subseteq$ t. As presented by Pearce and Odintsov [OP05], if we work in the lattice of extensions of Nelson's logic N^-, we can consider here-and-there models as a special kind of Routley models. Therefore, a Routley here-and-there model can be represented as a Routley model $M = (W \cup W^*, \subseteq, *, V)$, where $W = \{h, t\}$ comprises two worlds 'here' and 'there', such that h $\subseteq$ t, and $W^* = \{h^*, t^*\}$ comprises the 'starred' worlds. It follows that $t^* \subseteq h^*$.

Now, a total model that is minimal over the here-and-there frames has been called an equilibrium model. Evidently, if a model M is not an equilibrium model, there is a smaller HT-model which clearly also models the same program; hence M is not stable. While if M is not stable, there is a smaller HT-model, and so M is not in equilibrium. So we can conclude that stable models and equilibrium models coincide.

We will often use (H, T) as notation for a here-and-there interpretation and $HT(P)$ for a here-and-there model of a program P. Here, H corresponds to the here part and T to the there part of the model.

The answer-set programming paradigm

T. Eiter et al. present a very comprehensive summary of the answer-set programming paradigm in their description of the project Modular HEX-Programs [ea08]. We adapt and summarize it in the following: Over the last few years, the answer-set Programming (ASP) paradigm [EFL$^+$01, Tru04, Lif02, MT99, Nie98] developed as one of the most important methods for declarative knowledge representation and reasoning.

This approach is based in semantic notions and there are several methods to compute models. More specifically, problems are represented in terms of nonmonotonic logic programs, such that models of the latter compose solutions for the original problem. This is, in some way, similar to the method of reducing problems to Satisfiability Solving (SAT) but with clear advantages for certain instances. In this context, the most commonly used model notions are the ones of stable models [GL88] or, as a generalization, the answer-sets of a (possibly disjunctive) logic program [GL91b].

Both notions are in their basis nonmonotonic because the set of logical consequences from all stable models (respectively, all answer-sets) in general does not necessarily grow monotonically with increasing information. This can happen because of the usage of the negation as failure operator. Contrasting with semantics containing strong procedural elements, like the cut operator in Prolog, they are fully declarative. answer-set semantics extends stable model semantics to a syntactically richer class of logic programs. The answer-set semantics is defined for Extended Logic Programs (ELPs), in which negation as failure can occur in program rules, as well as often named classical negation - strong negation - and disjunctions. The stable model semantics, instead, is associated with normal logic programs (NLPs), which contain only negation as failure as a basic operator.

In general, ASP as a formalism suits the task of handling inconsistent and incomplete information, as well as the one of capturing nondeterministic features. However, the major problem of the usual answer-set semantics is that the answer-set becomes trivial in case of an inconsistent program and every formula can be implied from the program. This also happens with most of the traditional logic programming semantics in which one local inconsistency can make the whole program unusable.

Positive Extended Disjunctive Programs

A positive extended disjunctive program is a finite set of clauses of the form:

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \qquad (m \geq l \geq 0)$$

where L_i's are positive or negative literals. When L is a positive (resp. negative) literal, $\neg L$ denotes its complementary negative (resp. positive) literal and $L = \neg\neg L$ holds as usual. The left-hand side of the clause is called the head and the right-hand side of the clause is called the body. A clause is called disjunctive if its head contains more than one literal. A clause is called an integrity constraint if it has the empty

head and a non-empty body. A positive extended disjunctive program containing no disjunctive clause is called a positive extended logic program. A positive extended disjunctive program is called a positive disjunctive program if all L_i's are atoms.

Stable model semantics

In the case of disjunctive programs, we say that a set I of atoms is a stable model of P if I is minimal (with respect to set inclusion) among the models of the reduct P^I.

In *Sixty Years of Stable Models* [Pea08], Pearce gives a very good overview of Stable models which we next adapt to our purposes: The author considers two different techniques for studying the mathematical foundations of stable reasoning and ASP.

- One of these is based on classical, propositional and predicate logic. Its main advantages are its familiarity to logic programming users, its wealth of results and the fact that it is very suitable for rapid prototyping. Its drawbacks start with the fact that it lies, in a sense that can be made precise, one level removed from the action. We first have to translate, manipulate and modify, before we obtain relevant representations in classical logic that we could have obtained in simpler fashion using a non-classical logic. This need not but sometimes can add an ad hoc flavor to the modeling unless it is carefully spelt out why certain features are attached to the formalism at hand. Moreover in some cases it can add an unnecessary layer of complexity that also increases the difficulty of establishing properties and theorems.

- The second approach to understanding stable models is a more direct and immediate one. It it characterized by the usage of a non-classical logic in which stable models can be represented as minimal models as well as for the fact that formulas, programs and theories that are in a robust sense equivalent under stable reasoning can be shown to be logically equivalent. As Pearce shows in the remainder of his paper, by using logical and meta-logical results this approach to the foundations of stable reasoning has considerable explanatory power.

Usual definitions

The minimal models of a definite P can be computed (bottom-up) via operator T_P and they correspond to the stable models of P:

Definition 1.1 (T_P operator (2-valued case)) *Let I be an interpretation of definite P.* $T_P(I) = \{H : (H \leftarrow Body) \in P \wedge Body \subseteq I\}$

If P is definite, T_P is monotone and continuous. Its minimal fixpoint can be built by: $I_0 =$ and $I_n = T_P(I_{n-1})$ with $n > 0$.

The least model of a definite P is $T_P^{\uparrow\omega}(\{\})$ i.e., the limit that is achieved by iterating operator T ω times.

A fixpoint semantics for stable models Let's first start by introducing some notation:

- Let P be a disjunctive logic program. For each rule r in P of the form:

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n \quad (m \geq l \geq 0) \tag{1}$$

  - $Head_P(r) = \{L_1, \ldots, L_l\}$ is the set containing all the literals in the head of (1).
  - $Body_P^+(r) = \{L_{l+1}, \ldots, L_m\}$ is the set containing all the positive literals in the body of (1).
  - $Body_P^-(r) = \{L_{m+1}, \ldots, L_n\}$ is the set containing all the negative literals in the body of (1).
  - $Body_P^*(r) = \{L_{m+1}, \ldots, L_n\}$ is the set containing all literals in the body of (1).

- If a program is positive we will omit the superscript $^+$ in $Body_P^+(r)$ and write simply $Body_P(r)$. Also, if the context is clear enough we will omit the subscript mentioning the program and write simply $Head(r)$ and $Body(r)$.

Definition 1.2 ([LRS97]) *Let I be an interpretation for a program P. A set $X \subseteq Body_P$ of ground atoms is an* unfounded set *for P w.r.t I if, for each $a \in X$, for each rule $r \in ground(P)$ such that $a \in Head(r)$, at least one of the conditions holds:*

1. *$Body^*(r) \cap \neg.I \neq \emptyset$, that is, the body of r is false w.r.t. I.*[2]
2. *$Body^+(r) \cap X \neq \emptyset$, that is, some positive body literal belongs to X.*
3. *$(Head(r) - X) \cap I \neq \emptyset$, that is, an atom in the head of r, distinct from a and other elements in X, is true w.r.t. I.*

Conditions 1 and 2 are the same as in the classical definition of unfounded sets [GRS91]. Intuitively, the third condition expresses that an atom a, occurring in the head of rule r, is not derivable from r if the head of r is already true; in other words, there exists an atom b in the head of r which is true in I (indeed, inferences follow a minimality criterion). However, the truth of the atom b in the head is not taken into account (for the unfoundedness of a) if b itself is unfounded, that is, $b \in X$.

Definition 1.3 ([LRS97]) *Let I be an interpretation for a program P. Then I is* unfounded-free *if $I \cap X = \emptyset$ for each unfounded set X for P w.r.t. I.*

In traditional logic programming, the union of all unfounded sets w.r.t. an interpretation I is also an unfounded set w.r.t. I (called the greatest unfounded set) that includes all unfounded sets w.r.t. I [GRS91]. There, the authors show that in disjunctive logic programming, this is not generally true. They denote by $\mathbf{I}_P$ the set of all interpretations of P which possess this property.

Then Leone et al. state the following proposition:

[2] $\neg.I$ is the set formed by individually applying $\neg$ to each literal in I.

Proposition 1.3 ([LRS97]) *Let I be an unfounded-free interpretation for a program P. Then*

a *P has the greatest unfounded set $GUS_P(I)$ (i.e., $I \in \boldsymbol{I}_P$), and*

b *$GUS_P(I)$ is computable in polynomial time if P is a propositional program.*

As for the fixpoint semantics for stable models, the authors start by providing an extension of the immediate consequence operator T_P defined in [GRS91] for three-valued interpretations of normal logic programs to disjunctive logic programs.

Definition 1.4 ([LRS97]) *Let P be a program. Define the T_P operator as follows:*
$T_P : 2^{Body_P \cup \neg.Body_P} \rightarrow 2^{Body_P}$

$$I \mapsto \{a \in Body_P \mid \exists r \in ground(P) \text{ s.t. } a \in Head(r), Head(r) - \{a\} \subseteq \neg.I, \text{ and } Body(r) \subseteq I\}.$$

Intuitively, given an interpretation I, T_P derives a set of atoms belonging to every model containing I (i.e. atoms that are surely needed to extend I to a model). Note that, unlike other extensions of T_P to disjunctive logic programs, T_P is deterministic. That is, its result is a single set of atoms rather than a family of sets of atoms.

Definition 1.5 ([LRS97]) *Let P be a program. Define the W_P operator as follows:*[3]

$$W_P\text{: } \boldsymbol{I}_P \rightarrow 2^{Body_P \cup \neg.Body_P}$$
$$I \mapsto T_P(I) \cup \neg.GUS_P(I).$$

The next proposition confirms the intuition that Definition 1.5 extends the W_P operator defined in [GRS91] for disjunction-free programs (whose least fixpoint is the well-founded model) to disjunctive logic programs.

Proposition 1.4 ([LRS97]) *Let P be a disjunction-free program. Then the W_P operator of Definition 1.5 exactly coincides with W_P operator defined in [GRS91].*

Program reduct

Definition 1.6 (Program reduct) *Let P be an extended disjunctive program and I be a subset of $\mathcal{L}_P$. The reduct of P with respect to I is the positive extended disjunctive program P^I such that a clause*

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m$$

is in P^I if and only if there is a ground clause of the form

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n \ (l \geq 0)$$

from P such that $\{L_{m+1}, \ldots, L_n\} \cap I = \{\}$.

Notice that:

[3] Recall that $\mathbf{I}_P$ is the set of interpretations having the greatest unfounded set.

- A literal L is true in an HT-model (H, T) just in case $L \in H \bigcup -\overline{H^*}$.[4]
- A Routley model M can be denoted only by (H, T) with $H = (H, H^*)$ and $T = (T, T^*)$ if the program it models is positive.
- The set I in Definition 1.6 is called a paraconsistent answer-set of P (Section 1.8) if and only if the sets $i = H \bigcup -\overline{H^*}$ and $j = T \bigcup -\overline{T^*}$ are models of the reduct P^I.

Relation to other theories of negation as failure

Program completion According to [Marek and Subramanian, 1989], any stable model of a finite ground program will be not only a model of the program itself, but also a model of its completion. However, the converse is not true. For instance, the completion of a program with only one-rule

$$\{p \leftarrow p.\}$$

is the tautology $p \leftrightarrow p$. This tautology's model $\varnothing$ is stable, but its other model $\{p\}$ is not. In 1994, François Fages found a syntactic condition on logic programs that is able to eliminate such counterexamples and ensures the stability of every model of the program's completion. The programs that satisfy his condition are called *tight*.

Then, in 2004, Fangzhen Lin and Yuting Zhao showed how to obtain a stronger completion of a non-tight program so that all its nonstable models are eliminated. The additional formulaes that they added to the completion are called *loop formulas*.

Well-founded semantics The well-founded model of a logic program splits all ground atoms into three sets. They correspond to true, false and unknown. If an atom is true in the well-founded model of P then it belongs to every stable model of P. The converse however, generally, does not hold. For instance, the program:

$$\{p \leftarrow not\ q.\ \ q \leftarrow not\ p.\ \ r \leftarrow q.\ \ r \leftarrow p.\}$$

has two stable models, $\{p, r\}$ and $\{q, r\}$. Even though r belongs to both of them, its value in the well-founded model is unknown. Furthermore, if an atom is false in the well-founded model of a program then it does not belong to any of its stable models. Thus the well-founded model of a logic program provides a lower bound on the intersection of its stable models and an upper bound on their union.

Equilibrium logic

Equilibrium Logic [Pea97] is a logical characterization of the stable models (or answer-set) semantics for logic programs [GL88]. As shown in [LPV00], it allows capturing the important property of strong equivalence of logic programs, that is, when a piece of program can be safely replaced by another regardless the context they are included

[4] $\overline{H}$ denotes the complement of H.

in. Furthermore, it can also characterize as logical formulas most extensions and syntactic constructions defined for answer-set Programming. In fact, the current most general definitions of stable models for arbitrary propositional or first order theories are equivalent to the definition of Equilibrium Models.

Like answer-sets, Equilibrium Logic is a nonmonotonic formalism, but unlike answer-sets, which are defined in terms of a syntactic program transformation (the Gelfond-Lifschitz program reduct), Equilibrium Logic is defined in terms of a models selection criterion for a monotonic intermediate logic: the logic of here-and-there [Hey30].

Equilibrium models and strong equivalence

A total model that is minimal over the here-and-there frames is called an equilibrium model.

Equilibrium models are special kinds of minimal N_3-models. Let Π be a theory and (H, T) a model of Π. (H, T) is said to be total if $H = T$. (H, T) is said to be an equilibrium model if it is total and there is no model (H', T) of Π with $H' \subset H$. The expression $Eq(V, \Pi)$ denotes the set of the equilibrium models of theory Π on signature V. Equilibrium logic is the logic determined by the equilibrium models of a theory. It generalizes answer-set semantics in the following sense. For all the usual classes of logic programs, including normal, disjunctive and nested programs, equilibrium models correspond to answer-sets. The 'translation' from the syntax of programs to N_3 propositional formulas is the trivial one, eg. a ground rule of a disjunctive program of the form:

$$q_1 \vee \ldots \vee q_k \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ pn$$

where the p_i and q_j are atoms, corresponds to the N_3 sentence

$$p_1 \wedge \ldots \wedge p_m \wedge \neg p_{m+1} \wedge \ldots \wedge \neg p_n \rightarrow q_1 \vee \ldots \vee q_k$$

Following this short introduction, Cabalar, Pearce and Valverde present in [CPV08] two very interesting propositions that illustrate Equilibrium models in a comprehensive way.

Proposition 1 ([Pea97, LPV00]). For any logic program Π, an N_3 model (H, T) is an equilibrium model of Π if and only if T is an answer-set of Π.

Two theories, Π_1 and Π_2 are said to be logically equivalent, in symbols $\Pi_1 \equiv \Pi_2$, if they have the same N_3 models. They are said to be strongly equivalent, in symbols $\Pi_1 \equiv_s \Pi_2$, if and only if for any Π, $\Pi_1 \cup \Pi$ is equivalent to (has the same answer-sets as) $\Pi_2 \cup \Pi$. The two notions are connected as follows:

Proposition 2 ([LPV00]). Any two theories Π_1 and Π_2 are strongly equivalent if and only if they are logically equivalent, ie. $\Pi_1 \equiv_s \Pi_2$ if and only if $\Pi_1 \equiv \Pi_2$.

strong equivalence is important because it allows us to transform programs or theories to equivalent programs or theories independent of any larger context in which the theories concerned might be embedded.

Paraconsistent semantics in logic programming

In [PT09] we can find a very good introduction to paraconsistency and also a good motivation for it

Main stream contemporary logic says that from contradictory premises anything can be inferred. To be more specific, let $\models$ be a relation of logical consequence, either defined semantically or proof-theoretically. Let's call $\models$ *explosive* if it validates $\{A, \neg A\} \models B$ for every A and B (called in philosophical logic as ex contradictione quodlibet (ECQ)). The usual contemporary logic, i.e., classical logic, is explosive but also some 'non-classical' logics such as intuitionist logic and most other standard logics are.

Still in [PT09], they argue that the major motivation behind paraconsistent logic is to challenge this orthodoxy. A logical consequence relation, $\models$, is said to be paraconsistent if it is does not possess the principle of explosion. Thus, if $\models$ is paraconsistent, then even if we are in certain circumstances where the available information is inconsistent, an inference relation does not explode into triviality. Thus, paraconsistent logic accommodates inconsistency in a sensible manner that treats inconsistent information as valid and informative. There are several reasons driving such motivation. The development of the systems of paraconsistent logic has depended on these. This subsection is not meant to be a complete survey of paraconsistent logic because this field is historically vast even if the modern history of paraconsistent logic may be relatively short. Nevertheless, the development of the field has grown to the extent that a complete survey goes beyond the scope of the present thesis. We solely aim to providing some aspects and features of the field that are philosophically noticeable. This does not mean that paraconsistent logic has no mathematical significance or significance in such areas as computer science and linguistics. Indeed, the development of paraconsistent logic in the last two decades or so indicates that it has important applications in those areas. However, we shall tread over them lightly and focus more on the aspects that are of central interest for philosophers and philosophically trained logicians.

Motivation for paraconsistency

Throughout this subsection we will present an adaptation of the introduction in [PT09].

The reasons for paraconsistency that have been put forward appear to be specific to the development of the particular formal systems of paraconsistent logic. However, there are several general reasons for thinking that logic should be paraconsistent. The primary motivation for such logics is the conviction that it ought to be possible to reason with inconsistent information in a way that is controlled but at the same time allows to discriminate such inconsistencies. The principle of explosion makes this impossible and hence must be abandoned.

In non-paraconsistent logics, there is only one inconsistent theory: the trivial theory that has every sentence as a theorem. Paraconsistent logic makes it possible to distinguish between inconsistent theories and to reason with them. Sometimes it is possible to revise a theory to make it consistent. In other cases (e.g., large software systems) it is currently impossible to attain consistency.

One prominent view known as dialetheism is motivated by several considerations, most notably an inclination to take certain paradoxes such as the Liar and Russell's paradox at face value. Not every advocate of paraconsistent logic is a dialetheist. On the other hand, being a dialetheist rationally commits one to some form of paraconsistent logic, on expense of otherwise having to accept everything as true (i.e. trivialism).

Before we summarize some systems of paraconsistent logic and their motivations, we present some general motivations for paraconsistent logic.

Inconsistent but non-trivial theories A most telling reason for paraconsistent logic is the fact that there are theories which are inconsistent but not trivial. Once we admit the existence of such theories, their underlying logics must be a paraconsistent one. Examples of inconsistent but non-trivial theories are easy to produce. An example that can be derived from the history of science (In fact, many examples can be given from this area.) is Bohr's theory of the atom, as presented in Section 1.1.

Dialetheias (true contradictions) Despite the fact that dialetheism and paraconsistency needs to be distinguished, dialetheism can be a motivation for paraconsistent logic. If there are true contradictions (dialetheias), i.e., there are sentences, A, such that both A and $\neg A$ are true, then some inferences of the form $\{A, \neg A\} \models B$ must fail. For only true, and not arbitrary, conclusions follow validly from true premises. Hence logic has to be paraconsistent. One candidate for a dialetheia is the liar paradox. Consider the sentence: 'This sentence is not true'. There are two options: either the sentence is true or it is not. Suppose it is true. Then what it says is the case. Hence the sentence is not true. Suppose, on the other hand, it is not true. This is what it says. Hence the sentence is true. In either case it is both true and not true.

Automated reasoning Paraconsistent logic is motivated not only by philosophical considerations, but also by its applications and implications. One of the applications is automated reasoning (information processing) also called machine learning. Consider a computer which stores a large amount of information. As the computer stores information, it is also used to operate on it and, very importantly, to infer from it. Now, it is very common for such knowledge base to contain inconsistent information, because of mistakes by the data entry operators or because of multiple sourcing. This is certainly a problem for database operations with theorem-provers, and so has drawn much attention from computer scientists. Thus, some techniques for removing inconsistent information have been investigated. Yet all have limited applicability, and, in any case, are not guaranteed to produce consistency. (There is no algorithm for logical falsehood.) Hence, even if measures are taken to dismiss of contradictions when they are found, an underlying paraconsistent logic is desirable if hidden contradictions are not to generate trivial answers to queries.

Belief revision As a part of artificial intelligence research field, belief revision is one of the areas that have been more widely studied. Belief revision consists of rationally revising bodies of belief when in the presence of new evidence. Notoriously, people have inconsistent beliefs while they may even be rational in

doing so. For example, there may be apparently overwhelming evidence for both something and its negation. There may even be cases where it is in principle impossible to eliminate such inconsistency. For example, consider the 'paradox of the preface' as presented by the authors in [PT09]. A rational person, after thorough research, writes a book in which they claim $A_1, \ldots, A_n$. But they are also aware that no book of any complexity contains only truths. So they rationally believe $\neg(A_1 \wedge \ldots \wedge A_n)$ too. Hence, principles of rational belief revision must work on inconsistent sets of beliefs. Standard accounts of belief revision, e.g., that of Gärdenfors et al. [Gär90], all fail to do this, since they are based on classical logic. A more adequate account is one based on a paraconsistent logic.

Nelson's logic N^-

N^- is the weak, paraconsistent version of Nelson's constructive logic with strong negation. Formulas of N^- are built-up in the usual way using the logical connectives for atoms: $\wedge$, $\vee$, $\rightarrow$, $\sim$, standing respectively for conjunction, disjunction, implication and strong negation. The only rule of inference for N^- is modus ponens and the axioms are the axiom schemata of positive logic:

- **P1.** $\alpha \rightarrow (\beta \rightarrow \alpha)$
- **P2.** $(\alpha \wedge \beta) \rightarrow \alpha$
- **P3.** $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$
- **P4.** $(\alpha \wedge \beta) \rightarrow \beta$
- **P5.** $(\alpha \rightarrow \beta) \rightarrow ((\alpha \rightarrow \gamma) \rightarrow (\alpha \rightarrow (\beta \wedge \gamma)))$
- **P6**. $\alpha \rightarrow (\alpha \vee \beta)$
- **P7.** $(\alpha \rightarrow \gamma) \rightarrow ((\beta \rightarrow \gamma) \rightarrow ((\alpha \vee \beta) \rightarrow \gamma))$
- **P8.** $\beta \rightarrow (\alpha \vee \beta)$

plus the following axiom schemata involving strong negation taken from the calculus of Vorob'ev (where $'\alpha \leftrightarrow \beta'$ abbreviates $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$):

- **N1.** $\sim(\alpha \rightarrow \beta) \leftrightarrow \alpha \wedge \sim\beta$ **N2.** $\sim(\alpha \wedge \beta) \leftrightarrow \sim\alpha \vee \sim\beta$
- **N3.** $\sim(\alpha \vee \beta) \leftrightarrow \sim\alpha \wedge \sim\beta$ **N4.** $\sim\sim\alpha \leftrightarrow \alpha$

Paraconsistent answer-set (PAS) [SI95]

The Paraconsistent answer-set semantic (PAS) is a straightforward extension of the answer-set semantic. In fact, we can see PAS as a version of answer-sets where contradictory interpretations are admitted in the evaluation of the semantics. PAS is not defined for every extended logic programs, and cannot be used to determine the information depending on contradictions.

In [SI95] Sakama and Inoue defined paraconsistent answer-sets[5] for extended disjunctive logic programs by permitting contradictory interpretations to be taken into account when determining the answer-sets. In order to achieve this aim, the author presented the notion of satisfaction (denoted by $\vDash$), which is inductively defined as follows:

Definition 1.7 ([SI95](Sakama & Inoue, 1995)) *Let P be an extended disjunctive logic program and I be an extended interpretation. Then*

1. *For an objective literal L*
   - **(a)** *$I \vDash L$ if and only if $L \in I$*
   - **(b)** *$I \vDash not\ L$ if and only if $L \notin I$*
2. *For any disjunction of objective literals $F = L_1 \vee \ldots \vee L_n$, $I \vDash F$ if and only if $I \vDash L_i$ for some $i (1 \leq i \leq n)$.*
3. *For any conjunction of objective literals $G = L_1 \wedge \ldots \wedge L_n$, $I \vDash F$ if and only if $I \vDash L_i$ for every $i (1 \leq i \leq n)$.*
4. *For any rule $r = L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n$, $I \vDash r$ if and only if $I \vDash L_1 \vee \ldots \vee L_l$ or $I \nvDash L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n$.*

We say I is a model of P if and only if I satisfies every rule of P.

An interpretation I settles the set of true objective literals. If $L \in I$ then the objective literal L has the truth value true; otherwise its truth value is false. As expected, if an objective literal L is false, then *not* L is true.

For extended interpretations, the so-called knowledge ordering collapses into the ordinary subset ordering inclusion, i.e. given the extended interpretations I and J, $I \subseteq_k J$ if and only if $I \subseteq J$. We say $I <_k J$ when $I \subseteq_k J$ but $I \neq J$. A model I of P is p-minimal if there is no model J of P such that $J <_k I$.

The paraconsistent stable model semantics of an extended disjunctive logic program is defined as follows.

Definition 1.8 ([SI95](Sakama & Inoue, 1995)) *Let P be an extended disjunctive logic program and I be an extended interpretation. The program division $\frac{P}{I}$ is an extended positive disjunctive logic program such that $L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \in \frac{P}{I}$ if and only if there is a rule $L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge notL_{m+1} \wedge \ldots \wedge notL_n \in P$ such that $\{L_{m+1}, \ldots, L_n\} \cap I = \emptyset$. So I is a paraconsistent answer-set (PAS) of P if I is a p-minimal model of $\frac{P}{I}$.*

After all, the notion of PAS reduces to that of p-minimal models in extended positive disjunctive logic programs, as illustrated in the next example:

Example 1.5 *(Sakama & Inoue, 1995) Let P_1 be the extended logic program $\{a \vee b, \neg a, \neg b, c \leftarrow notd\}$. Then P_1 has two PASs $\{a, \neg a, \neg b, c\}$ and $\{b, \neg a, \neg b, c\}$.*

[5] In their original work [SI95], the authors prefer the term paraconsistent stable models.

A PAS I is contradictory if $\{L, \neg L\} \subseteq I$ for at least one $L \in HB_P$. In the program P_1 of Example 1.5, both PAS are contradictory. Like with answer-sets, it is also possible that a program has no PAS:

Example 1.6 *(Sakama & Inoue, 1995) The extended logic program $P_2 = \{a \leftarrow nota.\ b.\}$ has no PAS.*

Routley Semantics

Routley semantics for N^- The main idea of Routley semantics is that the validity of negation $\sim \alpha$ at a world w is equivalent to the falsity of α not at w as in classical logic but at some adjacent world w^{*1}. To define Routley semantics for N^- we have additionally to divide the set of possible worlds into parts, unstarred and starred worlds.

A Routley model for N^- is a quadruple $M = \langle W \cup W^*, \subseteq, *, V\rangle$ such that: $W \cup W^*$ is a non-empty set (of worlds), $W \cap W^* = \emptyset$, $\subseteq$ is a partial ordering on W, $*$ is a bijection on $W \cup W^*$, $*(W) = W^*$, and V is a valuation function from $Atoms \times W \rightarrow \{0, 1\}$ satisfying the following conditions:

1. $u \subseteq w \Rightarrow w^* \subseteq u^*$,

2. $w = w^{**}$,

3. $V(p, u) = 1$ and $u \subseteq w$ imply $V(p, w) = 1$.

V is extended to a valuation on all formulas via the following conditions:

- $V(\varphi \wedge (\vee)\psi, w) = 1$ if and only if $V(\varphi, w) = 1$ and (or) $V(\psi, w) = 1$
- $V(\sim \varphi, w) = 1$ if and only if $V(\varphi, w^*) = 0$

For implication, one distinguishes between starred and unstarred worlds as follows.

- For $w \in W$, $V(\varphi \rightarrow \psi, w) = 1$ if and only if for every $w' \in W$ such that $w \subseteq w'$, $V(\varphi, w') = 1 \Rightarrow V(\psi, w') = 1$.
- For $w \in W^*$, $V(\varphi \rightarrow \psi, w) = 1$ if and only if $V(\varphi, W^*) = 1 \Rightarrow V(\psi, w) = 1$.

A proposition φ is said to be satisfiable in a Routley model $M = \langle W, \subseteq, *, V\rangle$, if $V(\varphi, v) = 1$, where v is an arbitrarily selected unstarred element of W. A formula is valid if it is true in every interpretation. It is easy to prove by induction that condition 3 above holds for any formula φ, ie $V(\varphi, u) = 1$ plus $u \subseteq w \Rightarrow V(\varphi, w) = 1$.

Defining/characterizing Routley models in terms of program reducts: SE-models, defined by H. Turner, serve as an analogy on how we want to define Routley models without referring to (Kripke-like) entailment. In [OP05], the proof sketch for Proposition 3 gives us a hint at how this can be done.

For any Routley here-and-there model (H,T), by Theorem 1 of [ADP05], (H,T) is a model of a disjunctive program P if and only if for the sets $J = T \bigcup \overline{T^*}$ and $I = H \bigcup \overline{H^*}$, J is a model of P and I is a model of the reduct P^J.

Summary on Routley models In [OP05], the authors have shown how both ordinary and paraconsistent answer-sets can be captured via possible worlds models due to Routley [Rou74]. In the case of PAS, the underlying logic, $N9$, has been identified axiomatically and algebraically, and an important metalogical property - interpolation - has been proved. A consequence of their analysis is that PAS can easily be defined for arbitrary theories. An interesting feature to emerge is that the underlying logic of PAS, although paraconsistent, still extends intuitionistic inference (unlike say well-founded semantics), and is still a conservative extension of the logic of here-and-there.

semi-stable Semantics

Sakama and Inoue have introduced the notion of semi-stable models, which are paraconsistent for incoherent programs, in [SI95]. We will cite this article extensively throughout this thesis and particularly in this introduction to the semi-stable Semantics.

Some important notions must be introduced before we can explain the concept of semi-stable models. First, the definition of the epistemic transformation of non positive programs presented in Definition 3.2 of [SI95]:

Definition 1.9 (Epistemic Transformation) *Let P be an extended disjunctive program. Then its epistemic transformation is defined as the positive extended disjunctive program P^κ obtained from P by replacing each clause (of the following form) in P containing default negation:*

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n (n \neq n)$$

with the following not-free clauses in P:

(1) $\lambda_1 \vee \ldots \vee \lambda_l \vee K_{L_{m+1}} \vee \ldots \vee K_{L_n} \leftarrow L_{l+1} \wedge \ldots \wedge L_m$,

(2) $L_i \leftarrow \lambda_i$ *for* $i = 1, \ldots, l$,

(3) $\leftarrow \lambda_i \wedge L_j$ *for* $i = 1, \ldots, l$ *and* $j = m+1, \ldots, n$,

(4) $\lambda_i \leftarrow L_i \wedge \lambda_k$ *for* $i = 1, \ldots, l$ *and* $k = 1, \ldots, l$.

In particular, each integrity constraint containing default negation is transformed into

$$K_{L_{m+1}} \vee \ldots \vee K_{L_n} \leftarrow L_{l+1} \wedge \ldots \wedge L_m.$$

Note here that each not-free clause in P is included in P^κ as it is.

Let I^κ be an interpretation of P^κ. Then I^κ is called canonical if $KL \in I^\kappa$ implies $L \in I^\kappa$ for any $L \in \mathcal{L}_P$. That is, in a canonical interpretation each believed literal has a justification. Given a set of interpretations $\mathcal{I}_{P^\kappa}$, let

$$obj_c(\mathcal{I}_{P^\kappa}) = \{I^\kappa \cap \mathcal{L}_P \mid I^\kappa \in \mathcal{I}_{P^\kappa} \text{ and } I^\kappa \text{ is canonical } \}.$$

Another important notion is the one of maximally canonical sets:

Let $\mathcal{I}_{P^\kappa}$ be a set of interpretations of a program P^κ obtained by the epistemic transformation of an extended disjunctive program P. Then an interpretation $I^\kappa \in \mathcal{I}_{P^\kappa}$ is said *maximally canonical* if there is no interpretation $J^\kappa \in \mathcal{I}_{P^\kappa}$ such that $\{KL \mid KL \in J^\kappa \text{ and } L \notin J^\kappa\} \subset \{KL \mid KL \in I^\kappa \text{ and } L \notin I^\kappa\}$. That is, a maximally canonical interpretation is an interpretation such that the canonical condition is satisfied as much as possible. In particular, if $\mathcal{I}_{P^\kappa}$ contains an interpretation I^κ which is canonical, it is also maximally canonical. Now, let

$$obj^\kappa_{mc}(\mathcal{I}_{P^\kappa}) = \{I^\kappa \cap \mathcal{L}_{P^\kappa} \mid I^\kappa \in \mathcal{I}_{P^\kappa} \text{ and } I^\kappa \text{ is maximally canonical}\}.$$

Theorem 1.7 ([SI95]) *Let P be an extended disjunctive program. Then, any interpretation included in $SST(P) = obj^\kappa_{mc}(min(\mu(\tau_{P^\kappa} \uparrow \omega)))$ is a model of P.*

This way, intuitively, it is possible to define one of the values in the nine-valued logic above for each literal L in a set $\mathcal{L}$ in terms of L, $\neg L$, K_L and $\neg K_L$. Recall that *not* L corresponds to $\neg K_L$.

Accordingly, if the logic N_9 underlying the Routley models and the logic IX used to capture the semi-stable models are interchangeable, we can use the results applicable to either.

Sakama and Inoue introduced a new fixpoint semantics of positive extended disjunctive programs. In contrast to logic programs containing only definite information, a positive extended disjunctive program has multiple p-minimal models in general. In order to characterize such non-deterministic behavior of disjunctive programs, they first introduced a closure operator which acts over the lattice of sets of Herbrand interpretations $2^{2^{\mathcal{L}^P}}$.

Definition 1.10 ([SI95]) *Let P be a positive extended disjunctive program and I be a set of interpretations. Then, a mapping $\tau_P : 2^{2^{\mathcal{L}^P}} \to 2^{2^{\mathcal{L}^P}}$ is defined as:*

$$\tau_P(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \tau_P(I)$$

where the mapping $T_P : 2^{\mathcal{L}^P} \to 2^{2^{\mathcal{L}^P}}$ is defined as follows:

$$T_P(I) = \begin{cases} \emptyset, \textit{ if } \{L_1, \ldots, L_m\} \subseteq I \textit{ for some ground integrity constraint} \\ \quad \leftarrow L_1 \wedge \ldots \wedge L_m \textit{ from } P; \\ \{J \mid \textit{for each ground clause} \\ \quad C_i : L_1 \vee \ldots \vee L_{l_i} \leftarrow L_{l_i+1} \wedge \ldots \wedge L_{m_i} \\ \quad \textit{from } P \textit{ such that} \\ \quad \{L_{l_i+1}, \ldots, L_{m_i}\} \subseteq I, J = I \cup \bigcup_{C_i} \{L_j\}(1 \leq j \leq l_i)\}, \textit{ otherwise.} \end{cases}$$

Thus, $T_P(I)$ is the set of interpretations J's such that for each clause C_i whose body is satisfied by I, I is expanded into J by adding one disjunct L_j from the heads of every such C_i. In particular, if I does not satisfy an integrity constraint from P, I is removed in $T_P(I)$.

Example 1.8 ([SI95]) *Let P be the program:*

$$\{a \vee b \leftarrow c,\ \ \neg\, d \leftarrow c,\ \ c \leftarrow,\ \ \leftarrow a \wedge b\}.$$

Then,

$$T_P(\{c\}) = \{\{c, \neg d, a\}, \{c, \neg d, b\}\} \text{ and}$$
$$T_P(\{\{c, \neg d, a\}, \{c, \neg d, b\}\}) = \{\{c, \neg d, a\}, \{c, \neg d, b\}, \{c, \neg d, a, b\}\}.$$

Definition 1.11 ([SI95]) *The ordinal powers of τ_P are defined as follows:*

$$\tau_P \uparrow 0 = \{\emptyset\},$$
$$\tau_P \uparrow n + 1 = \tau_P(\tau_P \uparrow n),$$
$$\tau_P \uparrow \omega = \cup_{\alpha<\omega} \cap_{\alpha\leq n<\omega} \tau_P \uparrow n,$$

where n is a successor ordinal and ω is a limit ordinal.

The above definition means that at the limit ordinal ω the closure retains interpretations which are persistent in the preceding iterations. That is, for any interpretation I in $\tau_P \uparrow \omega$, there is an ordinal α smaller than ω such that, for every $n(\alpha \leq n < \omega)$, I is included in $\tau_P \uparrow n$. Such a closure definition is also used in other works for computing stable models of normal logic programs.

We now refer the reader to the properties (stated as lemmas, corollaries and theorems) from this fixpoint operator that are presented in the remaining parts of Section 2.2 of [SI95].

Underlying logic and further definitions: There is an extended disjunctive program which has no p-stable model but still contains useful information. For instance, in Example 1.9, P has no p-stable model but it seems reasonable to conclude the truth of b.

Example 1.9 *The program*

$$P = \{a \leftarrow \mathit{not}\ a.\ \ b \leftarrow\}$$

has no p-stable model.

Roughly speaking, incoherency arises when a literal is implied by its default negation in a program. Since incoherency is viewed as a kind of inconsistency, it is desirable to provide a framework which is paraconsistent for such incoherency. In this section, we introduce the notion of semi-stable models which is paraconsistent for incoherent programs.

To present incoherent facts, we first introduce five extra truth values **bt**, **bf**, **b**$\top$, **tcb**, and **fcb** which respectively denote believed true, believed false, believed contradictory, true with contradictory belief and false with contradictory belief. These values together with the values in IV constitute a lattice of nine-valued logic IX such that $\bot \preceq \mathbf{bx} \preceq \mathbf{x} \preceq \mathbf{xcb} \preceq \top$ and $\mathbf{bx} \preceq \mathbf{b}\top \preceq \mathbf{xcb}$ for $\mathbf{x} \in \{\mathbf{t}, \mathbf{f}\}$.

Let $\mathcal{L}_{P^\kappa} = \mathcal{L}_P \cup \{K_L \mid L \in \mathcal{L}_P\}$ and I^κ be a subset of $\mathcal{L}_P^\kappa$. Then, an interpretation under the logic IX is defined as a function $I^\kappa : \mathcal{L}_{P^\kappa} \rightarrow IX$ such that for each literal

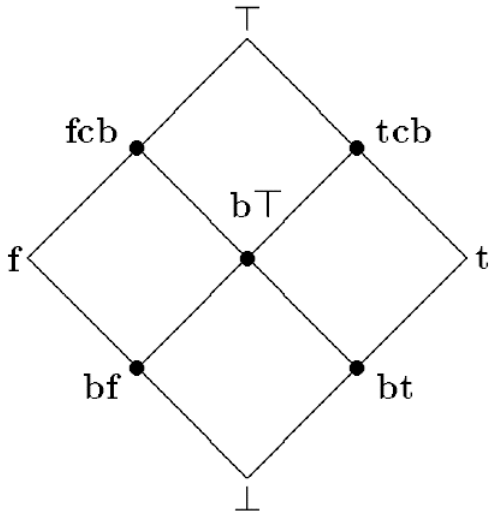

Figure 1.2: Logic IX

$L \in \mathcal{L}_P$,

$$I'^{\kappa}(L) = \text{lub}\{x \mid x = \mathbf{t} \text{ if } L \in I^{\kappa},$$
$$x = \mathbf{f} \text{ if } \neg L \in I^{\kappa},$$
$$x = \mathbf{bt} \text{ if } KL \in I^{\kappa},$$
$$x = \mathbf{bf} \text{ if } K\neg L \in I^{\kappa},$$
$$x = \bot \text{ otherwise } \}$$

Thus, $I^{\kappa}(L) = b\top$ if and only if both $KL \in I^{\kappa}$ and $K\neg L \in I^{\kappa}$; $I^{\kappa}(L) = fcb$ if and only if both $KL \in I^{\kappa}$ and $\neg L \in I^{\kappa}$; $I^{\kappa}(L) = tcb$ if and only if both $K\neg L \in I^{\kappa}$ and $L \in I^{\kappa}$, and so on. Note that $I^{\kappa}(L) = bt$ if and only if $I^{\kappa}(\neg L) = bf$, $I^{\kappa}(L) = b\top$ if and only if $I^{\kappa}(\neg L) = b\top$, and $I^{\kappa}(L) = tcb$ if and only if $I^{\kappa}(\neg L) = fcb$.

The intuitive reading of each newly introduced truth value is that if $I^{\kappa}(L) = bt$, I^{κ} contains a belief KL without its justification L. On the other hand, if $I^{\kappa}(L) = tcb$, I^{κ} contains a fact L with its opposite belief $K\neg L$. Under this logic, satisfaction of literals and default negation is defined in the same way as Section 3 of [SI95], i.e., $I \models L$ if and only if $t \not\models I(L)$; $I \models \neg L$ if and only if $f \not\models I(L)$; $I \models not\ L$ if and only if $I(L) \not\models f$; and $I \models not \neg L$ if and only if $I(L) \not\models t$. Satisfaction of clauses is also defined as before.

According to the above definition, when $I(L) = bt$ or $I(L) = b\top$, it holds that $I \not\models L$ and $I \not\models not\ L$. This means when L is believed true, it is too weak to conclude the truth of L, but enough to reject $not\ L$.[6] Else when $I(L) = tcb$, $I \models L$ while $I \not\models not\ \neg L$. This means when L is true with contradictory belief, I concludes the truth of L but rejects $not\ \neg L$ in the presence of its opposite belief $K\neg L$.

[6]Recall that $not\ L$ corresponds to $\neg KL$.

Next, let I_{P^κ} be a set of interpretations of a program P^κ obtained by the epistemic transformation of an extended disjunctive program P. Then an interpretation $I^\kappa \in I_{P^\kappa}$ is said maximally canonical if there is no interpretation $J^\kappa \in I_{P^\kappa}$ such that $\{KL \mid KL \in J^\kappa \text{ and } L \notin J^\kappa\} \subset \{KL \mid KL \in I^\kappa \text{ and } L \notin I^\kappa\}$. That is, a maximally canonical interpretation is an interpretation such that the canonical condition is satisfied as much as possible. In particular, if I_{P^κ} contains an interpretation I^κ which is canonical, it is also maximally canonical. Now, let

$$obj^\kappa_{mc}(I_{P^\kappa}) = \{I^\kappa \cap \mathcal{L}_{P^\kappa} \mid I^\kappa \in I_{P^\kappa} \text{ and } I^\kappa \text{ is maximally canonical}\}.$$

Next, they introduce their notation for the global minimization operation (*min*) and for the the fixpoint operator τ iterated n times ($\mu(\tau \uparrow n)$).

Theorem 1.10 (Semi-stable model theorem [SI95, Theorem 4.4]) *Let P be an extended disjunctive program. Then, any interpretation included in*

$$SST(P) = obj^\kappa_{mc}(min(\mu(\tau_P^{kappa} \uparrow \omega)))$$

is a model of P.

We call models $SST(P)$ the semi-stable models of P. The notion of semi-stable models can be reduced to the one of p-stable models in coherent programs.

Corollary 1.11 (p-stable model corollary [SI95, Corollary 4.5]) *Let P be a coherent program. Then its semi-stable models coincide with the p-stable models.*

The existence of semi-stable models is guaranteed for any program which has models. Then, Theorem 4.6 in [SI95] states: When a program has a model, it has a semi-stable model. Thus incoherent programs get the meaning by considering semi-stable models. We now refer the reader to the examples in Subsection 2.3 and in particular to Example 1.12 where we present the Barber's paradox and its models according to the semi-stable semantics.

Example 1.12 (The Barber's Paradox) $P = \{shaves(noel, X) \leftarrow not\ shaves(X, X).\ mayor(casanova).\}$ [7]

The models for the grounded program are:

- $(m(c), m(c)s(c, c)s(n, n))$
- $(m(c)s(c, c), m(c)s(c, c)s(n, n))$
- $(m(c)s(n, n), m(c)s(c, c)s(n, n))$
- $(m(c)s(c, c)s(n, n), m(c)s(c, c)s(n, n))$
- $(m(c)s(n, c), m(c)s(n, n)s(n, c))$ $SST\ Model$ $\{m(c), Ks(n, n), s(n, c)\}$
- $(m(c)s(n, n)s(n, c), m(c)s(n, n)s(n, c))$

[7] We will be using predicates s and m for Shave and Mayor, constants n and c for Noel and Casanova.

Note that the above program has neither standard two-valued stable models nor answer-sets.

As such, after performing the epistemic transformation described above to a program with default negation, therefore transforming it in a positive program, we can evaluate it with the usual paraconsistent stable model semantics. In fact the fixed point operator $min(\mu(\tau_{P'^{\kappa}} \uparrow \omega))$, where the minimization is performed in the end of the iterative process, is equivalent to the way the minimality is assured in the usual fixed point operator for stable models presented in Definition 1.5.

This is so because the transformed program is a positive disjunctive program and the authors state that: for positive disjunctive programs, their fixpoint construction characterizes Minker's minimal model semantics. Let MM_P be the set of all minimal models of a positive disjunctive program P. Then, their Theorem 2.9 says:

Theorem 1.13 ([SI95]) *Let P be a positive disjunctive program. Then,*

- $MM_P = min(\mu(\tau_P \uparrow \omega))$.

Because of this and because the minimal models of a positive disjunctive program are also its answer-sets, we can add that $AS'^{\kappa}_P = MM'^{\kappa}_P = min(\mu(\tau'^{\kappa}_P \uparrow \omega))$.

CHAPTER 2

Contributions

2.1 Dealing with default and strong negation

Introduction

In [Pea08], David Pearce draws a very comprehensive introduction to the problem of multiple negations in logic programming which we adapt in this section.

After the first presentation of their stable model semantics, Michael Gelfond and Vladimir Lifschitz contributed with a further paper to the 1990 ICLP conference [GL90]. This paper introduced for the first time the answer-set nomenclature. These authors observed that, besides the typical logic programming negation-as-failure or negation-by-default operators, it would be very useful to have an additional form of negation that could directly correspond to falsity. They named this form of negation 'classical' negation which might have seemed appropriate at first glance but turned out to be slightly misleading after closer examining. Several fundamental properties of classical logic like the excluded middle, contraposition and modus tollens did not hold for this operator. The notion of strong negation was a much closer logical conceptual fit.

Rather than specifying a particular logic containing strong negation as one of its operators, it may, at first, seem odd to speak of an axiomatic system for strong negation. In fact some axiomatic systems, like the Vorob'ev axioms, contain a very interesting property of allowing one to add them to any super-intuitionistic logic, including classical logic, and still obtaining a conservative extension of that logic. Furthermore many key properties of a logic are preserved when passing to its least strong negation extension.

Vorob'ev managed to establish in this way an important property of strong negation. The same property became, many years later, a design feature of the way the second negation operator was introduced into answer-set semantics: the new sort of negation was allowed only to stand directly in front of an atom. This convention was maintained throughout successive extensions of a programs' syntax, from normal pro-

grams, through to disjunctive programs and even the more recent concept of programs with nested expressions. Hence, in each case the only difference between a program rule with or without strong negation was that in the first case the basic elements were, instead of atoms, literals.[1] This convention allowed for a very simple stable model definition extension and an easy proof of some of its important properties.

As soon as someone studying answer-set programming encounters this new second operator, strong negation, then one learns a reduction technique that shows how answer-sets can be characterized in terms of the stable models of a program without strong negation. This technique for reduction consists of introducing a new atom or predicate symbol for each strongly negated literal.

An earlier version of this idea has its roots in Nelson's constructive logic as well. There, one associates with any formula ψ of the original language a strong negation-free formula, say ψ'. If T is any theory in the language of constructive logic, set $T' = \{\psi' : \psi \in T\}$. Let S stand for the set of all formulas of the form $P'(x) \rightarrow \neg P(x)$. Then this early technique establishes that

$$T \vdash_N \psi \text{ if and only if } T' \cup S \vdash_I \psi' \qquad (1)$$

where $\vdash_N$ (resp. $\vdash_I$) stands for inference in Nelson's logic N (respectively intuitionistic predicate logic I). It is easy to see that (1) continues to hold if we replace intuitionistic inference by that of here-and-there logic and replace N by the least strong negation extension of here-and-there. By the method mentioned before, one can quickly derive the main property of Gelfond and Lifschitz's second negation, established in [GL91b] in which the answer-sets of T can be obtained from the answer-sets or stable models of T', by translating the primed atoms back into strongly negated literals.

As a conclusion, a very positive aspect of this technique (and one of great importance to us) is its generality: since it works for arbitrary theories it covers the case of answer-sets for any syntactical class of logic program.

Contribution towards dealing with strong negation

Sakama and Inoue dealt with default negation by means of the epistemic transformation described in 1.9. However, we must still deal with strong negation. As suggested previously, we can do that by transforming negated literals into their primed version:

Definition 2.1 (Prime Transformation) *Let l be a literal, then we define its prime transformation as:*

$$l' = \begin{cases} \text{if } l = q \text{ then } l' = q; \\ \text{if } l = \neg q \text{ then } l' = q'. \end{cases}$$

Now, let L be a set of literals. Then its prime transformation is defined as the set of literals L' obtained from L by applying the prime transformation to every literal in L.

$$L' = \{l' \mid l \in L\}$$

[1] Here a literal should be taken to be an atom or its strong negation.

Let now P be an extended disjunctive program. Then its prime transformation is defined as the disjunctive program P' obtained from P by replacing every literal by its prime transformation in each clause contained in P.

This transformation will allow us to compute models containing pairs of the same atom in their unprimed L and primed L' (instead of $\neg L$) forms. Those models would otherwise be inconsistent. This notion has a direct translation into Routley's starred worlds being that the H and T (H^* and T^*) correspond to, respectively, known and believed literals. In this case we define a seven-valued interpretation for this logic as follows:

Definition 2.2 (Interpretation I'^{κ}) *An interpretation I'^{κ} is defined as a function $I'^{\kappa} : \mathcal{L}_{P'^{\kappa}} \rightarrow VII$ such that for each literal $L \in \mathcal{L}_P$,*

$$\begin{aligned} I'^{\kappa}(L) = \text{lub}\{x \mid\ & x = \mathbf{t} \textit{ if } L \in I'^{\kappa}, \\ & x = \mathbf{f} \textit{ if } L' \in I'^{\kappa}, \\ & x = \mathbf{bt} \textit{ if } KL \in I'^{\kappa} \textit{ and } L \notin I'^{\kappa}, \\ & x = \mathbf{bf} \textit{ if } KL' \in I'^{\kappa} \textit{ and } L' \notin I'^{\kappa}, \\ & x = \mathbf{b}\top \textit{ if both } KL \in I'^{\kappa} \textit{ and } K\neg L \in I'^{\kappa}, \\ & x = \top \textit{ if both } L \textit{ and } L' \in I'^{\kappa}, \\ & x = \bot \textit{ otherwise.} \} \end{aligned}$$

Notice that, in the original definition of semi-stable models, Sakama and Inoue introduced the logic IX that we refer in Figure 1.2 as the underlying logic of semi-stable models. In our approach, since we eliminate strong negation with the prime transformation, we can simplify the underlying logic without loss of generality with this logic VII.

2.2 Model Transformations / Embeddings

In this section, we provide several embeddings as well as some results that will be necessary for the next section. From these definitions and lemmas, we highlight Proposition 2.3 where we establish an equivalence between semi-stable models and maximally canonical answer-sets and Lemma 2.5 where a here-and-there and Routley models correspondence is presented.

semi-stable model semantics equivalences

Definition 2.3 (Constructing an HT Model from I'^{κ}) *Let P'^{κ} be a program obtained from P after preforming both the prime and the epistemic transformations. To an interpretation I'^{κ} of P'^{κ} we define its equivalent here-and-there model $HT_{I'^{\kappa}}(P'^{\kappa})$ as a pair (H, T) such that:*

- $H = \{L \mid I'^{\kappa}(L) = \top \vee I'^{\kappa}(L) = \boldsymbol{t} \vee I'^{\kappa}(L) = \boldsymbol{f}\}$
- $T = \{L \mid I'^{\kappa}(L) \neq \bot\}$

The fixpoint semantics that we will be using to calculate both semi-stable models and answer-sets reuses the definition presented in [SI95] and characterizes operational aspects of logic programs and is also implemented using bottom-up model generation techniques as presented in [IKH92].

Lemma 2.1 (Fixpoint equivalence) *Let P^κ be a program obtained after applying the epistemic transformation K in Definition 1.9 to a program P and P'^κ be its prime transformation according to Definition 2.1. Then, the results of the fixpoint calculations are semantically equivalent $\mu(\tau_{P^\kappa} \uparrow w)' = \mu(\tau_{P'^\kappa} \uparrow w)$.*

Proof *Intuitively, $S \in \mu(\tau_{P^\kappa} \uparrow w)$ if and only if $S' \in \mu(\tau_{P'^\kappa} \uparrow w)$. We can show this by induction:*

Basis: *For $S_0 \in \mu(\tau_{P^\kappa} \uparrow 0)$, $S_0 = \{\emptyset\}$ and also $S'_0 = \{\emptyset\}$*

Hypothesis: *Assume that if $S_n \in \mu(\tau_{P^\kappa} \uparrow n)$, then $S'_n \in \mu(\tau_{P'^\kappa} \uparrow n)$.*

Using the hypothesis,

$$\begin{array}{l} S_{n+1} \in \mu(\tau_{P^\kappa} \uparrow n) \cup \tau_{P^\kappa_{n+1}} \text{ and} \\ S'_{n+1} \in \mu(\tau_{P'^\kappa} \uparrow n) \cup \tau_{P'^\kappa_{n+1}} \text{ are equivalent iff} \\ \tau'_{P^\kappa_{n+1}} = \tau_{P'^\kappa_{n+1}}. \end{array}$$

For any $S_n \in \mu(\tau_{P^\kappa} \uparrow n)$, considering S_{n+1} such that S_{n+1} is a minimal set of head atoms h from P^κ such that for each $r \in P^\kappa$ of the form $a_1 \wedge \ldots \wedge a_n \leftarrow b_1, \ldots, b_e, not\ b_{e+1}, \ldots, not\ n_m$ where:

$$\begin{array}{l} b_1, \ldots, b_e \in S_n \text{ and} \\ b_{e+1}, \ldots, b_m \notin S_n \end{array}$$

there is a $h \in S_{n+1}$.

Then $r' \in P'^\kappa$ and by hypothesis:

$$\begin{array}{l} b'_1, \ldots, b'_e \in S'_n \text{ and} \\ b_{e+1}, \ldots, b_m \notin S'_n \end{array}$$

Therefore, $\exists h'$ such that $h' \in S'_{n+1}$. S_{n+1} is a minimal set of atoms h in the head of the rule r such that $Body(r) \vDash S_n$ and S'_{n+1} is a minimal set of atoms h' in the head of the rule r' such that $Body(r') \vDash S'_n$

So, if l is a literal derived by such rule r, then l' is an atom derived by rule r':

$$(\forall \neg L \in \tau_{P^\kappa_{n+1}} \rightarrow L' \in \tau_{P^{\kappa'}_{n+1}}) \wedge (\forall L \in \tau_{P^\kappa_{n+1}} \rightarrow L \in \tau_{P^{\kappa'}_{n+1}}).$$

□

Lemma 2.2 ($AS(P'^\kappa) = min(\mu(\tau_{P'^\kappa} \uparrow w))$) *Let P'^κ be a program obtained after transforming program P as in Definitions 1.9 and 2.1. Then, the minimal sets of literals calculated with the fixpoint operator $\mu(\tau_{P'^\kappa} \uparrow w)$ will be the same as its answer-sets. More formally: $AS(P'^\kappa) = min(\mu(\tau_{P'^\kappa} \uparrow w))$.*

Proof *Let's start by noticing that given an arbitrary extended disjunctive program P, its primed epistemic transformation P'^{κ} will, by definition, be positive - without default negation - and strong negation free.*

Furthermore, definition 2.2 in [SI95] deals with positive extended disjunctive programs where the literals can be either strongly negated or not. So, in this case, strong negation is dealt with syntactically without any explicit semantics which is provided a posteriori *in Definition 2.2. Now, we have $AS(P'^{\kappa}) = min(\mu(\tau_{P'^{\kappa}} \uparrow w))$ if and only if:*

(1) *There is a model $S \in AS(P'^{\kappa})$ implies that there is a model $S \in min(\mu(\tau_{P'^{\kappa}} \uparrow w))$*

(2) *There is a model $S \in min(\mu(\tau_{P'^{\kappa}} \uparrow w))$ implies that there is a model $S \in AS(P'^{\kappa})$*

(1) *Let's start by assuming a set of literals I such that $AS(P'^{\kappa}) = I$. Since after the epistemic transformation is performed we obtain a positive and strong negation free disjunctive program such that I is minimal among the models of the reduct P'^{κ^I}, then the minimal set of interpretations obtained with the operator $\mu(\tau_{P'^{\kappa}} \uparrow w)$ will also contain I. This is the case because the usual fixpoint semantics used to characterize answer-sets that we pointed in Definition 1.5 ([LRS97]) turns out to be equivalent to the one presented in 1.10 ([SI95]) after selecting the interpretations that are minimal*[2].

So, (1) holds.

(2) *Let's now assume an interpretation $I^{\kappa} = min(\mu(\tau_{P^{\kappa}} \uparrow w))$.*[3] *The prime transformation we introduced is merely a syntactical transformation so each maximally canonical interpretation I'^{κ} included in $min(\mu(\tau_{P'^{\kappa}} \uparrow w))$ is also a model of P.*

To prove this, for each transformed clauses (1) and (2), $\{L_{l+1}, \ldots, L_m\} \subseteq I'^{\kappa}$ implies either $L^i \in I'^{\kappa}(1 \leq i \leq l)$ or $KL_j \in I'^{\kappa}(m+1 \leq j \leq n)$. In case of $L_i \in I'^{\kappa}$, I'^{κ} satisfies the corresponding clause (2) in P. In case of $KL_j \in I'^{\kappa}$, when $L_j \in I'^{\kappa}$, I'^{κ} satisfies the clause (2) in P. So, I'^{κ} satisfies the clause (2) in P. Else when $L_j \notin I^{\kappa'}$,

(i) *if $L'_j \notin I'^{\kappa}$, the truth value of L_j is **bt**, then $I'^{\kappa} \not\models$ not L_j.*

(ii) *Else if $\neg L_j \in I'^{\kappa}$, the truth value of L_j becomes **fcb**, then $I'^{\kappa} \not\models$ not L_j.*

Therefore, I'^{κ} satisfies each clause in P. As follows, $I'^{\kappa} \cap \mathcal{L}_{P'^{\kappa}}$, which is obtained from I'^{κ} by removing every λ_i, is also a model of P.

Let's now assume that I'^{κ} is not a minimal model of P'^{κ}. Then, there will be a subset I^*, such that $I^* \subset I'^{\kappa}$, that is a model of P'^{κ}. But, because of the minimality condition used to calculate interpretation I'^{κ}, this is not possible

[2] Notice that both operators are non deterministic in a sense that they result in multiple sets of atoms. The one for semi-stable models is reported in [SI95] as being equivalent to the one used to calculate answer-sets if the program in not disjunctive. As for the disjunctive case, the selection of the atoms to include in each model is virtually the same except for the minimality condition that is enforced step-by-step in [LRS97] and not in [SI95]. So, the fixpoint operator presented for semi-stable models potentially produces multiple sets of atoms from which some are not minimal. Selecting the minimal one leaves us with the same sets as with the answer-sets fixpoint operator.

[3] The proof of Theorem 4.4 in [SI95] is very similar to what we need.

and as such I'^{κ} is a minimal model of P'^{κ}. This is the case because the T_P operator in Definition 1.10 does not encode a minimization in each step, this minimization is performed after the fixpoint is reached.

□

Proposition 2.3 (Semi-stable models as maximally canonical answer-sets) *Let P'^{κ} be a program obtained after applying the epistemic and the prime transformations to a program P as in Definitions 1.9 and 2.1. Then, its answer-sets will coincide with the semi-stable models of the same program $P^{\kappa'}$ after selecting the maximally canonical interpretations obj^{κ}_{mc}. That is, $obj^{\kappa'}_{mc}(AS(P^{\kappa'})) = SST(P)$.*

Proposition 2.3 follows trivially from Lemma 2.2 and from the modified definition of SST models as $SST_{P'^{\kappa}} = obj^{\kappa'}_{mc}(min(\mu(\tau_{P^{\kappa'}} \uparrow \omega)))$.

Here-and-there and Routley models embeddings

Corollary 2.4 *For any strong negation free program P, the starred worlds in its Routley models will be empty and so the validity of every $\neg$ free formulae will be defined via unstarred worlds only. As such, for any strong negation free program P we have that for a here-and-there model (H, T) we will have a Routley model $(H, \emptyset, T, \emptyset)$.*

We must still introduce some definitions, that will allow us to make some crucial correspondences, before arguing the equivalence between HT-models of a primed program P' and the Routley models of the original program P.

Definition 2.4 ($R^P(H, T, P')$ Transformation) *Let P' be a primed disjunctive logic program and (H, T) a here-and-there interpretation of P'. We can construct a correspondent Routley interpretation of P as follows:*

$$R^P(H, T, P') = \langle \{h \mid h \in H \wedge h \in \mathcal{L}_P\}, \{t \mid t \in T \wedge t \in \mathcal{L}_P\}, \{h \mid h' \in H \wedge h' \in \mathcal{L}_{P'}\}, \{t \mid t' \in T \wedge t' \in \mathcal{L}_{P'}\}\rangle.$$

Definition 2.5 ($HT^{P'}(R, P)$ Transformation) *Let P be a disjunctive logic program and (Rh, Rt, Rh^*, Rt^*) be a Routley interpretation of P. Now let P' be the program obtained from P by applying the transformation in Definition 2.1. We can construct a here-and-there interpretation of P' as follows:*

$$HT^{P'}(R, P) = \langle\ Rh \cup Rt^{*\prime},\ Rt \cup Rh^{*\prime}\ \rangle.$$

The next lemma follows from the previous definitions and the one of prime transformation:

Lemma 2.5 (Here-and-there and Routley models correspondence) *Let P' be the program obtained from P by applying the Prime transformation in Definition 2.1. Then,*

(a) *If R is a Routley model of P there will be a here-and-there pair $HT^{P'}(R, P)$ that is a model of P'.*

(b) *If (H, T) is a here-and-there model of P', then $R^P(H, T, P')$ is a Routley model of P.*

Proof *The Routley models for any program P and the here-and-there models for its prime transformation P' will be equivalent in a sense that the truth value for any formula, modulo the prime transformation, will be the same.*

The proof for this lemma is in fact very simple since this equivalence is based on a syntactical transformation of the model's representation. As such, the way the validity of formulae is checked is accordingly equivalent. The starred and unstarred worlds in the Routley model are merged, 'priming' all the literals in the starred worlds as presented in Definition 2.5.

(a) *In order to prove this direction of the lemma, assume $R = (J, K, J^*, K^*)$ is a Routley model of a program P. Towards a contradiction, assume that $HT^{P'}(R, P) = (H, T)$ (where $H = J \cup J^{*\prime}$ and $T = K \cup K^{*\prime}$) is not an HT-model. Then:*

1. *$T \not\models P'$ implies that there is a rule $r' \in P'$ such that $T \models Body_P(r') \wedge T \not\models Head_P(r')$. Now, consider the unprimed version of the same rule $r \in P$. Then, by construction $K \cup K^{*\prime} \models Body_P(r) \wedge K \cup K^{*\prime} \not\models Head_P(r)$ which is contradictory to the assumption that R is a model of P.*
2. *$\not\models P'^T$ implies that there is a rule $r' \in P'^T$ in the reduct of P according to T, such that $H \models Body_P(r') \wedge H \not\models Head_P(r')$.*
   This implies that given the unprimed version of the same rule $r \in P^K$, $J \cup J^{*\prime} \models Body(r) \wedge J \cup J^{*\prime} \models Body(r) \wedge J \cup J^{*\prime} \not\models Head(r)$ which contradicts the assumption.

(b) *As for the second direction, assume (H, T) is an HT-model of P', then $T \models P' \wedge H \models P'^T$. Now let a Routley model R such that $R^P(H, T, P') = (J, K, J^*, K^*)$ constructed from (H, T) according to Definition 2.4, such that $K \cup K^* \models P \wedge J \cup J^* \models P^{K \cup K^*}$.*

1. *Towards a contradiction, let's assume $K \cup K^* \not\models P$ then there is an $r \in P$ s.t. $K \cup K^* \models Body_P(r) \wedge K \cup K^* \not\models Head_P(r)$. Now let's take an r' as in Definition 2.1 such that: $r' \in P'$ and $T \models Body_P(r')$ but $H \not\models Head_P(r')$.*
   This contradicts the assumption that (H,T) models P'.
2. *Towards a contradiction, let's assume $J \not\models P^K$. Let $r \in P^K$ be a set of literals such that $J \models Body_P(r) \wedge J \not\models Head_P(r)$. Now consider an r' such that $r' \in P'$, $H \models Body_{P'}(r')$ and $H \not\models Head_{P'}(r')$.*
   This contradicts the assumption because this way $H \not\models P'^T$.

□

By using Lemma 2.5, we get HT-models with both primed and unprimed literals. The validity of any formulae can now be defined in the usual way after matching the unprimed and the primed literals. We present in Example 2.6 a simple illustration of this operation:

Example 2.6 $R(\{a\}, \{b, c\}, \{a\}, \{b\}) = HT(\{a, b'\}, \{a, b', c'\})$

Where, as expected, in both cases $a = \mathbf{t}$, $b = \mathbf{f}$, $c = \mathbf{bf}$.

Let's now introduce a relaxed notion of here-and-there interpretations that we will call semi-stable here-and-there interpretations (*SHT*):[4]

Definition 2.6 (Semi-stable here-and-there interpretations and models (*SHT*)) *We call any pair (H, T) of interpretations over $\mathcal{L}_{P'}$ an semi-stable here-and-there interpretation (SHT-interpretation).*

We then define SHT-models as SHT-interpretations such that:

(a) $H \models P'^T$ *and*

(b) *if $H \cap Body^-(r) \neq \emptyset$ and $H \models Body^+(r)$, then $T \cap Body^-(r) \neq \emptyset$, for any rule $r \in P'$.*

Hence, $SHT(P)$ is the set of semi-stable here-and-there models of P.

With these semantic structures, we will be able characterize SST-models exactly (i.e., there will be a one-to-one correspondence between *SHT* and SST models). Let's start by defining some syntactical transformations between such models.

Definition 2.7 (Transformation Y^κ) *Given an SHT-interpretation (H, T) we define a corresponding interpretation $Y^K(H, T)$ over the language $\mathcal{L}'^\kappa$ as*

$$Y^K(H, T) = H \cup \{KL \mid L \in T\}$$

and a corresponding interpretation $Y^\kappa(H, T)$ over $\mathcal{L}'^\kappa$ (i.e., the "epistemic language" including lambdas) as

$$Y^\kappa(H, T) = Y^K(H, T) \cup \{\lambda_{r,i} \mid Body^-(r) \neq \emptyset,\ L_i \in H,\ H \models Body^+(r),\ (H \cup T) \cap Body^-(r) = \emptyset\}$$

We will use simply the implicit notations Y^K and Y^κ if both sets H and T from (H, T) are understood.

Intuitively, the set of believed literals introduced in the previous definition will contain every literal that belongs to the set T but not to H. It will also contain the minimum sets of literals such that if there is a rule with default negated literals in its body, at least one of them will be included in previously mentioned set.

The complete Y^κ set will be formed by the set of believed literals together with all literals in H and the lambda literals $\lambda_{r,i}$ such that L_i is the i-th literal in the head of rule r, L belongs to H and T models the negative part of the rule's body which, in turn, cannot be empty.

Definition 2.8 (Transformation $SHT^Y(Y)$) *Given an interpretation Y^K over $\mathcal{L}'^\kappa$, respectively an interpretation Y^κ over $\mathcal{L}'^\kappa$, a corresponding SHT-interpretation is obtained by*

[4] It is not required for T to be a model of P'

$$H = \{L \in \mathcal{L}_{P'} \mid L \in Y^K\}$$

and

$$T = \{L \in \mathcal{L}_{P'} \mid KL \in Y^K\},$$

where Y^K is the restriction to $\mathcal{L}^K$ in case of a given interpretation over $\mathcal{L}'^\kappa$ (symbolically, $Y^K = Y^\kappa_{\mathcal{L}^K}$).

Lemma 2.7 ($Y^\kappa \models P'^\kappa \Leftrightarrow (H,T) \in SHT(P)$) *If (H,T) is an SHT-model of P, then Y^κ is a model of P'^κ.*

Conversely, if Y^κ is a model of P'^κ, then (H,T) is an SHT-model of P.

Proof ($\Rightarrow$) *If (H,T) is an SHT-model of P, then Y^κ is a model of P'^κ.*

Towards contradiction assume the contrary. For a rule r:

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n (m \neq n)$$

Case (0): $Body^-(r) = \emptyset$

Some rule r that in the original program P had no default negation (as such was not transformed by the epistemic transformation) is not satisfied. If this is the case then there is a literal $L \in Head(r)$ s.t. $L \notin Y^\kappa$ which implies that $L \notin H$ by Definition 2.8. Now, r has no default negation so P is the same as its reduct P^T. Hence, $H \not\models P^T$ and (H,T) is not an SHT-model of P which contradicts the assumption.

Case (1): $Y^\kappa \not\models$ *(1)* $\lambda_1 \vee \ldots \vee \lambda_l \vee K_{L_{m+1}} \vee \ldots \vee K_{L_n} \leftarrow L_{l+1} \wedge \ldots \wedge L_m$ [5]

Then, the following holds:

- *$Y^\kappa \models Body^+(r)$. Hence, $H \models Body^+(r)$ because $Body^+(r)$ contains plain literals which are in Y^K and as such in $H = \{L \mid L \in Y^K\}$.*
  And,
- *$Y^\kappa \not\models Head_{\mathcal{L}^\kappa}(r)$. Hence, $T \cap Body^-(r) = \emptyset$ because the head of the transformed rule $Head_{\mathcal{L}^\kappa}(r)$ contains believed literals KL such that negated literals not L were in $Body^-(r)$ and because by construction $T = \{L \mid KL \in Y^K\}$, $T \cap Body^-(r) = \emptyset$.*

We conclude that r^T in $P'^{T'}$, and since $H \models Body^+(r)$ and (H,T) is an SHT-model, it follows that $H \models Head(r)$.

Now consider two cases. First assume that $(H \cup T) \cap Body^-(r) \neq \emptyset$. Then, $H \cap Body^-(r) \neq \emptyset$ follows, since we already know $T \cap Body^-(r) = \emptyset$. However, since (H,T) is an SHT-model, this implies $T \cap Body^-(r) \neq \emptyset$ which is a contradiction to condition b) of the SHT-models definition. Therefore, $(H \cup T) \cap Body^-(r) = \emptyset$ has to hold.

[5] The notation (1) - (4) refers to the respective types of rules generated by the epistemic transformation in Definition 1.9.

Consequently $\lambda_{r,i} \in Y^\kappa$ for some i because by definition Y^κ contains $\{\lambda_{r,i}\}$ such that $Body^-(r)$ is not empty, $L_i \in H$, $H \models Body^+(r)$ and $(H \cup T) \cap Body^-(r) = \emptyset$ conditions which all hold.

But then, Y^κ models the head of (1) which is a contradiction to our assumption that $Y^\kappa \not\models$ (1) $\lambda_1 \vee \ldots \vee \lambda_l \vee K_{L_{m+1}} \vee \ldots \vee K_{L_n} \leftarrow L_{l+1} \wedge \ldots \wedge L_m$.

Case (2): *$Y^\kappa \not\models$ (2) $L_i \leftarrow \lambda_i$ for $i = 1, \ldots, l$*

This implies that $Y^\kappa \models Body^+(r)$ and $Y^\kappa \not\models Head_{\mathcal{L}^\kappa}(r)$. By definition, the set $Body^+(r)$ is formed by one literal $\lambda_{r,i}$. As such, $\lambda_{r,i} \in Y^\kappa$ which by construction implies that $L_i \in H$ which makes L_i be in Y^κ. The set $Head_{\mathcal{L}^\kappa}(r)$ is formed by a single literal L_i so $Y^\kappa \models Head_{\mathcal{L}^\kappa}(r)$ which contradicts our assumption.

Case (3): *$Y^\kappa \not\models$ (3) $\leftarrow \lambda_i \wedge L_j$ for $i = 1, \ldots, l$ and $j = m+1, \ldots, n$*

Then, $\lambda_{r,i}$ and L_j in Y^κ for some i and j. The former implies $(H \cup T) \cap Body^-(r) = \emptyset$ because, by Definition 2.7, $Y^\kappa = Y^K \cup \{\lambda_{r,i} \mid Body^-(r) \neq \emptyset, L_i \in H, H \models Body^+(r), (H \cup T) \cap Body^-(r) = \emptyset\}$ and thus if we have $\lambda_{r,i}$ then the conditions including $(H \cup T) \cap Body^-(r) = \emptyset$ must be fulfilled.

The latter implies $H \cap Body^-(r) \neq \emptyset$, and thus $(H \cup T) \cap Body^-(r) \neq \emptyset$. We thus have a contradiction.

Case (4): *$Y^\kappa \not\models$ (4) $\lambda_i \leftarrow L_i \wedge \lambda_k$ for $i = 1, \ldots, l$ and $k = 1, \ldots, l$*

Then, $\lambda_{r,k}$ and L_i in Y^κ and $\lambda_{r,i} \notin Y^\kappa$ for some i and j.

Since $\lambda_{r,k} \in Y^\kappa$, $(H \cup T) \cap Body^-(r) = \emptyset$, $Body^-(r) \neq \emptyset$, and $H \models Body^+(r)$ hold by construction. Since $L_i \in Y^\kappa$, it also is the case that $L_i \in H$. Hence, again by construction, $\lambda_{r,i} \in Y^\kappa$ has to hold which once again is a contradiction.

($\Leftarrow$) *If Y^κ is a model of P'^κ, then (H, T) is an SHT-model.*

Towards contradiction assume the contrary.

Case (a): *$H \not\models {P'}^{T'}$. Then, there is some r in ${P'}^{T'}$, such that $H \not\models r^T$. If $Body^-(r) = \emptyset$, then, since r is not transformed by the epistemic transformation, $H \not\models r$ implies $Y^\kappa \not\models r$ which is a contradiction.*

So let $Body^-(r) \neq \emptyset$. Then, $T \cap Body^-(r) = \emptyset$ (since if some literal in $Body^-(r)$ would also be in T, the reduct r^T would be void), $H \models Body^+(r)$ and $H \not\models Head(r)$. It follows that $Y^\kappa \models Body^+(r)$.

Moreover, $Y^\kappa \not\models Head_{\mathcal{L}^\kappa}(r)$. We have this because the transformed rule of type (1) will contain in its head the believed form of the literals that were in the $Body^-(r)$. Now, because $T \cap Body^-(r) = \emptyset$ and remembering the construction in Definition 2.7: $Y^K = H \cup \{KL \mid L \in T\}$ and Y^κ contains Y^K (plus some lambda literals), we have that Y^κ will not contain the believed literals it should in order to model the $Head_{\mathcal{L}^\kappa}(r)$.

In addition, $L_i \notin H$ for all i since $H \not\models Head(r)$. Consequently, $\lambda_{r,i} \notin Y^\kappa$ for all i. Thus, Y^κ does not model the head of a rule of type (1) which is a contradiction.

Case (b): *We have that:*

- $H \models Body^+(r)$ *because by construction* Y^κ *contains all literals in* H*, which implies that* $Y^\kappa \models Body^+(r)$*.*
- $H \cap Body^-(r) \neq \emptyset$*, implies that* $L_i \in Y^\kappa$ *for some i. This is so because having all conditions satisfied in* $\{\lambda_{r,i} \mid Body^-(r) \neq \emptyset,\ L_i \in H,\ H \models Body^+(r),\ (H \cup T) \cap Body^-(r) = \emptyset\}$ *(according to Definition 2.7 of* Y^κ*), this* λ_j *will be in* Y^κ*. Now, because of our assumption that this set* Y^κ *models* P'^κ L_j *has to be also* Y^κ *in order to satisfy transformed rules of type (2) which enforce that* $L_i \leftarrow \lambda_i$ *for* $i = 1, \ldots, l$*.*
  And,
- $T \cap Body^-(r) = \emptyset$ *for some* r *in* P' *implies that* $Y^\kappa \not\models Head_{\mathcal{L}^\kappa}(r)$ *because, for rules of type (1), each literal in* $Body^-(r)$ *will appear as a believed literal KL in the* $Head_{\mathcal{L}^\kappa}(r)$*. This is so because ,since these KL literals appear (by Definition 2.7 in* Y^κ *because the latter contains* $\{KL \mid L \in T\}$*, having that L is not in* $Body^-(r)$ *means that KL is not in* Y^κ*.*

Since $H \cap Body^-(r) \neq \emptyset$ *implies* $(H \cup T) \cap Body^-(r) \neq \emptyset$*, we conclude that* $\lambda_{r,i} \notin Y^\kappa$ *for all i. Thus,* Y^κ *does not model the head of (1). Note that* $H \cap Body^-(r) \neq \emptyset$ *implies* $H \cap Body^-(r) \neq \emptyset$*, i.e., (1) is in* P'^κ*. This presents a contradiction to our assumption.*

This proves both directions of Lemma 2.7, i.e., that if (H, T) *is an SHT-model of* P*, then* Y^κ *is a model of* P'^κ *and vice-versa.* □

We will now establish a relation between answer-sets Y^κ of P'^κ and its corresponding *SHT*-interpretation (H, T) of P.

Lemma 2.8 *If* Y^κ *is an answer-set of* P'^κ*, then the corresponding SHT-interpretation* (H, T) *is an SHT-model of P that satisfies condition (i).*

(i) $\nexists H' \subset H$ *such that* $(H', T) \in SHT(P)$

Proof *If* Y^κ *is an answer-set of* P'^κ*, then the corresponding SHT-interpretation* (H, T) *is an SHT-model that satisfies (i). Furthermore, if* Y^κ *is an answer-set of* P'^κ *then it is a model hence, by Lemma 2.7,* (H, T) *is an SHT-model.*

Let now $\bar{Y}^\kappa$ *be the interpretation over* $\mathcal{L}^\kappa$ *obtained from* (H, T) *by the construction given by the construction given.*

$\bar{Y}^\kappa = Y^\kappa$**:** *We first show that* $\bar{Y}^\kappa = Y^\kappa$*. Obviously,* $\bar{Y}^\kappa$ *and* Y^κ *coincide on* $\mathcal{L}^K$ *because the construction in Definition 2.7:* $Y^K = H \cup \{KL \mid L \in T\}$ *and* Y^κ *contains* Y^K *(plus some lambda literals not in* $\mathcal{L}^K$*).*

Suppose they differ, and first assume $\bar{Y}^\kappa \subset Y^\kappa$*. Since* $\bar{Y}^\kappa \models P'^\kappa$ *by Lemma 2.7, this contradicts the assumption that* Y^κ *is an answer-set of* P'^κ*. Hence, assume* $\bar{Y}^\kappa \not\subseteq Y^\kappa$ *such that both* $\bar{Y}^\kappa \not\subset Y^\kappa$ *and* $\bar{Y}^\kappa \neq Y^\kappa$*. Then, there exists some* $\lambda_{r,i} \in \bar{Y}^\kappa$*, such that* $\lambda_{r,i} \notin Y^\kappa$*. This implies, by the construction given, that* $Body^-(r) \neq \emptyset$*,* L_i *is in* H*,* $H \models Body^+(r)$*, and* $(H \cup T) \cap Body^-(r) = \emptyset$*.*

This further implies that $L_i \in \bar{Y}^\kappa$, and thus $L_i \in Y^\kappa$, as well as $\bar{Y}^\kappa \models Body^+(r)$ and hence $Y^\kappa \models Body^+(r)$. $(H \cup T) \cap Body^-(r) = \emptyset$ implies $T \cap Body^-(r) = \emptyset$, and hence $\bar{Y}^\kappa \not\models Head(r)|_{\mathcal{L}_K}$. Consequently, also $Y^\kappa \not\models Head(r)|_{\mathcal{L}_K}$. Note that since $Body^-(r) \neq \emptyset$, there are negated literals in the body of the rule r as so it is transformed by the epistemic transformation.

Now:

- *First, assume that there is some k such that $\lambda_{r,k} \in Y^\kappa$. Then, $\mathcal{L}_i$ and $\lambda_{r,k}$ in Y^κ and $\lambda_{r,i} \notin Y^\kappa$ implies that Y^κ does not model (4), a contradiction.*
- *Therefore, there is no k such that $\lambda_{r,k} \in Y^\kappa$. But then Y^κ does not model the head of (1), again a contradiction.*

We thus conclude that neither $\bar{Y}^\kappa \subset Y^\kappa$ nor $\bar{Y}^\kappa \not\subseteq Y^\kappa$ can hold. This proves $\bar{Y}^\kappa = Y^\kappa$.

Contradiction: *So for the remainder of the proof, we identify Y^κ with the interpretation over $\mathcal{L}^\kappa$ obtained from (H, T). Towards a contradiction with the claim of Lemma 2.8, assume that (H, T) does not satisfy (i), i.e., there exists (H', T) such that $H' \subset H$ and (H', T) is an SHT-model:*

By Lemma 2.7, we know that then Y'^κ is a model of P'^κ. We show that $\{\lambda_{r,i} \mid \lambda_{r,i} \in Y'^\kappa\} \subseteq \{\lambda_{r,i} \mid \lambda_{r,i} \in Y^\kappa\}$. Assume that this is not the case. Then, there exists some $\lambda_{r,i}$ in Y'^κ, such that $\lambda_{r,i} \notin Y^\kappa$. Then, by construction $Body^-(r) \neq \emptyset$, L_i is in H', $H' \models Body^+(r)$, and $(H' \cup T) \cap Body^-(r) = \emptyset$. It follows that $L_i \in Y'^\kappa$, and thus $L_i \in Y^\kappa$, as well as $Y'^\kappa \models Body^+(r)$, $H \models Body^+(r)$, and hence $Y^\kappa \models Body^+(r)$.

Now consider two cases for $(H \cup T) \cap Body^-(r)$.

- *First assume that this set is empty. Then, by construction $\lambda_{r,i} \in Y'^\kappa$, a contradiction to our assumption that this is not the case. Hence $(H \cup T) \cap Body^-(r)$ is nonempty.*
- *Then, since we know that $(H' \cup T) \cap Body^-(r) = \emptyset$ which implies trivially that $T \cap Body^-(r) = \emptyset$, $H \cap Body^-(r) \neq \emptyset$. Together with $H \models Body^+(r)$ and the fact that $T \cap Body^-(r) = \emptyset$, we get a contradiction with the fact that (H, T) is an SHT-model according to condition (b) on the definition of SHT-models.*

This proves $\{\lambda_{r,i} \mid \lambda_{r,i} \in Y'^\kappa\} \subseteq \{\lambda_{r,i} \mid \lambda_{r,i} \in Y^\kappa\}$. As a consequence, $Y'^\kappa \subset Y^\kappa$ which contradicts our assumption that Y^κ is an answer-set of P'^κ.

This proves Lemma 2.8, i.e., that (H, T) is an SHT-model that satisfies (i). □

Lemma 2.9 *If (H, T) is an SHT-model that satisfies (i), then there exists an answer-set Y'^κ of P'^κ, such that the corresponding SHT-interpretation is of the form (H, T') such that $T' \subseteq T$, and (H, T') is an SHT-model that satisfies (i).*

(i) *$\nexists H' \subset H$ such that $(H', T) \in SHT(P)$*

Proof *Let (H,T) be an SHT-model satisfying (i), and let Y^κ be the interpretation over $\mathcal{L}^\kappa$ obtained from (H,T) as given by construction in Definition 2.7. If Y^κ is an answer-set of P'^κ, then we are done, since its corresponding SHT-interpretation is (H,T), $T \subseteq T$, and (H,T) satisfies (i).*

So, assume that Y^κ is not an answer-set of P'^κ. Since by Lemma 2.7 Y^κ it is a model, we conclude that there is an answer-set Y'^κ of P'^κ, such that $Y'^\kappa \subset Y^\kappa$ (Recall that, because of the prime transformation in Definition 2.1, P'^κ is default negation free). Let (H',T') be the SHT-interpretation obtained from Y'^κ. By Lemma 2.7, we know that (H',T') is an SHT-model of P'. Note that $Y'^\kappa \subset Y^\kappa$ implies $T' \subseteq T$, by construction.

Now, towards a contradiction assume that $H' \subset H$. We show that (H',T) is an SHT-model of P', as well.

Case (a) *$H' \not\models {P'}^T$ implies $H' \not\models {P'}^{T'}$. Now, if some rule in ${P'}^T$ is not satisfied by H' it means that its body is satisfied but not its head. As we know by construction of the reduct three things can happen to a rule that is not satisfied by H' in ${P'}^T$:*

- *The rule was not changed because it contained no default negation in its body. In this case $H' \not\models Head(r^{T'})$ trivially implies $H' \not\models Head(r^T)$ because $Head(r) = Head(r^T) = Head(r^{T'})$.*
- *The original rule contained default negation but $T \cap Body^-(r) = \emptyset$. Then, the rule's body is deleted from its reduct r^T. In this case (because $T' \subseteq T$) $T' \cap Body^-(r) = \emptyset$ and as such the rule's body is also deleted from the reduct $r^{T'}$. This way, because the transformed rule that is not satisfied is the same in both reducts, $H' \not\models {P'}^T$ implies $H' \not\models {P'}^{T'}$.*
- *The original rule contained default negation and $T \cap Body^-(r) \neq \emptyset$. Then, this rule was deleted from the program's reduct P^T in the first place and so it was trivially satisfied by H' and is not to be considered here.*

This presents a contradiction to (H',T') being an SHT-model of P'. Hence, $H' \models {P'}^{T'}$.

Case (b) *Assume that there is some r in P' such that the following hold: $H' \models Body^+(r)$, $H' \cap Body^-(r) \neq \emptyset$, and $T \cap Body^-(r) = \emptyset$. Then, $H \models Body^+(r)$ and $H \cap Body^-(r) \neq \emptyset$, follow. Together with $T \cap Body^-(r) = \emptyset$, this is in contradiction with our assumption that (H,T) is an SHT-model of P' according to condition (b) in the SHT-models definition. Therefore, $H' \models Body^+(r)$ and $H' \cap Body^-(r) \neq \emptyset$ implies $T \cap Body^-(r) \neq \emptyset$, for all r in P'.*

This proves that (H',T) is an SHT-model of P'. However, this contradicts our assumption that (H,T) satisfies (i). We thus conclude that $H' = H$.

Carrying trough with our indirect argument, assume that (H,T') (which we have seen that equals (H',T')) does not satisfy (i), i.e., there exists an SHT-model (H'',T') such that $H'' \subset H$. We show that then (H'',T) is an SHT-model of P', as well. The argument is the same as above replacing H' with H''. Again, this contradicts our assumption that (H,T) satisfies (i).

Therefore, we conclude that (H,T') satisfies (i), which proves Lemma 2.9. □

2.3 Characterization of semi-stable models

A one to one characterization of semi-stable models as semi-stable here-and-there models where to one SST model corresponds an *SHT* model is easy to obtain, but the opposite is not quite as easy.

Characterization of semi-stable models as HT or Routley models

Proposition 2.10 *If (H, T) is an SHT-model that satisfies (i), (ii) and (iii), then the corresponding interpretation Y^K over $\mathcal{L}^K$ is an SST-model. If Y^K is an SST-model then the corresponding SHT-interpretation (H, T) is an SHT-model that satisfies (i), (ii), and (iii).*

(i) *There is no (H', T) such that $H' \subset H$ and (H', T) is an SHT-model of P'.*

(ii) *$T \backslash H$ is subset-minimal among the semi-stable HT-interpretations (SHT) satisfying (i), i.e.,*

- *$\nexists (H', T') \in SHT(P')$, s.t. (H', T') satisfies (i) and $T' \backslash H' \subset T \backslash H$.*

(iii) *There is no (H, T') such that $T' \subset T$ and (H, T') is an SHT-model of P'.*

Proof *Throughout this proof we will extensively use Definition 2.8 (as well as its inverse Definition 2.7) and Lemma 2.7, but we will omit these references when their usage is clear.*

($\Rightarrow$) In order to prove the if direction, let's assume (H, T) is an SHT model of P' such that (H, T) satisfies conditions (i), (ii) and (iii). Therefore, we claim that the set of literals Y^K is an SST model.

Towards a contradiction, assume that Y^K is not a semi-stable model. Then, by Proposition 2.3:

$$Y^\kappa \notin obj'^\kappa_{mc}(AS(P'^\kappa)) \ [6]$$

In order for this to happen, either that:

Case 1: *$Y^\kappa \notin AS(P'^\kappa)$ which implies either that:*

(a) *According to the characterization of answer-sets as Equilibrium Models, the total model $(Y^\kappa, Y^\kappa) \notin EQ(P'^\kappa)$ and consequently (Y^κ, Y^κ) is not an SHT-model of P'^κ which implies that $Y^\kappa \not\models P'^{\kappa^Y}$.*

In fact program P'^κ and its reduct P'^{κ^Y} are equivalent because P'^κ is positive so $Y^\kappa \not\models P'^\kappa$. Hence, according to Lemma 2.7, the set $(H, T) = HT^{Y^\kappa}(Y^\kappa)$ will not be an SHT-model of P' which contradicts the assumption.

Or:

[6] Y^κ is the set constructed from the *SHT* model, containing Y^κ plus the lambdas, according to definition 2.7.

(b) *There will be a model $Y^{\kappa*}$ such that $Y^{\kappa*} \subset Y^{\kappa}$ and the pair $SHT^{Y^{\kappa}}(Y^{\kappa*}) = (H', T')$ is an SHTmodel of P'.*

Now, according to the construction presented in Definition 2.8 we have that $H' = \{L \mid L \in Y^{K^*}\}$ and $T' = \{L \mid KL \in Y^{K^*}\}$. We now have to distinguish between several possibilities for the construction of $Y^{\kappa*}$:

1. *if $Y^{K^*} \subset Y\kappa$ because one or more lambda literals λ_L were removed from Y^{κ}, in which case $(H', T') = (H, T)$. By construction, though, a set Y^{κ} constructed from (H, T) will have one correspondent set of lambdas. Deterministically always the same. As such, $Y^{\kappa*}$ cannot be a subset of $Y\kappa$ but instead has to be equal. So, this case where lambdas are removed from the set can be discarded.*
2. *if $Y^{\kappa*} \subset Y^{\kappa}$ because one or more literals L were removed from Y^{κ} then $H' \subset H$ because $H = \{L \mid L \in Y^{K}\}$ which is contradictory to the assumption that (H, T) satisfies (i). Or:*
3. *if $Y^{\kappa*} \subset Y^{\kappa}$ because one or more K-believed literals KL were removed from Y^{κ}, then $T' \subset T$ because $T = \{L \mid KL \in Y^{K}\}$ which implies that $T'\backslash H' \subset T\backslash H$ which is a contradiction to the assumption that (H, T) satisfies (ii). Or:*
4. *if $Y^{\kappa*} \subset Y^{\kappa}$ because one or more literals L and K-believed literals KL were removed from Y^{κ} then three things might have occurred[7]:*
   - *more believed literals KL were removed from Y^{κ}, making it so that $T'\backslash H' \subset T\backslash H$ which is a contradiction to the assumption that (H, T) satisfies (ii). Or:*
   - *more literals L were removed from Y^{κ}, in which case $H' \subset H$ because $H = \{L|L \in Y\}$ which is contradictory to the assumption that (H, T) satisfies (i). Or:*
   - *the same set of believed and normal literals were removed from Y^{κ} (the same set of literals must have been removed from H and from T). In this case conditions (i) and (iii) are violated.*

Or:

Case 2: *Y^K is not maximally canonical according to obj'^{κ}_{mc}. Then, there is a model $Y^{K^*} \subset Y$ that can be transformed into a semi-stable here-and-there model $(H', T') = SHT^{Y}(Y^{K^*})$.*

Having that the set Y^K is not maximally canonical (remember that an interpretation $I^K \in \mathcal{I}_{P'^{\kappa}}$ is said maximally canonical *if there is no interpretation $J^K \in \mathcal{I}_{P'^{\kappa}}$ such that $\{KL \mid KL \in J^K$ and $L \notin J^K\} \subset \{KL \mid KL \in I^K$ and $L \notin I^K\}$.) implies that a given (H', T') will be such that $T'\backslash H' \subset T\backslash H$ which is a contradiction to the assumption that (H, T) satisfies (ii).*

Hence, the if direction of Proposition 2.10 holds.

[7]Note that because we assumed that (H', T') is an *SHT*interpretation of P', $H' \subseteq T'$ and so it cannot contain literals in the H part without being also in the T part.

(⇐) As for the only-if direction, given an SST-model Y^K, let Y^K be the corresponding AS of P'^κ and let (H,T) be its corresponding SHT-interpretation. We need to show that (H,T) is an SHT-model that satisfies (i), (ii), and (iii). We first prove that (H,T) is an SHT-model such that:

(H,T) satisfies (i) and (ii): *According to Proposition 2.3, $Y^K \in AS(P'^\kappa)$ and it is also an Equilibrium model $Y^K \in EQ(P'^\kappa)$. Because of this $(Y^K, Y^K) \in SHT(P'^\kappa)$.*

Towards a contradiction, assume that (H,T) does not satisfy (i) or (ii). Then:

Case (i): *Assume (i) is not satisfied, then there is an $H' \subset H$ such that $(H',T) \in SHT(P')$. Let Y^{K^*} be the set constructed from (H',T). We can show that $Y^{K^*} \subset Y^K$ because it is constructed from the SHT-model as follows: $Y^K = Y^K \cup \{\lambda_{r,i} \mid Body^-(r) \neq \emptyset, L_i \in H, H \models Body^+(r), (H \cup T) \cap Body^-(r) = \emptyset\}$ where $Y^K = H \cup \{KL \mid L \in T\}$.*

As such, if the here *part of the semi-stable here-and-there model (H',T) is a subset of the one in (H,T), we trivially have that $Y^{K^*} \subset Y^K$.*

In line with the assumption that $(H',T) \in SHT(P')$, we have that (Y^{K^*}, Y^K) is an SHT-model.

Having $Y^{K^*} \subset Y^K$ and (Y^{K^*}, Y^K) is an SHT-model implies evidently that because there is a smaller SHT-model, (Y^K, Y^K) is not an equilibrium model hence it is not stable. Having that (Y^K, Y^K) is not an equilibrium model of P'^κ further implies that $Y^K \notin AS(P'^\kappa)$ which contradicts the assumption that it is a semi-stable model.

Case (ii): *Assume (ii) is not satisfied. Then, there is an (H',T') such that $T' \backslash H' \subset T \backslash H$ i.e., a set $Y^{K^*} = Y^K$ constructed from (H,T) according to Definition 2.7 will not be maximally canonical and as such $Y^{K^*} \notin obj^{\prime\kappa}_{mc}(AS(P'^\kappa))$. Because $Y^{K^*} = Y^K$ (both are constructed in the same way, from the same SHTmodel), this is contradictory to the assumption that $Y^K \in SST_{P'^\kappa}$.*

Next, let $\bar{Y}^\kappa$ be the interpretation over $\mathcal{L}^\kappa$ obtained from (H,T) by the construction given. We prove that $\bar{Y}^\kappa = Y^\kappa$.[8]

$\bar{Y}^\kappa = Y^\kappa$: *We first show that $\bar{Y}^\kappa = Y^\kappa$. Obviously, $\bar{Y}^\kappa$ and Y^κ coincide on $\mathcal{L}^K$ because the construction in Definition 2.7: $Y^K = H \cup \{KL \mid L \in T\}$ and Y^κ contains Y^K (plus some lambda literals not in $\mathcal{L}^K$).*

Suppose they differ, and first assume $\bar{Y}^\kappa \subset Y^\kappa$. Since $\bar{Y}^\kappa \models P'^\kappa$ by Lemma 2.7, this contradicts the assumption that Y^κ is an answer-set of P'^κ. Hence, assume $\bar{Y}^\kappa \not\subseteq Y^\kappa$ such that both $\bar{Y}^\kappa \not\subset Y^\kappa$ and $\bar{Y}^\kappa \neq Y^\kappa$. Then, there exists some $\lambda_{r,i} \in \bar{Y}^\kappa$, such that $\lambda_{r,i} \notin Y^\kappa$. This implies, by the construction given, that $Body^-(r) \neq \emptyset$, L_i is in H, $H \models Body^+(r)$, and $(H \cup T) \cap Body^-(r) = \emptyset$.

This further implies that $L_i \in \bar{Y}^\kappa$, and thus $L_i \in Y^\kappa$, as well as $\bar{Y}^\kappa \models Body^+(r)$ and hence $Y^\kappa \models Body^+(r)$. $(H \cup T) \cap Body^-(r) = \emptyset$ implies $T \cap Body^-(r) = \emptyset$, and hence $\bar{Y}^\kappa \not\models Head(r)|_{\mathcal{L}_K}$. Consequently, also $Y^\kappa \not\models Head(r)|_{\mathcal{L}_K}$. Note that

[8] The proof is as in the proof of Lemma 2.8. We recall this part here anyway to make the reading easier.

since $Body^-(r) \neq \emptyset$, there are negated literals in the body of the rule r as so it is transformed by the epistemic transformation.

Now:

- *First, assume that there is some k such that $\lambda_{r,k} \in Y^\kappa$. Then, $\mathcal{L}_i$ and $\lambda_{r,k}$ in Y^κ and $\lambda_{r,i} \notin Y^\kappa$ implies that Y^κ does not model (4), a contradiction.*
- *Therefore, there is no k such that $\lambda_{r,k} \in Y^\kappa$. But then Y^κ does not model the head of (1), again a contradiction.*

We thus conclude that neither $\bar{Y}^\kappa \subset Y^\kappa$ nor $\bar{Y}^\kappa \not\subseteq Y^\kappa$ can hold. This proves $\bar{Y}^\kappa = Y^\kappa$.

(H,T) satisfies (iii): *Towards a contradiction assume that (H, T) does not satisfy (iii). Then, there exists an SHT-model (H, T') of P', such that $T' \subset T$.*

We first show that (H, T') satisfies (i): Otherwise there exists an SHT-model (H', T') of P', such that $H' \subset H$. But then, also (H', T) is an SHT-model of P'. To see the latter, observe that $H' \models P'^{T'}$ implies $H' \models P'^{T}$ because for every r^T which is not void in P'^T (i.e., which is not deleted because T contained no literal in the rule's negative body), it holds that $r^{T'} = r^T$ (i.e., it is equal and hence also not deleted).

So, whenever $H' \cap Body^-(r) \neq \emptyset$ and $H' \models Body^+(r)$ implies $T' \cap Body^-(r) \neq \emptyset$, then (since $T' \subset T$) also $T \cap Body^-(r) \neq \emptyset$ follows. This proves that if (H, T') does not satisfy (i), then (H, T) does not satisfy (i), and since we have established above that (H, T) satisfies (i), so does (H, T').

From this, we can conclude that $T' \setminus H = T \setminus H$ (Otherwise (H, T) does not satisfy (ii), which we have established above) i.e., $H \cup T' = H \cup T$. It follows that Y'^κ, as constructed from (H, T') according to Definition 2.7, coincides with Y^κ on the lambdas, i.e., $\{\lambda_{r,i} \mid \lambda_{r,i} \in Y'^\kappa\} = \{\lambda_{r,i} \mid \lambda_{r,i} \in Y^\kappa\}$. This is so because (recall the construction) $Y^\kappa = Y^K \cup \{\lambda_{r,i} \mid Body^-(r) \neq \emptyset, L_i \in H, H \models Body^+(r), (H \cup T) \cap Body^-(r) = \emptyset\}$, $Y'^\kappa = Y'^K \cup \{\lambda_{r,i} \mid Body^-(r) \neq \emptyset, L_i \in H, H \models Body^+(r), (H \cup T') \cap Body^-(r) = \emptyset\}$ and since H is the same in both models and $H \cup T' = H \cup T$, all conditions for the inclusion if a given lambda in the sets Y^κ and Y'^κ are equivalent.

Consequently, Y'^κ is a proper subset of Y^κ (and a model of P'^κ), a contradiction to our assumption that Y^κ is an AS of P'^κ. Therefore, (H, T) satisfies(iii) as well. Hence, the only-if direction also holds. This ends our proof of Proposition 2.10. □

As a remark also serving as a conclusion to this section, with this characterization we can easily see that because of the global condition (ii) in Proposition 2.10, the *unfounded* sets will be as small as possible i.e., the knowledge that is preserved will be as big as possible. This is a very nice property of semi-stable models that the original definition is not able to make completely clear.

Examples

Using the prototypical implementation described in Section 2.4, we obtained both the here-and-there and the semi-stable models of some programs. From these models and after we were confident about the basic properties described earlier, we conjecturized

the characterization presented in Proposition 2.10. Some examples were also explored in order to verify the equivalence of Stable and semi-stable semantics when no inconsistency or incoherence is present and also some basic examples that hopefully allow getting a better intuition of the semi-stable semantics behavior.

Notice that Routley models are quadruples (H, H^*, T, T^*) of interpretations (subsets) over some language L, such that $H \subseteq T$ and $T^* \subseteq H^*$ and here we present the equivalent here-and-there interpretations whenever it is more convenient for the reader.[9]

Example 2.11 *Let $P = \{a \leftarrow not\ \neg a\}$ over the language $L = \{a\}$.*

The program is not inconsistent nor incoherent and has an answer-set a. This corresponds to its semi-stable model.

- $(a, \varnothing, a, \varnothing)$ *SST model* $\{Ka, a\}$.
- $(a, a, a, \varnothing)$
- $(\varnothing, a, \varnothing, \varnothing)$
- $(\varnothing, a, a, \varnothing)$
- (a, a, a, a)
- $(\varnothing, a, \varnothing, a)$
- $(\varnothing, a, a, a)$

For strong-negation free programs, we can omit the starred parts of the Routley models. Hence, we can work solely with semi-stable here-and-there models.

Example 2.12 $P = \{e \leftarrow not\ f. \quad f \leftarrow not\ e.\}$

In this example, the program has answer-sets and they coincide (by definition) with the semi-stable models.

- (f, f) *SST model* $\{Kf, f\}$
- (e, e) *SST model* $\{Ke, e\}$
- (e, ef)
- (f, ef)
- $(\varnothing, ef)$
- (ef, ef)

Example 2.13 $P = \{a \leftarrow not\ a.\}$ *(For checking inconsistency)*

- $(\varnothing, a)$ *SST model* $\{Ka\}$

[9] To ease notation, we write instead of $\{a, b, c\}$ just 'abc', etc.

- (a, a)

In this incoherent program P it is known that interpretations 'oscillate' between $\emptyset$ and $\{a\}$ under the stable class semantics [BS92]. Accordingly, it is interesting to observe that the truth value $I(a) = bt$ in its semi-stable model correspondingly lies between $\bot$ and t.

In Example 2.14, a *Depth 3 loop through negation*, we get a clear idea of how the characterization of semi-stable models presented in Proposition 2.10 works. We get several here-and-there interpretations (and their corresponding Routley versions) from which only the ones where $\nexists H' \subset H$ such that $(H', T) \in HT(P)$ and the ones where $[T \backslash H]$ is globally minimal are semi-stable models.

Example 2.14 *[Depth 3 loop through negation.]*
$P = \{a \leftarrow not\ b. \quad b \leftarrow not\ c. \quad c \leftarrow not\ a.\}$
We have the following semi-stable here-and-there interpretations.

- $(\emptyset, abc)$
- (a, abc)
- (b, abc)
- (c, abc)
- (ab, abc)
- (ac, abc)
- (bc, abc)
- (abc, abc)
- (b, ab) $SST\ Model$ $\{Ka, Kb, b\}$
- (ab, ab)
- (a, ac) $SST\ Model$ $\{Ka, a, Kc\}$
- (ac, ac)
- (c, bc) $SST\ Model$ $\{Kb, Kc, c\}$
- (bc, bc)

Example 2.15 $P = \{a \leftarrow b. \quad b \leftarrow a. \quad \leftarrow a, b.\}$
There are no SST nor any SHT models.

Let's end this section with an example that clearly shows how traditional here-and-there models would be too strong to characterize semi-stable models and instead we need semi-stable here-and-there models:

Example 2.16 $P = \{b \leftarrow a. \quad a \leftarrow not\ a.\}$

We have the following semi-stable here-and-there interpretations.

- $(\emptyset, a)$
- $(\emptyset, ab)$
- (b, ab)
- (ab, ab)

Selected semi-stable here-and-there Models:

- $(\emptyset, a)$

semi-stable models:

- $\{Ka\}$

2.4 Implementation

Two prototypes were developed, one for each of the characterizations presented.

Prototype for SST characterization as maximally canonical answer-sets. For this implementation, we have used the dlv system as the answer-set solver in order to obtain the answer-sets of a transformed program.

In Figure 2.1 we present the static dependencies between the modules of which our system is composed.

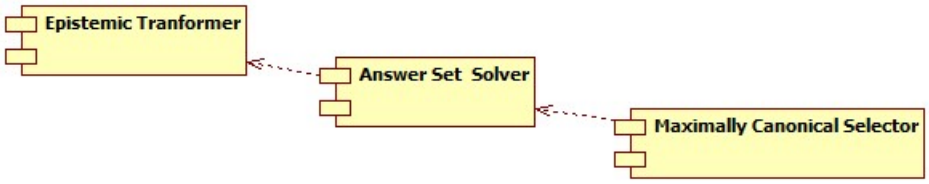

Figure 2.1: Implementation Diagram

The process of calculating the semi-stable models for a given program consists, in this case, of three steps:

1. We start by producing the epistemic and prime transformation P'^{κ} for a given program P. The resulting program is already in the language $\mathcal{L}_{P'^{\kappa}}$.
2. After this transformation, we evaluate P'^{κ} with dlv and get its answer-sets.
3. Afterwards, we just need to select the models that are maximally canonical as defined before in Section 1.8.

Example 2.17 *Reconsider the loop-through-negation in Example 2.14:*

$$P = \{a \leftarrow not\ b. \quad b \leftarrow not\ c. \quad c \leftarrow not\ a.\}$$

This program is translated into:

$$\begin{array}{l} P'^{\kappa} = \{\lambda_a \vee Kb. \quad a \leftarrow \lambda_a. \quad \leftarrow \lambda_a, b. \\ \lambda_b \vee Kc. \quad b \leftarrow \lambda_b. \quad \leftarrow \lambda_b, c. \\ \lambda_c \vee Ka. \quad c \leftarrow \lambda_c. \quad \leftarrow \lambda_c, a.\} \end{array}$$

For this program, the answer-sets are:

- $\{Ka, Kb, b, \lambda_b\}$
- $\{Kb, Kc, c, \lambda_c\}$
- $\{Kc, Ka, a, \lambda_a\}$
- $\{Ka, Kb, Kc\}$

We can easily see that $\{Ka, Kb, Kc\}$ is not maximally canonical, hence cannot correspond to a semi-stable model.

After removing the λ atoms we get, as expected, three semi-stable models:

- $\{Ka, Kb, b\}$
- $\{Kb, Kc, c\}$
- $\{Kc, Ka, a\}$

Prototype for SST characterization in the logic of here-and-there. We developed a prototypical implementation that allowed us to obtain both the semi-stable here-and-there interpretations (as well as the corresponding Routley interpretations) and the semi-stable models for disjunctive logic programs. We depict the components that form its implementation in Figure 2.2.

For obtaining quick results we started by reusing a module that calculates here-and-there models[10] but afterwards we developed an HT-Model generator in order to have the whole prototype integrated in one piece of software.

The following Figure 2.3 depicts the sequence of actions that are performed in order to calculate the semi-stables models of a program according to this characterization.

1. We start the computation of a program's semi-stable models by generating all possible combinations of literals in the original language of the program we have in hand ($\mathcal{L}_P$)).

[10] Our thanks to Stefan Woltran woltran@dbai.tuwien.ac.at for making his implementation available for us as well as for his kind help in using it.

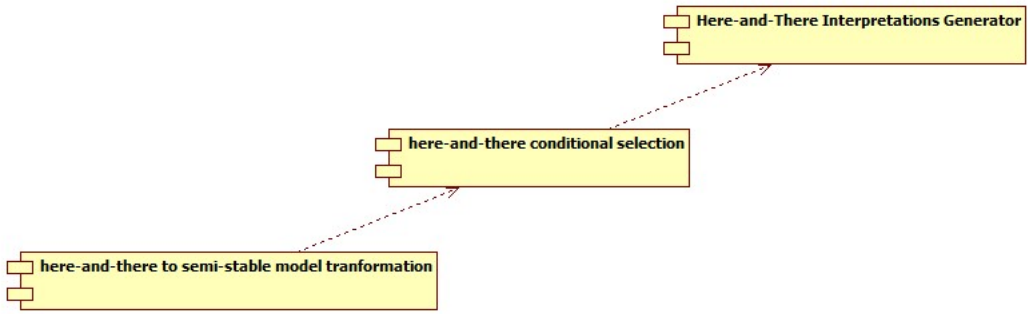

Figure 2.2: Implementation Diagram

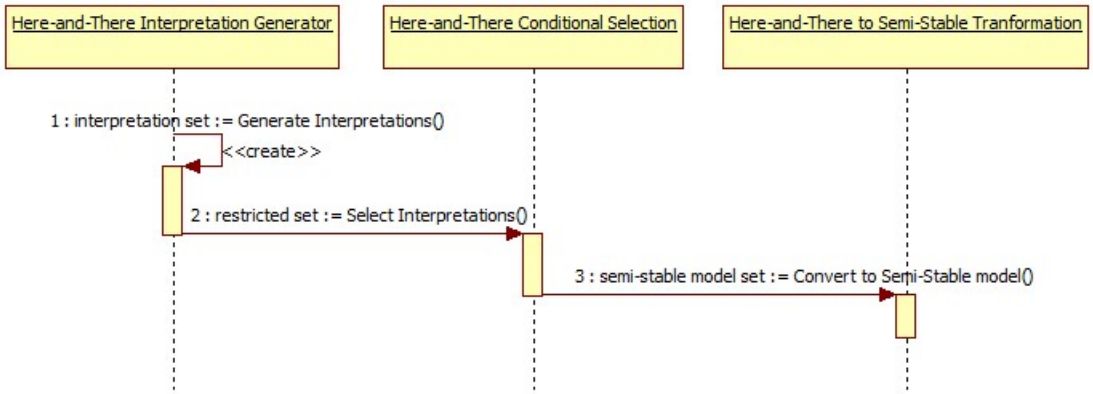

Figure 2.3: Sequence Diagram

2. Then, the candidate semi-stable here-and-there interpretations will be the ones where $H \subseteq T$ and $H \models P^T$. In order to do this we need to calculate the reduct P^T of P for each set of literals T and check if the model's corresponding H part satisfies this reduct.
3. Then, having finally a set of candidate *SHT* interpretations (over the language $\mathcal{L}_P$) for a program P, we only need to apply the conditions in Proposition 2.10 thus selecting the here-and-there interpretations that are able to be converted into semi-stable models.
4. Now, each of the models in the resulting set of *SHT*models are transformable into semi-stable models (over the language $\mathcal{L}_{P^\kappa}$) by performing the transformation in Definition 2.7.

We now present an example of a step-by-step computation of the semi-stable models of a program with our implementation.

Example 2.18 *The loop through negation program presented before:*

$$P = \{a \leftarrow not\ b. \quad b \leftarrow not\ c. \quad c \leftarrow not\ a.\}$$

Has the following semi-stable here-and-there models, obtained from a set of all possible interpretations where $H \subseteq T$ and $T \models P \wedge H \models P^T$:

- $(\{\}, \{abc\})$
- $(\{a\}, \{ac\})$
- $(\{a\}, \{abc\})$
- $(\{b\}, \{ab\})$
- $(\{b\}, \{abc\})$
- $(\{c\}, \{bc\})$ $(\{c\}, \{abc\})$
- $(\{ab\}, \{abc\})$
- $(\{ac\}, \{abc\})$
- $(\{bc\}, \{abc\})$
- $(\{ab\}, \{ab\})$
- $(\{ac\}, \{ac\})$
- $(\{bc\}, \{bc\})$
- $(\{abc\}, \{abc\})$

After selecting the SHTModels according to the two conditions in our characterization, we get:

- $(\{a\}, \{ac\})$
- $(\{b\}, \{ab\})$
- $(\{c\}, \{bc\})$

Let's remember that in accordance to Definition 2.7 a semi-stable interpretation can be constructed from a semi-stable here-and-there interpretation in the following way:[11]

$$Y^{\kappa} = Y^{K} \cup \{\lambda_{r,i} \mid Body^{-}(r) \neq \emptyset,\ L_i \in H,\ H \models Body^{+}(r),\\ (H \cup T) \cap Body^{-}(r) = \emptyset\} \text{ where } Y^{K} = H \cup \{KL \mid L \in T\}$$

and that after removing the λ atoms, the corresponding semi-stable models for the selected here-and-there interpretations will respectively be:

- $\{Ka, Kc, a\}$
- $\{Ka, Kb, b\}$
- $\{Kb, Kc, c\}$

Note that, our characterization of semi-stable models is computationally very straightforward and of easy implementation. We chose Java for its portability and because a prototype could be produced with reasonable effort.

In Figure 2.3 we depict the three operations necessary to calculate the semi-stable models of a program P. The first is formally defined by definition of semi-stable here-and-there models. The second is defined in this document in Proposition 2.10 and the third in Definition 2.7.

[11] Once again, note that in the epistemic transformation, the lambdas are *uniquely associated with each ground instance of a clause* (cf. [SI95] p.12). This means, we can have different lambdas for the same literal if it appears in different rule heads. Hence the additional subscript r for the lambdas above.

CHAPTER 3

Discussion

3.1 Related work

In their trail blazing paper [SI95], Inoue and Sakama presented declarative semantics for extended disjunctive programs. They have introduced the paraconsistent minimal and stable model semantics for extended disjunctive programs based on lattice-structured multi-valued logics. The paraconsistent semantics were characterized by a new fixpoint semantics of extended disjunctive programs. They have also discussed applications of the paraconsistent semantics for reasoning with inconsistency. Furthermore, they argue that the paraconsistent minimal/stable model semantics are natural extensions of the usual minimal/stable model semantics for disjunctive programs, and compared with Gelfond and Lifschitz's answer-set semantics, the proposed semantics do not trivialize a program in the presence of inconsistent information. The paraconsistent semantics presented in that paper generalizes previous studies of paraconsistent logic programming and provides a uniform framework of logic programming possibly containing inconsistent information, disjunctive information, integrity constraints, and both explicit and default negation in a program.

To overcome the problem of non-existing models for some extended logic programs, Sakama and Inoue defined the semi-stable model semantics, which suffers from problems in the treatment of undefined literals manifested in the need of having different languages for the program itself and for it's models.

In [ADP05] Alcantara, Damasio and Pereira defined a fully declarative approach for paraconsistent answer-sets. They achieved this by resorting to a framebased semantics while claiming this was the first time a complete declarative characterization was presented for paraconsistent answer-sets. No syntactic transformation was used in their approach. In fact, paraconsistent answer-sets are obtained simply by minimizing models satisfying some conditions. They have also shown how to embed answer-sets and stable models via frames. They introduced point sets as $P = \langle Q, \sqsubseteq \rangle$, with Q a set

and $\sqsubseteq$ a partial ordering on Q. Propositions on P are upwards closed subsets of Q. Frames are point sets together with accessibility relations. According to [ADP05], an HT^2 frame is defined as follows: The underlying point set $P = \langle Q, \sqsubseteq \rangle$ is such that Q has four elements that [OP05] denote by h, h^*, t, t^*, where $h \sqsubseteq t$ and $t^* \sqsubseteq h^*$. Secondly, three accessibility relations are defined on P that we will denote here by R, $R_{\sim}$, $R_{\neg}$. R is a ternary relation and $R_{\sim}$ and $R_{\neg}$ are binary relations determined by:

$$
\begin{array}{l}
R(h,h,h), R(h,h,t), R(h,t,t), R(t,t,t), R(t,h,t), \\
R(t^*,t,t^*), R(t^*,t,h^*), R(t^*,h,t^*), R(t^*,h,h^*), R(h^*,h,h^*). \\
R_{\sim}(h,h^*), R_{\sim}(t,t^*), R_{\sim}(h^*,h), R_{\sim}(t^*,t), R_{\sim}(h,t^*), R_{\sim}(t^*,h). \\
R_{\neg}(h,h), R_{\neg}(t,t), R_{\neg}(h,t), R_{\neg}(t,h), R_{\neg}(h^*,h), R_{\neg}(t^*,h), R_{\neg}(t^*,t).
\end{array}
$$

An HT^2 model M is formed from an HT^2 frame by assigning atoms to the four points in accordance with the interpretation that propositions form upwards closed sets. In other words if we denote the set of atoms true in w by W, then we have $w \subseteq w'$ implies $W \subseteq W'$. The assignment of atoms to points is extended to all propositions via standard conditions for conjunction and disjunction and the following clauses:

$(M, w) \vDash \varphi \rightarrow \psi$ if and only if $\forall w', w''$ s.t. $R(w, w', w''), (M, w') \vDash \varphi \Rightarrow (M, w'') \vDash \psi$.

$(M, w) \vDash \neg\varphi$ if and only if $\forall w'$ s.t. $R_{\neg}(w, w')$, $(M, w') \nvDash \varphi$.

$(M, w) \vDash\sim \varphi$ if and only if $\forall w'$ s.t. $R_{\sim}(w, w')$, $(M, w') \nvDash \varphi$.

Odintsov and Pearce in [OP05] extended the results of [Pea97] and [ADP05] and showed how both ordinary and paraconsistent answer-sets can be captured via possible worlds models due to Routley. In the case of PAS, the underlying logic $N9$ has been identified axiomatically and algebraically, and an important metalogical property - interpolation - was proved. They also showed that it is straightforward to check that HT^2 models are equivalent to Routley here-and-there models which in turn are much simpler and easy to use:

- Let M be an HT^2 model. Then the corresponding Routley model

$$M = \langle W, W^*, \subseteq, *, V \rangle$$

  consists of the same four points or worlds h, h^*, t, t^*, such that for any propositional atom p and world w, $V(p, w) = 1 \Leftrightarrow (M, w) \vDash p$, and vice versa.

Osorio and his research group have been working on characterizations as well as some other practical usages of Pstable models which are also paraconsistent in the presence of inconsistencies but not in the presence of incoherence. In their paper [OL06], the authors were able to prove that Pstable model semantics can express the stable model semantics within the class of normal programs, although they conjecture that the expressiveness of Pstable model semantics is even greater than that of stable semantics. They propose to explore the possibility to extend the theorem to the context of disjunctive programs. The main contributions were two theorems where the authors

argue that if they have a stable model of P then $M \cup M'$ is a Pstable model of a program transformed (according to a definition they present) and vice versa.

Still on P-Stable models, Eiter; Leone and Sacca in [ELS97] extended the notion of unfounded set from normal programs to disjunctive programs. They presented the notion of P-stable model for acceptable partial models for a disjunctive program by using unfounded sets. They also showed that P-stable models coincide with the 3-valued stable models (extended with a third, undefined, truth value) of Przymusinski. Afterwards, they investigated improvements of P-stable models under the principle of minimal undefinedness. This principle has been used in the context of normal programs as well as for disjunctive programs. Maximal P-stable (M-stable) models were proposed and analyzed as well as the more restrictive concept of least undefined P-stable (L-stable) models. In particular, they obtained a result stating that the M-stable models can be computed bottom-up for a normal program LP. This property is important in practice. However, L-stable models do not have this property.

3.2 Conclusions and Open issues

Conclusions

We characterize and present a new way of calculating the semi-stable models of a program, without having to perform an epistemic transformation. We do this by calculating a program's semi-stable here-and-there models and then performing a selection according to Proposition 2.10. No manner of syntactic transformation is used over the program itself but obviously the resulting models contain literals over the language $\mathcal{L}_{P^\kappa}$ and not over $\mathcal{L}_P$. This must be so because of the definition of semi-stable models itself where, since the logic $N9$ is underlying these models, there will be literals that have for instance the truth value *believed true* and as such must be epistemically marked with the belief operator K.

Semi-stable models are obtained simply by selecting semi-stable here-and-there models satisfying some conditions. This is done over a modified language of the original program, following the original definition in [SI95]. As no further restriction is imposed, our proposal not only captures semi-stable models for disjunctive logic programs, but also models for any theory composed by formulae definable for all program connectives.

We also characterized SST models as being maximally canonical answer-sets of an epistemically transformed program.

Furthermore, we have shown how one can embed semi-stable models into here-and-there interpretations as well as into Routley HT-models.

Future work

The computational complexity of this approach to semi-stable models must still be thoroughly investigated. The implementations must be improved and perhaps integrated in some existing answer-set solver.

Complexity A complete complexity study for the main reasoning tasks in our approach is still to be done, but we refer the reader to a survey Damasio presented in [DLMPea] where many complexity results for a great deal of semantics related with ASP are listed.

Table 3.1: Complexity results

Semantics	Complexity of the Propositional case
PSM / PAS	Co-NP-complete

Anyway, one question that arises is if we can argue that our characterization is in the same complexity class as ASP or if it lies one level above it.

Furthermore, in terms of the computational complexity of both implementations, we can say that from a computational point of view, our prototypical implementation of the new characterization is not the most efficient due to immaturity when compared to the first implementation of the original definition of semi-stable models due to [SI95]. Still, our approach characterization in the logic of here-and-there is more elegant and declarative than the previous ones.

In particular, the conditions that are imposed in Proposition 2.10 are of great importance for this study because the last transformation from here-and-there to semi-stable models is linear on the size of the model. Furthermore, the complexity of calculating the HT-models of a program is known already and its most difficult part resides mostly in calculating the program reduct.

After improving the implementations, making them more mature and optimized, we must also perform a study for some practical computations using both of our prototypes.

Bibliography

[ADP05] João Alcântara, Carlos Viegas Damásio, and Luís Moniz Pereira. A declarative characterisation of disjunctive paraconsistent answer sets. In *Proceedings of 16th European Conference on Artificial Intelligence (2004), Valencia, Spain (22nd–27th September 2004)*, pages 915–952, 2005.

[Ari02] Ofer Arieli. Paraconsistent declarative semantics for extended logic programs. *Annals of Mathematics and Artificial Intelligence*, 36:381–417, 2002.

[BS87] H.A. Blair and V.S. Subramanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:127–158, 1987.

[BS92] Chitta R. Baral and V. S. Subrahmanian. Stable and extension class theory for logic programs and default logics. *J. Autom. Reason.*, 8(3):345–366, 1992.

[CPV08] Pedro Cabalar, David Pearce, and Agustìn Valverde. Reducing propositional theories in equilibrium logic to logic programs. 2008.

[DLMPea] Carlos Viegas Damásio and booktitle = Handbook of Defeasible Reasoning and Uncertainty Management Systems year = 1998 pages = 241–320 publisher = Kluwer Academic Publishers Luís Moniz Pereira et al., title = A Survey of Paraconsistent Semantics for Logic Programs.

[ea08] Thomas Eiter et al. Modular hex-programs - project summary. 2008.

[EFL+01] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Tu Wien. Computing preferred and weakly preferred answer sets by meta-interpretation in answer set programming. In *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 45–52. AAAI Press, 2001.

[ELS97] Thomas Eiter, Nicola Leone, and Domenico Saccà. On the partial semantics for disjunctive deductive databases. *Ann. Math. Artif. Intell.*, 19(1-2):59–96, 1997.

[Gär90] Peter Gärdenfors. Belief revision and nonmonotonic logic: Two sides of the same coin? In *European Conference on Artificial Intelligence (ECAI 1990)*, pages 768–773, 1990.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In *ICLP*, pages 579–597, 1990.

[GL91a] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[GL91b] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GRS91] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.

[Hey30] A. Heyting. Die formalen regeln der intuitionistischen logik. *Sitz*, Berlin:42–56, 1930.

[IKH92] Katsumi Inoue, Miyuki Koshimura, and Ryuzo Hasegawa. Embedding negation as failure into a model generation theorem prover. In *CADE-11: Proceedings of the 11th International Conference on Automated Deduction*, pages 400–415, London, UK, 1992. Springer-Verlag.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002.

[LPV00] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:2001, 2000.

[LRS97] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135:69–112, 1997.

[Mon92] Luis Moniz. Well founded semantics for logic programs with explicit negation. In *European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, 1992.

[MT99] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[Nel49] David Nelson. Constructible falsity. *J. Symb. Log.*, 14(1):16–26, 1949.

[Nie98] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:72–79, 1998.

[Odi05] Sergei P. Odintsov. The class of extensions of nelson's paraconsistent logic. *Studia Logica*, 80(2-3):291–320, 2005.

[OL06] Mauricio Osorio and Alejandra López. Expressing the stable semantics in terms of the pstable semantics. In *Workshop in Logic, Language and Computation (LoLaCOM-2006)*, 2006.

[OP05] Sergei P. Odintsov and David Pearce. Routley semantics for answer sets. In *LPNMR*, pages 343–355, 2005.

[Pea97] David Pearce. A new logical characterisation of stable models and answer sets. In *Proc. of NMELP 96, LNCS 1216*, pages 57–70. Springer, 1997.

[Pea08] David Pearce. Sixty years of stable models. In Maria Garcia de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, page 52. Springer, 2008.

[PM93] Teodor C. Przymusinski and Jack Minker. Static semantics for normal and disjunctive logic programs, 1993.

[PT09] Graham Priest and Koji Tanaka. Paraconsistent logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2009.

[Rou74] Richard Routley. Semantical analyses of propositional systems of fitch and nelson. In *Studia Logica 33*, pages 283–298, 1974.

[Sak92] Chiaki Sakama. Extended well-founded semantics for paraconsistent logic programs. In *In Fifth Generation Computer Systems*, pages 592–599, 1992.

[SI95] Chiaki Sakama and Katsumi Inoue. Paraconsistent stable semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5:265–285, 1995.

[Tru04] Miroslaw Truszczynski. Knowledge representation, reasoning and declarative problem solving by chitta baral, cambridge university press, 2003. isbn 0-521-81802-8 (hardback), xiv + 530 pages. *Theory Pract. Log. Program.*, 4(2):235–237, 2004.

[Wag93] Gerd Wagner. Reasoning with inconsistency in extended deductive databases. In *Proceedings of the second international workshop on Logic programming and non-monotonic reasoning*, pages 300–315, Cambridge, MA, USA, 1993. MIT Press.

Index