FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Inconsistency Management for Traffic Regulations

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Harald Beck, BSc
Matrikelnummer 0303187

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Mitwirkung: Univ.Ass. Dipl.-Ing. Thomas Krennwallner

Wien, 01.10.2013

(Unterschrift Verfasser)

(Unterschrift Betreuung)

Meinen Eltern gewidmet

Erklärung zur Verfassung der Arbeit

Harald Beck, BSc
Preßgasse 1-3/10, 1040 Wien

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

(Ort, Datum)	(Unterschrift Verfasser)

Abstract

Road Traffic Regulations (Straßenverkehrsordnung; StVO) define legal road use on public streets. In particular, they describe the meaning and appropriation of traffic signs by means of which road use restrictions can be adopted. Traffic regulation orders become legally binding when they are announced by according traffic signs. This may lead to inconsistencies with existing regulations. In order to invalidate outdated, contradictory regulations, respective traffic signs need to be removed.

Currently, road and traffic management officials in Vienna and Lower Austria are supported by a web application that allows editing and visualizing traffic regulations on a digital street map. However, tools are lacking that help to ensure the compliance of such restrictions with the Road Traffic Regulations and with supplementary criteria. The goal of this thesis is to develop such tools based on intuitively readable, flexibly extendable formal specifications.

To this end, we first analyze the kind of information, which is expressed by traffic signs and traffic measures in a domain analysis. We develop a formal model which allows an examination of high-level relations between measures and signs and their compliance with a formal specification, without neither being bound to specific street maps, traffic signs, regulations, nor to data models of existing implementations.

We identify practically relevant tasks related to the correctness of traffic regulations and develop methods to solve them. First, we show how to evaluate a given scenario based on rules that declare the meaning of traffic signs and traffic measures. In case of an error, we can recognize which conflicts exist and where they occur. We provide diagnoses which determine the causes of conflicts. Furthermore, we show how corrections for inconsistent scenarios can be obtained by the specification itself. In addition to ensuring the correctness of measures and signs with respect to the rules, these repairs guarantee that the restrictions as announced by traffic signs correspond with the intended restrictions as expressed by traffic measures.

We define these use cases in form of reasoning tasks and analyze the computational complexity of associated decision problems. Finally, we present in detail a prototypical solution of the defined problems with Answer Set Programming (ASP), which is a rule-based programming paradigm. By exploiting the purely declarative semantics of ASP, we obtain a modular, highly flexible and maintainable implementation, which is crucial to this intrinsically complex domain.

Kurzfassung

Die Straßenverkehrsordnung (StVO) ist das Bundesgesetz, welches den Verkehr auf öffentlichen Straßen regelt. Insbesondere wird darin die Bedeutung und Anwendbarkeit von Verkehrszeichen bestimmt, mittels derer die Straßenbenutzung reguliert werden kann. Verordnungen verkehrlicher Maßnahmen werden rechtlich wirksam, indem sie durch entsprechende Verkehrszeichen kundgemacht werden. Dies kann zu Widersprüchen mit bestehenden Vorschriften führen, die dann durch die Entfernung der entsprechenden Verkehrszeichen außer Kraft gesetzt werden müssen.

Derzeit werden Beamte der Straßen- und Verkehrsverwaltung in Wien und Niederösterreich durch eine Webanwendung unterstützt, welche die Editierung und Visualisierung von verkehrlichen Maßnahmen und Verkehrszeichen auf einer digitalen Straßenkarte ermöglicht. Allerdings fehlt es noch an Werkzeugen, um die Konformität solcher Vorschriften bezüglich der StVO und weiterer Kriterien sicherzustellen. Das Ziel dieser Arbeit ist es, Methoden zu entwicklen, um dies auf Basis einer intuitiv lesbaren, flexibel erweiterbaren Spezifikation zu ermöglichen.

Dazu analysieren wir zunächst in einer Domänenanalyse, welche Informationen durch Verkehrszeichen und verkehrliche Maßnahmen ausgedrückt werden. Wir entwickeln ein formales Modell, das erlaubt, Beziehungen zwischen Maßnahmen und Verkehrszeichen und deren Konformität bezüglich einer formalen Spezifikation zu untersuchen, ohne an bestimmte Straßenkarten, Verkehrszeichen oder Vorschriften, beziehungsweise an Datenmodelle konkreter Umsetzungen gebunden zu sein.

Wir identifizieren praxisrelevante Aufgabenstellungen, die im Zusammenhang mit der Korrektheit von Verkehrsvorschriften stehen, und entwickeln Methoden zu deren Lösung. Zunächst wird gezeigt, wie Verkehrszeichen, die einer digitalen Straßenkarte zugeordnet sind, aufgrund von Regelsätzen evaluiert werden können. Im Fehlerfall wird erkannt, welche Konflikte existieren, wo diese auftreten und wodurch sie zustande kommen. Danach erläutern wir, wie Korrekturvorschläge direkt aus der Spezifikation erzeugt werden können. Diese Reparaturen stellen zum einen die Gültigkeit von Maßnahmen und Verkehrszeichen gemäß der Spezifikation sicher, und zum anderen, dass die mittels Verkehrszeichen kundgemachten Vorschriften den verordneten Maßnahmen entsprechen.

Wir definieren diese Anwendungsfälle in Form von logischen Problemen (reasoning tasks) und bestimmen die Komplexität von zugehörigen Entscheidungsproblemen. Abschließend präsentieren wir im Detail eine prototypische Lösung der beschriebenen Aufgabenstellungen mithilfe eines regelbasierten Systems (Answer Set Programming; ASP). Unterstützt durch die rein deklarative Semantik von ASP erzielen wir dabei eine modulare Implementierung, die sich durch hohe Flexiblität und Wartbarkeit auszeichnet. Dies erweist sich als besonders wichtig auf diesem inhärent komplexen Gebiet.

Danksagung

Ich möchte mich herzlich bei meinen Betreuern Prof. Thomas Eiter und Ass. Thomas Krennwallner bedanken, deren Einsatz weit über das hinaus ging, was zur Ausführung einer Diplomarbeit nötig oder zu erwarten gewesen wäre. Die zahlreichen, intensiven und lehrreichen Besprechungen haben es möglich gemacht, die zunächst vagen Ideen schrittweise zu entwickeln, zu präzisieren und schließlich auch in klarer Form darzustellen. Insbesondere danke ich Thomas Eiter für seine Beiträge zu den zwei wissenschaftlichen Artikeln, die im Rahmen dieser Arbeit veröffentlicht wurden. Dazu gehören auch die Komplexitätsresultate, die mithilfe seines Engagements eingearbeitet werden konnten. Nicht zuletzt möchte ich mich auch für die Möglichkeit bedanken, den gemeinsamen Konferenz-Artikel bei der letztjährigen JELIA (13th European Conference on Logics in Artificial Intelligence) präsentieren zu können.

Mein besonderer Dank gilt Stefan Kollarits, meinem ehemaligen Arbeitgeber bei Prisma Solutions, der mir die Arbeit an diesem Thema ermöglicht hat, sowie meinen Projektleitern Torsten Schönberg und Marlene Handschuh. Meinem Kollegen Christoph Hillinger danke ich für seine Geduld während der Domänenanalyse, und dafür, dass er mir eine detailliertere Auseinandersetzung mit der StVO erspart hat. Weiters hat mich bei Prisma Solutions Roman Steiner in der Frühphase der Arbeit unterstützt, die Aufgabenstellung zu definieren und von anderen Themen abzugrenzen.

Da auch das Studium mit dem Abschluss dieser Arbeit zu Ende geht, möchte ich die Gelegenheit nutzen, mich bei weiteren Personen zu bedanken, die nicht direkt mit dieser Arbeit in Verbindung stehen, ohne die das Studium aber einen anderen Verlauf genommen hätte. Hier möchte ich Michael Reiter und Hans Tompits nennen, die mir die Möglichkeit gegeben haben, in unterschiedlichen Lehrveranstaltungen als Tutor tätig zu werden. Ein Seminar bei Hans Tompits war es auch, welches früh im Studium mein Interesse an Answer Set Programming, bzw. an der Logik im Allgemeinen, geweckt hat.

Meinen Eltern, Stephanie und Franz Beck, möchte ich danken, dass sie mich von klein auf ermutigt haben, meine eigenen Entscheidungen zu treffen, und dass sie mich stets in meiner Entwicklung, sowie bei der Erreichung meiner Ziele unterstützt haben.

Schließlich danke ich Mirjam Moser für all ihre Liebe und Geduld.

Contents

CHAPTER 1

Introduction

In recent years, digital street maps such as Google Maps[1] and the OpenStreetMap[2] have become standard software products for geospatial applications and transport related use cases. The Graph Integration Platform (GIP), developed by *PRISMA solutions*, is such a street graph specifically designed for the integration of existing transport networks.[3] The topology provided by the GIP is used by different web applications for data management tasks in the field of road and traffic administration. One of these systems is SKAT,[4] which assists road operators and traffic authorities with the management of road use restrictions by traffic signs and related traffic regulation data.[5] Officials from Lower Austria and Vienna already use this software to collect, store and visualize such information on top of the GIP.[6]

A typical use case is the introduction of a new restriction of a certain *type*, for instance, a maximum speed limit. In order for this so-called *traffic measure* to become legally binding, according traffic signs must be posted. Then, the question is which traffic signs are required and where these signs must be posted. In general, many options are possible, depending on the street topology and on the context, i.e., existing traffic signs for other restrictions. Furthermore, outdated measures and signs can be in conflict with the new restriction and must then be retracted. However, these updates in turn may have context-dependent side effects.

So far, SKAT can propose correct traffic signs for a certain set of frequently enacted measures. That is, if a new traffic measure is created in the software by selecting an according type and drawing its shape onto the street graph, the options for according traffic sign posting are listed. The user can choose one option and then the software

[1]`http://maps.google.com`

[2]`http://www.openstreetmap.org`

[3]`http://www.prisma-solutions.at/index.php/en/solutions/gip-multimodal-graphintegrationplatform`

[4]SKAT abbreviates "Straßen-Kartographie und Administrations-Tools"

[5]`http://www.prisma-solutions.at/index.php/en/solutions/skat`

[6]`http://www.prisma-solutions.at/index.php/en/references/skat-province-of-lower-austria`

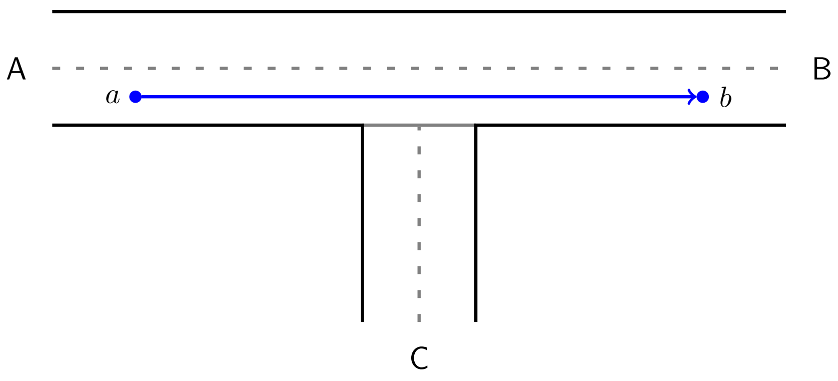


Figure 1.1: Intended extent of a 30 km/h speed limit restriction at a T-junction

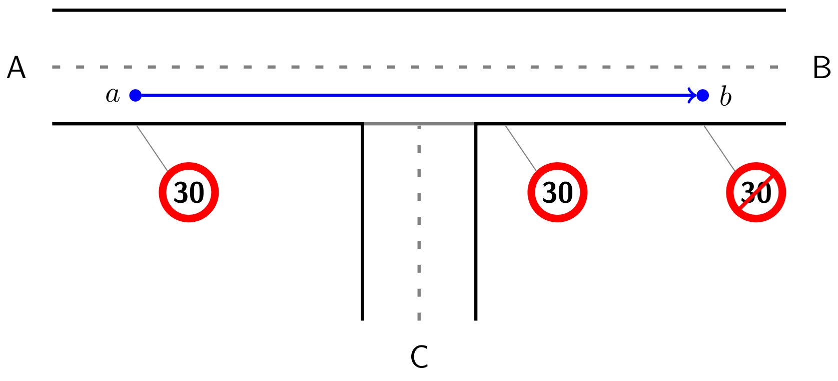


Figure 1.2: Correct sign posting to announce the 30 km/h speed limit measure

automatically adds the respective signs to the database, relates it with the measure and visualizes the posting position based on the GIP.

Example 1 Figure 1.1 schematically depicts a T-junction, i.e., a junction with three arms. Imagine a new speed limit restriction of 30 km/h shall be enacted on the horizontal street in direction from A to B as indicated by the blue line from point a to point b. To make such a traffic measure legally binding, a start sign needs to be posted at point a, and an end sign at point b. Now, consider the road users who turn right from arm C into arm B. To inform those drivers about the speed limit, a further start sign is needed directly after the junction. The correct announcement of the 30 km/h speed limit is shown in Figure 1.2. ■

Currently, the generation of such announcement proposals by SKAT only takes the street topology into account but ignores existing traffic regulation data. To ensure logical consistency and compliance with the Road Traffic Regulations (Straßenverkehrsordnung; StVO)[7] during update processes, we need a new methodology that can deal with the

[7]`http://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10011336`

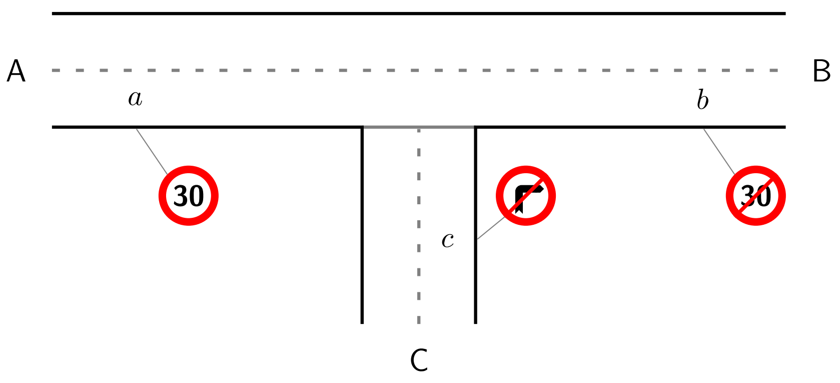


Figure 1.3: Two restrictions at a T-junction: A 30 km/h speed limit from a to b and a No Right Turn from arm C to arm B

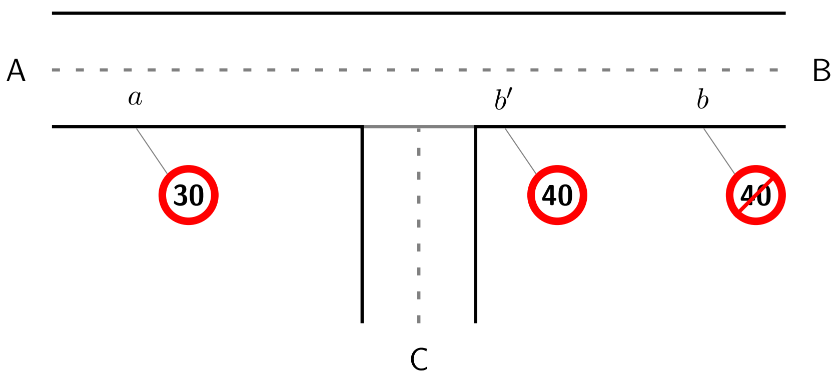


Figure 1.4: Correct sign posting for two consecutive speed limit restrictions

meaning of arbitrary combinations of traffic measures and traffic signs.

Example 2 In the previous example, the repeated start sign was necessary to inform traffic turning from arm C to arm B about the speed limit. However, if this driving direction is prohibited, like in the scenario shown in Figure 1.3, this repeated start sign is not necessary.

Now, imagine an update of the speed limit of the previous example, where the 30 km/h restriction is shortened and replaced by a 40 km/h restriction on arm B. The correct sign posting for these restriction shown in Figure 1.4. The current mechanism cannot recognize that the start sign for the 40 km/h limit at b' implicitly ends the former restriction. Thus, it would incorrectly add an explicit end sign for the 30 km/h limit at point b' in addition to the 40 km/h start sign. ■

An older desktop version of SKAT provided tailored tests for a fixed set of conflicts which were implemented by ad-hoc checks. For instance, the software could detect simple data faults like wrong posting positions (e.g., with respective to street side), one-way streets assigned to opposing directions, or start signs not having according end signs.

However, we are not aware of any implemented or formal system that allows reasoning over the complex semantics (i.e., the logic) of arbitrary combinations of traffic measures and traffic signs in a systematic, reliable way. In particular, we are not aware of any work that deals with the meaning of both kinds of information at the same time.

1.1 Motivation

To further motivate this work, we will exemplify different usage scenarios of envisaged software tools that allow to evaluate traffic regulation data against a specification like the Road Traffic Regulations. Moreover, if this specification can be flexibly adapted to capture other correctness criteria, problems of various traffic related domains can be tackled based on a uniform approach.

Road administration. Inconsistent traffic regulations create problems in daily traffic. For instance, officials are confronted with legal issues like challenging of speeding tickets when ambiguous or contradicting speed limits are announced. Even more problematic is the aspect of legal responsibility in case of accidents caused by wrong traffic sign posting. Different from that, errors in the data acquisition in traffic sign software may lead to wrong assumptions on the state of traffic regulations which in turn may lead to bad decision making in planning scenarios. Legal traffic management procedures, such as the introduction of new traffic measures, are already supported by storage and visualization tools that help to keep track of current traffic regulations and their announcements. However, mechanisms that ensure the correctness of the involved data are still in need.

Intelligent transport systems. The automatization efforts in urban infrastructures further illustrate the usefulness of tool support to ensure consistency of the data. Within the broader trend towards *smart cities*,[8] *intelligent transport systems* seek to improve the collective use of transport networks by technological aids. Our approach is based on static data but shall also provide a first step towards dynamic environments with streaming real-time traffic and traffic regulation data.

For instance, in so-called *active traffic management*, variable-message signs on motorways help smoothing traffic flows by varying speed limits based on events like traffic congestions, or weather conditions like fog or black ice. Without consistency evaluation, contradicting speed limits may be posted by operators of such message signs, leading to aforementioned legal issues.

Urban planning. Apart from legal issues, we emphasize the utility of software support in urban traffic planning, which also involves safety considerations. For instance, one wants to avoid constructing major roads near schools or parks. Likewise, heavy trucks should be banned from small roads in densely populated residential areas, if possible. A traffic planner, using a software like SKAT, may then want to inspect the effects of

[8]See, e.g., `http://www.ibm.com/smarterplanet/us/en/traffic_congestion/ideas/index.html`

such restrictions; potentially during a large-scale restructuring of urban areas, adaptions of one-way systems and the like. For these usage scenarios, it is desirable for domain experts to have an editable specification against which the database is evaluated. Then, a virtually set up scenario can also be tested against local, use case specific criteria.

Open data and crowdsourcing. Open data initiatives, such as open government data, and crowdsourced databases like the OpenStreetMap provide freely available data sets, which will very likely contain different kinds of errors for various reasons. For instance, geotagged user data may be noisy, imprecise, or incomplete. In particular, the records of different users may contradict each other. With a tool that can detect, diagnose and semi-automatically repair inconsistencies, the quality of such databases might be drastically improved.

1.2 Goals and Scope

To the best of our knowledge, systematic inconsistency management for traffic regulations has been an entirely unexplored domain prior to this thesis. As a result, tools that help to ensure the correctness of traffic measure data and corresponding traffic sign posting (beyond simple ad-hoc checks) are lacking. The aim of this thesis is to lay the foundation for an extension of SKAT such that the stored data can be guaranteed to comply with a flexibly editable specification. In addition to the Road Traffic Regulations this specification may include expert knowledge such as traffic planning experience, safety requirements and also common sense.

Central to our understanding of inconsistency is the notion of a *conflict*, which occurs whenever a set of traffic measures or traffic signs (on a digital street map) contradict the given specification. We intend to provide automated assistance for the following major use cases.

1. *Evaluation.* Determine whether the database is consistent, i.e., whether the stored traffic signs and traffic measures are compliant with the specification. In case of inconsistencies we want to know which conflicts occur, and where.
2. *Correspondence.* Traffic measures and traffic signs are two ways to express restrictions on road use. We need to ensure that given data of both kinds express the same restrictions.
3. *Diagnosis.* In order to resolve inconsistencies, we need an understanding of their causes. Thus, we want to determine those signs and measures that lead to conflicts.
4. *Repair.* The system should suggest repairs for inconsistent scenarios, i.e., additions and removals of measures and signs such that the updated database is consistent.

Due to the complex nature of traffic regulations we cannot expect any specification to be sufficient for all scenarios. Consequently, a systematic way must be found for an implementation such that the underlying specification can be flexibly adapted and extended, potentially even by domain experts at runtime.

1.3 Contributions

In this thesis, we lay the groundwork for software support in advanced use cases of inconsistency management for traffic regulations, with the following main contributions.

Domain Analysis. Towards a model of the traffic regulation domain, we first discern in a comprehensive domain analysis relevant aspects in traffic measure and traffic sign information. We categorize different classes of errors and identify major use cases to deal with them: consistency evaluation, diagnosis, repair, correspondence and strict repair.

Formal Model for Traffic Regulations. Based on this study, we develop a logic-based model, where we regard traffic signs and traffic measures as two input languages expressing road use restrictions in form of so-called effects. We abstract away from specific street maps by considering directed graphs. We also generalize the discussed measures and signs by employing edge and node labels to reflect them, as well as their intended effects and potential conflicts. We assume that the traffic regulation and supplementary specifications are encoded as mappings between these labels by means of formulas in a predicate logic.

As a result, we can focus on high-level relations between measures and signs and their compliance with a formal specification without being bound to specific kinds of input, legal interpretations, particular data representations of available street maps, or the expressiveness required to describe certain properties. We also present an exemplary instantiation of this model using Answer Set Programming [5, 23] as underlying logic.

Reasoning Tasks. On top of the formal model, we define the intended use cases in form of reasoning tasks and investigate several practically relevant special cases of so-called strict repairs, where traffic measures and traffic signs can be repaired at the same time. We define and characterize contexts due to which conflicts arise and examine relations between diagnoses and repairs. This leads to the observation of challenges for an implementation of the defined tasks.

Computational Complexity. We present in more detail the complexity results established in [3]. For our reasoning tasks, we study first-order logic under domain closure and three classes of Answer Set Programs, namely stratified programs, normal programs, and disjunctive programs [17]. For ASP, the complexity increases in that order.

Implementation. We present in detail a prototypical, elegant implementation for our reasoning tasks with Answer Set Programming, using the solver DLV.[9] We explain and demonstrate the benefits of automated reasoning and the high readability which comes with a rule-based implementation for a logic-oriented domain. Moreover, we show how the fully declarative semantics of ASP assists modular composition and thus enables a high degree of flexibility, which is crucial in this intrinsically complex domain.

[9] `http://www.dlvsystem.com`

1.4 Thesis Organization

After reviewing some technical background in Chapter 2, we first analyze the traffic regulation domain in Chapter 3. We can then build a formal model in Chapter 4 to lay the foundation towards a systematic solution of the intended use cases which we define in form of reasoning tasks in Chapter 5. After investigating the computational complexity of according decision problems in Chapter 6, we then explain in Chapter 7 in detail how the reasoning tasks can be implemented with Answer Set Programming. Finally, in Chapter 8 we will draw conclusions and point towards future work. The Appendix contains the source code of the presented implementation and a translation table for relevant domain vocabulary.

Parts of the result of this thesis have been presented, in preliminary form, at the AAAI 2012 Workshop on Semantic Cities [2] and at the 13th European Conference on Logics in Artificial Intelligence [3].

CHAPTER 2

Preliminaries

In this chapter, we review the two main approaches to diagnosis in the context of knowledge-based systems, namely consistency-based diagnosis and abductive diagnosis. Then, we give a brief introduction to declarative programming and Answer Set Programming. Finally, we review basic definitions of computational complexity theory.

2.1 Diagnosis

Diagnosis is the task to determine reasons for the unexpected behaviour of a system. Given exceptional or abnormal findings, one wants to explain them based on background knowledge. For instance, in medical diagnosis, the problem is to identify diseases by their symptoms. The result of this inference from effects to their causes is also called a diagnosis, i.e., a set of assumptions that account for the observations. Within the field of logic in artificial intelligence, diagnosis is usually understood as the task to infer reasons for abnormal behaviour of a fixed system as described by a formal model. Model-based diagnosis [9, 35] is traditionally divided into two (related [8, 27]) categories, *consistency-based diagnosis* [11,44] and *abductive diagnosis* [6,7,39,41,42]. Informally, in consistency-based diagnosis one wants to determine those hypotheses (causes) that are logically consistent with the background knowledge and the observations (effects) that indicate a discrepancy between the expected and the actual behaviour of the system. Abductive diagnosis is a stronger concept which additionally requires that the observations logically follow from the assumptions being made. We will review the respective definitions in line with [15] and assume that some version of predicate logic with negation is fixed.

Definition 1 (Diagnostic problem) *A* diagnostic problem *$\mathcal{P}$ is a triple $\langle H, T, O \rangle$, where H is a set of ground atoms, called the* hypotheses*, T is a set of formulas, called the* theory*, and O is a set of ground literals, called the* observations*.*

The consistency-based approach to diagnosis is usually presented in the context of a fixed, well-defined system consisting of components, where each one hypothetically works abnormally. Typically, the following assumptions are made. The logical theory describes the normal behaviour of these components. Observations, such as the input and output values of its components, are used to reason about the system's state. In the presence of *abnormal behaviour* of any of the components, these observations shall give insight into which components work abnormally. A set of components is defined to be a consistency-based diagnosis, if the assumption of their abnormal behaviour is logically consistent with the theory and the observations. Therefore, the set of hypotheses usually consists of an *abnormality assumption* $ab(c)$ for each component c (represented as constant) of the system. Here, ab is a designated predicate symbol.

Definition 2 (Consistency-based diagnosis) *Let $\mathcal{P} = \langle H, T, O \rangle$ be a diagnostic problem. A* consistency-based diagnosis for O *(in $\mathcal{P}$) is a set of* assumptions $\Delta \subseteq H$, *s.t.*

$$T \cup O \cup \Delta \cup \{\neg h \mid h \in H \setminus \Delta\} \quad \not\models \quad \bot. \tag{2.1}$$

The set Δ represents those components which are assumed to function abnormally. The *negated complement* $\{\neg h \mid h \in H \setminus \Delta\}$ completes this assumption by explicitly stipulating that all other components work correctly. If this assumption is consistent with the theory and the observations, then Δ is a consistency-based diagnosis for O. Normally, one is not interested in arbitrary diagnoses, but plausible ones. Following *Occam's razor*, one is typically looking for minimal diagnoses, e.g., with respect to subset inclusion.

Consistency-based diagnosis is usually defined towards the localization of malfunctioning components in a fixed system. We presented a more general definition, which emphasizes the applicability for other domains.

Example 3 Suppose we are given the following domain knowledge, structurally resembling an example in [40].

(i) *To prove theorems, one must work at university or at a big tech corporation.*
(ii) *To make money, one must work at a big tech corporation or pursue some other career in industry.*

This can be formalized propositionally in a straightforward way:

$$\begin{array}{rcl} T_1 & = & \{\ \mathit{theorems} \rightarrow \mathit{uni} \vee \mathit{big\text{-}corp} \\ & & \quad \mathit{dollars} \rightarrow \mathit{industry} \vee \mathit{big\text{-}corp}\ \} \end{array}$$

Let our goal be to prove theorems, i.e., $O = \{\mathit{theorems}\}$. The hypotheses are

$$H \quad = \quad \{\ \mathit{uni}, \mathit{industry}, \mathit{big\text{-}corp}\ \}.$$

The set $\{\mathit{uni}\}$ is a consistency-based diagnosis for O, as $\Delta \cup \{\neg \mathit{industry}, \neg \mathit{big\text{-}corp}\}$ is consistent with $T_1 \cup O$. Similarly, $\{\mathit{big\text{-}corp}\}$ satisfies the criterion. However, $\{\mathit{industry}\}$ is not consistent with $\{\mathit{theorems}\}$, i.e., $T_1 \cup O \cup \{\mathit{industry}\} \cup \{\neg \mathit{uni}, \neg \mathit{big\text{-}corp}\} \models \bot$.

Note that the entire set of hypotheses H is itself a consistency-based diagnosis for all observations $\{\mathit{theorems}\}$, $\{\mathit{dollars}\}$, and $\{\mathit{theorems}, \mathit{dollars}\}$; albeit not an informative one. Thus, we are normally interested in $\subseteq$-minimal diagnoses. Furthermore, we note that the empty set is not a consistency-based diagnosis for any of these observations. ■

The fact that we spoke of *goals* rather than observations brings us to the relation between a diagnostic task and the kind of knowledge being modelled.

Example 4 We rephrase the above example as follows.

(i) At universities people prove theorems.
(ii) In industrial careers people make money.
(iii) In big tech corporations people prove theorems and make money.

This presentation suggests a different way of modelling.

$$\begin{aligned} T_2 \;=\; \{\; & \mathit{uni} \rightarrow \mathit{theorems} \\ & \mathit{industry} \rightarrow \mathit{dollars} \\ & \mathit{big\text{-}corp} \rightarrow \mathit{theorems} \wedge \mathit{dollars} \;\} \end{aligned}$$

Let us consider $O = \{\mathit{theorems}\}$ and H as before. Again, $\{\mathit{uni}\}$ and $\{\mathit{big\text{-}corp}\}$ are consistency-based diagnoses for O. However, $\{\mathit{industry}\}$ and the empty set are also diagnoses now, which is not intended. ■

The problem with T_2 in the above example are the implications from hypotheses to observations; in other words, from causes to effects. Such causal models do not (automatically) fit the consistency-based approach. However, a theory like T_2 is suitable if we are looking for hypotheses which are not only *consistent* with the observations (or goals), but which also *entail* them. This requirement that observations must logically follow from the assumptions is the idea behind abductive diagnosis.

Definition 3 (Abductive diagnosis) *Let $\mathcal{P} = \langle H, T, O\rangle$ be a diagnostic problem. An* abductive diagnosis for O *(in $\mathcal{P}$) is a set of* assumptions $\Delta \subseteq H$*, such that*

$$T \cup \Delta \;\not\models\; \bot, \textit{ and} \tag{2.2}$$
$$T \cup \Delta \;\models\; O. \tag{2.3}$$

Abductive diagnoses are also often called *explanations* since they give sufficient reasons to infer the observations.

Example 5 (cont'd) Consider again the diagnostic problem $\langle H, T_2, O\rangle$ of Example 4. As intended, the empty set is not an abductive diagnosis for $O = \{\mathit{theorems}\}$, but $\{\mathit{uni}\}$ and $\{\mathit{big\text{-}corp}\}$ are. In fact, all consistency-based diagnoses for O in $\langle H, T_1, O\rangle$, i.e., all subsets of H including at least *uni* or *big-corp*, are also abductive diagnoses for O in $\langle H, T_2, O\rangle$. On the other hand, $\{\mathit{uni}\}$ and $\{\mathit{big\text{-}corp}\}$ are not abductive diagnoses for O in $\langle H, T_1, O\rangle$. ■

What we see in these examples is that the domain model and the diagnostic task cannot be viewed separately. The choice which diagnostic approach to use depends on the kind of knowledge which it requires [35, 40]. Consistency-based diagnosis is usually the right choice when the normal behaviour of a system can be specified, and if there is no explicit *fault model* relating malfunctioning components to observable effects. If the relation between causes and effects (or diseases and symptoms) can be directly modelled, or when the notion of *goals* is more natural than *observations*, the domain usually fits abductive diagnosis.

Note that domain knowledge can be encoded in different ways, regardless of its nature. The theory T_1 in Example 3 is also a causal model, only encoded in such a way that it works for consistency-based diagnosis in the intended way.

2.2 Answer Set Programming

Answer Set Programming (ASP) [24] is a modern logic-oriented programming paradigm, gaining increasing popularity [5] due to the availability of efficient solvers like DLV [30] and the Potassco suite [21]. We first put ASP in context by contrasting it with more prominent programming languages and paradigms. Then, we review its formal definition. A general introduction into Answer Set Programming, various extensions and prominent solvers can be found in [17].

2.2.1 Declarative Programming

Programming languages can be categorized in different ways. Prominent paradigms include *object-oriented* programming (Java, C++, C#), *functional* programming (Haskell, LISP), and *logic-oriented* programming (Prolog). Many languages combine aspects of different paradigms. For instance, C# and Scala combine object-oriented programming with functional programming.

The relevant distinction for our considerations is whether a language is *imperative* or *declarative*. In imperative languages, the programmer states *how* the computation is carried out by defining control flow and sequences of statements altering the program's state. In Kowalski's terms, where "Algorithm = Logic + Control" [28], imperative programming focuses on the *control*. By contrast, declarative languages abstract from control flow and instead allow the programmer to directly specify *what* shall be computed. Declarative languages thus focus on the *logic* of programs.

Similar to other distinctions, there is no sharp line between imperative and declarative languages. However, functional and logic-oriented languages are typically largely declarative. Answer Set Programming is a logic-oriented approach which is purely declarative in the sense that it is free from side effects and features no notion of control flow or means to direct how the computation shall be done.

Example 6 Consider the following imperatively written function in Python, which determines whether the argument `x` represents a bird.

```
def bird(x):
  if x == 'tweety': return True
  elif x == 'sam': return True
  elif penguin(x): return True
  else: return False
```

The function has no assignments but explicitly defines the control flow: *First* it tests whether `x` is a string `'tweety'`. If this check fails, *then* `x` is tested for equality with `'sam'`, and so on.

The following Clojure *expression* is functional and directly specifies the intended meaning that `x` is a bird, if it is `"tweety"`, `"sam"` *or* a penguin. It is evaluated by function applications[1] from inside out.

```
(defn bird? [x]
  (or (= "tweety" x)
      (= "sam" x)
      (penguin? x)))
```

The corresponding answer set program comprises the following three rules in *arbitrary* order. We have no strings, but constants `tweety` and `sam` and unary predicate symbols `bird` and `penguin`. The uppercase `X` denotes a variable.

```
bird(tweety).
bird(sam).
bird(X) :- penguin(X).
```

The first two lines are so-called *facts*. The third rule says that if something is a penguin, then it is a bird. The order in which these three lines are specified does not play a role. This set-oriented approach assists modular composition. Suppose we hear that "Larry" and "Charlie P" are also (certain kinds of) birds, as well as subjects of ornithologists' study. With Answer Set Programming, we would leave the previous specifications untouched and simply add new rules. Clearly, it depends on the *context* whether something is referred to as "bird". Later, we will encounter such flexibility issues where we demonstrate the usefulness of ASP in this regard. ∎

Next, we formally introduce Answer Set Programming.

2.2.2 Logic Programming under the Answer Set Semantics

Logic-oriented programming under the answer set semantics [23], or Answer Set Programming, for short, is formally defined as follows.

Syntax

We use three disjoint sets of symbols over a first-order vocabulary Φ; predicates $\mathcal{P}$ (countable), constants $\mathcal{C}$ (countable), and variables $\mathcal{V}$ (infinite). Members of $\mathcal{C}$ and $\mathcal{V}$ are called *terms*. If p is a predicate and $t_1, \ldots, t_k$ are terms, then $p(t_1, \ldots, t_k)$ is an *atom*

[1]To be precise, `or` is implemented as macro in Clojure, but it could be a function.

Name	Restriction
disjunctive	(none)
positive	$n = m$
normal	$k \leq 1$
definite	$k = 1$
horn	$k \leq 1, n = m$
definite horn	$k = 1, n = m$
unary	$k = 1, n = m \leq 1$
fact	$k = 1, n = m = 0$
constraint	$k = 0$

Table 2.1: Different classes of rules, resp. programs

with *arity* k. If $k = 0$, we speak of *propositional atoms*. Variable-free atoms are called *ground atoms*. A *(classical) literal* ℓ is an atom a or a *classically negated* atom $\neg a$. The *complement* of a literal a is $\neg a$, and vice versa. A *negation as failure (NAF) literal* is either a literal ℓ, or a *default negated* literal *not* ℓ, which evaluates to true if ℓ is not provably true, i.e., if the truth of ℓ cannot be concluded.

A *rule* r is an expression of the form

$$a_1 \vee \cdots \vee a_k \quad \leftarrow \quad b_1, \ldots, b_m, \text{ not } b_{m+1}, \ldots, \text{ not } b_n, \tag{2.4}$$

where $k, m, n \geq 0$, and all a_i and b_j are literals. The set of literals $a_1, \ldots, a_k$ is called the *head* of r, denoted $H(r)$. The *body* $B(r) = B^+(r) \cup B^-(r)$ of r is the union of the *positive body* $B^+(r) = \{b_1, \ldots, b_m\}$ and the *negative body* $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. Table 2.1 defines different classes of rules.

An *(extended disjunctive logic) program* (EDLP) P is a finite set of rules of the form (2.4). *Extended logic programs* do not allow disjunctive heads, i.e., $k \leq 1$. *Normal logic programs* additionally disallow the use of classical negation. *Generalized logic programs* are also free of classical negation but allow default negation in the heads of rules. If a certain restriction applies to all rules of a program, the rule name of Table 2.1 extends to the respective program name. For instance, an EDLP is called *positive*, if all rules are positive.

Example 7 Let P be the program consisting of the following two rules r_1 and r_2:

$$\begin{array}{ll} a \vee b \leftarrow c, \text{ not } d. & (r_1) \\ c. & (r_2) \end{array}$$

The program is propositional and thus ground. The head of the first rule consists of the (propositional) atoms a and b, i.e., $H(r_1) = \{a, b\}$. The positive body contains only c, the negative body consists of the default negated literal d. That is, $B^+(r_1) = \{c\}$ and $B^-(r_1) = \{d\}$. Rule r_2 consists only of a head, i.e., $H(r_2) = \{c\}$ and $B(r_2) = \emptyset$ and thus is a fact. We write facts without "$\leftarrow$" as usual. ■

Semantics

Similar to atoms, rules and programs are called *ground*, if they are variable-free, respectively. The semantics of Answer Set Programs is defined for variable-free programs. Therefore, we first define the *ground instantiation* of a program.

Let P be an EDLP. The *Herbrand universe* of P, denoted HU_P, is the set of constant symbols appearing in P. If P does not contain a constant symbol, HU_P comprises a single arbitrary constant from $\mathcal{C}$. The *Herbrand base* of P, denoted HB_P, is defined as the set of all ground literals that can be constructed using predicate symbols that appear in P and constant symbols that appear in HU_P. If $r \in P$ is a rule, then replacing every variable of r by some constant of HU_P yields a *ground instance* of r. By $ground(P)$ we denote the set of all ground instances of rules in P.

We now define the semantics of *positive ground EDLPs*. An *interpretation* $I \subseteq HB_P$ is called *consistent* iff $\{p, \neg p\} \nsubseteq I$ for every atom $p \in HB_P$. We say I *satisfies* a rule $r \in P$, written $I \models r$, if $H(r) \cap I \neq \emptyset$ whenever $B^+ \subseteq I$ and $B^- \cap I = \emptyset$. That is, if no default negated atom of the rule occurs in I and all literals of the positive body occur in I, then some head literal must occur in I. If I satisfies all rules of a positive program P, we say I is a *model* of P, and also write $I \models P$. If a model exists, the least one with respect to set inclusion is called the *answer set* of P.

This definition is extended to programs with negation as failure, as follows. The *(Gelfond-Lifschitz) reduct* [23] of an EDLP P *relative to an interpretation* $I \subseteq HB_P$, denoted P^I, is the positive program obtained from P by (i) deleting every rule r where $B^-(r) \cap I \neq \emptyset$, and (ii) deleting the negative body from the remaining rules. An *answer set* of an extended disjunctive logic program P is an interpretation $I \subseteq HB_P$ such that I is an answer set of P^I.

Example 8 (cont'd) We consider for program P of Example 7 the following three interpretations: $I_a = \{a\}$, $I_c = \{c\}$, and $I_{ac} = \{a, c\}$. Since $B^+(r_1) = \{c\} \nsubseteq I_a$, $I_a \models r_1$ holds. However, $I_c \not\models r_1$, since $B^+(r_1) \subseteq I_c$ and $B^-(r_1) \cap I_c = \emptyset$, but $H(r_1) \cap I_c = \emptyset$. Finally, $I_{ac} \models r_1$, since $B^+(r_1) \subseteq I_{ac}$ and $B^-(r_1) \cap I_{ac} = \emptyset$, and $H(r_1) \cap I_{ac} = \{a\} \neq \emptyset$.

The reduct for I_{ac} is $P^{I_{ac}} = \{a \vee b \leftarrow c;\ c\}$. The negative body from r_1 is deleted, since it does not occur in I_{ac}. (Otherwise, the entire rule would have been deleted.) The interpretation satisfies both rules of the reduct, and therefore it is a model of the reduct, i.e., $I_{ac} \models P^{I_{ac}}$. To see that it is also the least model, we test the subsets $\emptyset$, I_a and I_c. The latter does not satisfy $a \vee b \leftarrow c$, the other interpretations both do not satisfy the fact c. We conclude that I_{ac} is a $\subseteq$-minimal model of $P^{I_{ac}}$ and therefore is an answer set of P. Similarly, $I_{bc} = \{b, c\}$ is the program's second answer set. ■

Orthogonal to the ASP variants as defined by Table 2.1, other restrictions and properties of Answer Set Programs have been studied in the literature. We review some of them next.

2.2.3 Restrictions

The *dependency graph* $D(P)$ of an EDLP P is a directed graph $(V, E^+ \cup E^-)$, where the set of nodes V comprises the predicates occurring in P, denoted *pred*(P). The edges are obtained as follows. For every rule $r \in P$, where $p \in H(r)$,

(i) $(p, q) \in E^+$, if $q \in B^+$, and

(ii) $(p, q) \in E^-$, if $q \in B^-$.

Let $D(P) = (V, E^+ \cup E^-)$ be the dependency graph of a program P. A predicate cycle $p_1, \ldots, p_n, p_1$ with edges (p_i, p_{i+1}), where $1 \leq i \leq n$, and (p_n, p_1) in E^+ is called a *head-cycle*, if two (or more) of its predicates p_i occur in the head of some rule $r \in P$. If a program does not have a head-cycle, it is called *head-cycle free* [4].

A *stratification* of P is a set partition $\Sigma = \bigcup_{i \geq 1} P_i$ of *pred*(P), such that for each $p \in P_i$ and $q \in P_j$,

(i) $(p, q) \in E^+$ implies $i \geq j$, and

(ii) $(p, q) \in E^-$ implies $i > j$.

If such a stratification Σ exists we call P a *stratified program.* The subsets P_i are called the *strata* of P (with respect to Σ). Stratified programs allow for a more efficient evaluation. In the absence of integrity constraints and strong negation, they have at least one answer set.

2.3 Computational Complexity

We will now recall some important concepts of computational complexity theory [1, 38].

A *string* is a sequence of symbols over some alphabet and a *language* $\mathcal{L}$ is a set of strings. We usually consider binary strings, i.e., the alphabet $\{0, 1\}$. Given a string S, the *decision problem* for a language $\mathcal{L}$ asks whether the language contains the string, i.e., "$S \in \mathcal{L}$?". The complexity class P (respectively NP) contains all decision problems solvable on a deterministic (respectively nondeterministic) Turing machine in polynomial time. That is, the number of steps carried out by the respective Turing machine to decide whether a given string S is in $\mathcal{L}$ is bounded by a polynomial in the size of S.

The *complement* $\overline{\mathcal{L}}$ of a language $\mathcal{L}$ is the set of strings over the same alphabet not contained in $\mathcal{L}$. Given a complexity class C, we define the class co-$C = \{\mathcal{L} \mid \overline{\mathcal{L}} \in C\}$. We recall that $\mathsf{P} \subseteq \mathsf{NP} \cap \text{co-}\mathsf{NP}$.

Furthermore, we obtain classes by employing subroutines, so called *oracle queries.* By P^O (respectively NP^O) we denote the class of decision problems computable in (non-deterministic) polynomial time, provided a subroutine for deciding a problem in a class of languages O is available. (If O has complete problems the particular problem is not of interest.) A subroutine call to an oracle is considered to take constant time. Additionally, $\mathsf{P}^O_{\|}$ is the restriction of P^O that all oracle queries are independent of each other, i.e., they are evaluable in parallel. By $\mathsf{P}^{O}_{\|[k]}$ we denote the restriction on $\mathsf{P}^{O}_{\|}$ such that the parallel oracle calls can be limited to k consecutive rounds.

The *Polynomial Hierarchy* is defined as $\mathsf{PH} = \bigcup_{k \geq 0} \Sigma^p_k$, where

- $\Sigma^p_0 = \Pi^p_0 = \Delta^p_0 = \mathsf{P}$, and for $k \geq 1$
- $\Sigma^p_k = \mathsf{NP}^{\Sigma^p_{k-1}}$
- $\Pi^p_k = \text{co-}\Sigma^p_k$, and
- $\Delta^p_k = \mathsf{P}^{\Sigma^p_{k-1}}$.

It is easily seen that $\Sigma^p_1 = \mathsf{NP}$, $\Pi^p_1 = \text{co-}\mathsf{NP}$ and $\Delta^p_2 = \mathsf{P}^{\mathsf{NP}}$.

The class EXP consists of problems which are computable on a deterministic Turing Machine in exponential time. Similarly, NEXP is the exponential analog of NP. We have $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NEXP}$ and it is believed that these inclusions are strict. We note that PSPACE is equivalent to its nondeterministic analog $\mathsf{NPSPACE}$.

CHAPTER 3

Domain Analysis

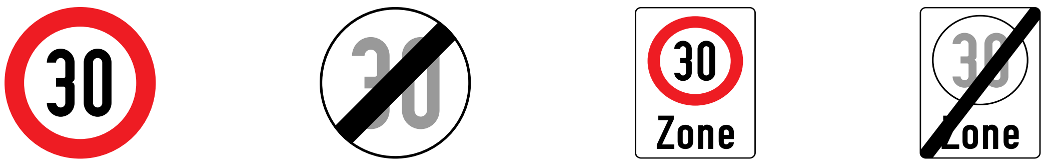
In this chapter we analyze the problem domain of traffic regulations. After introducing our vocabulary for relevant street attributes we investigate the meaning of frequently used traffic signs[1] and corresponding traffic measures, and we point out potential conflicts which may arise from their application. Based on these observations, we will outline inconsistency management tasks and derive criteria towards their implementation. Finally, we argue why Answer Set Programming is a suitable choice.

3.1 Street Maps

A street (or road) typically consists of one or more *lanes*, each of which is associated with a unique *direction*. Main roads often have more than one lane per direction. Additional lanes, not available for regular traffic, are *frontage roads*, which are usually physically separated from the outermost lane. Streets cross at *junctions* (or crossroads), where one can *turn* into other *arms*. The *incoming streets* of a junction are those arms ending there. Roundabouts can be viewed as special types of junctions. Different from these physical properties are legal distinctions, i.e., whether a street is a normal road, within a residential area, a motorway, etc. We will discuss such regulatory attributes later.

When writing software, one must design based on the intended use cases. Consequently, well-known digital street maps focus on different aspects. The OpenStreetMap, for instance, is designed for flexible, crowdsourced editing and generic data collection. In this thesis, we are interested in a light-weight street model that allows to locate traffic signs and related information. To keep things as simple as possible, we will not consider roundabouts or frontage roads and assume that every street consists of exactly two lanes; one per direction. These restriction will allow us to focus on the knowledge-based tasks around inconsistency management.

[1]Figure source: `http://de.wikipedia.org/wiki/Bildtafel_der_Verkehrszeichen_in_%C3%96sterreich`



(a) Start sign (b) End sign (c) Zone start sign (d) Zone end sign

Figure 3.1: Traffic signs for 30 km/h maximum speed limits

3.2 Traffic Measures and Traffic Signs

A *Road Traffic Regulations* is a federal law describing how road users can use the street and how these usages can be restricted by means of *traffic signs*. The legal act to change such restrictions by means of adding new traffic signs, or by removing existing ones, is called a *traffic regulation order*, which comes in form of a document describing (in natural language) a *traffic measure* that has to be taken to reach a desired effect. This measure needs to be *announced* by means of traffic signs and becomes legally binding as soon as the corresponding signs are posted on the street. We view road markings as special cases of traffic signs.

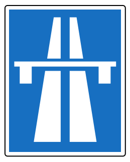
By a *measure type*, respectively *sign type*, we understand the name of its category, e.g., highway, residential area, speed limit, No Right Turn, one-way streets, etc. Each measure type is associated with a topological extent or *shape*, which can be either a point, a line or a zone. By the word *traffic regulation* we usually mean the specific restrictions as expressed by traffic signs or traffic measures.

Example 9 Figure 1.2 (page 2) showed the correct announcement for a 30 km/h speed limit measure across a T-junction. The Road Traffic Regulations state the according rules of sign posting, e.g., the necessity for the repeated start sign. The respective measure type is “speed limit of 30 km/h,” and has a linear shape. The depicted sign types are “start (respectively end) sign for a 30 km/h speed limit.” ■

This work is based on the Austrian Road Traffic Regulations and its potential measures and signs. However, we aim to provide a more general solution and thus focus on principles that are not bound to regional differences.

We will now investigate frequently used Austrian traffic signs and describe their meaning and implications, i.e., the restrictions they impose on road use. This will also reveal inconsistencies that might occur due to certain combinations.

Speed limits. Speed limits can come in two forms, *normal* (or linear) speed limits (Figures 3.1a, 3.1b), and speed limit *zones* (Figures 3.1c, 3.1d). There are two main differences. First, the announcement of a normal speed limit needs to be repeated after every junction. In other words, without a repeated start sign at the begin of another road’s arm, the speed limit restriction ends after turning. This is not the case for speed limit zones, which take effect until a sign is reached that ends the according restriction.


(a) Highway

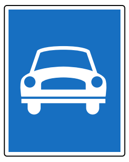

(b) Motorway

Figure 3.2: Announcements for major roads

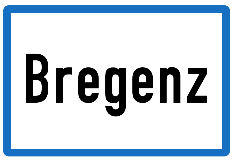
The second difference lies in the *default* character of speed limit zone. When arriving at a start sign like in Figure 3.1a, the previous speed limit is overruled. However, if the according speed ends (Figure 3.1b), then the default speed limit takes effect again. That is, if a normal start sign is placed within a speed limit zone, it only *temporarily* overrules the latter. If the end of the normal limit is reached, the zone limit is active again. However, it is good practice to remind road users of that by another zone start sign.

Like for most signs, the *validity* of speed limits can be restricted by *additional panels* below the main sign, stating the conditions or the road users the restriction applies for. A typical validity restriction of speed limits is on heavy trucks which usually are assigned a lower maximum speed then other vehicles. Similarly, speed limits are often bound to weather conditions, particularly on major roads.

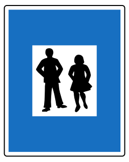
Since a start sign of a speed limit overrules the previous speed limit (permanently or temporarily), the maximum speed limit is unique for every path. However, as argued earlier, the speed limit must be unique on every position regardless of the taken path. In case of insufficient announcements, this might not be the case, as we have seen in Example 2 (page 3). Further, in accumulated traffic *measure* information there might as well be contradicting speed limits at the same position, if a traffic regulation order does not explicitly overturn a previous one that is in conflict with it.

Highways and motorways. On highways (Figure 3.2a) there is an obligatory minimum speed of 60 km/h, implying that only vehicles with a maximum design speed exceeding 60 km/h are allowed for traffic. For different vehicle types, different default speed limits are valid. Motorcycles and passenger cars are permitted a driving speed of 130 km/h, unless they have a gross vehicle weight greater than 3.5t (tons), are drawing a trailer, or using studs. In these cases, the maximum allowed speed is reduced to 100 km/h, which is also the limit for omnibuses. For passenger cars with a gross vehicle weight of at most 3.5t drawing heavy trailers, or if the total weight of car and trailer exceeds 3.5t, the limit is 80 km/h. This is also the maximum speed limit for all vehicles with gross vehicle weight greater than 3.5t, except omnibuses. These default restrictions implied by highways may be overruled by explicit speed limits as explained above.

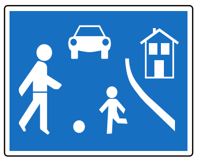
Likewise, motorways (Figure 3.2b) allow traffic only for motor vehicles and define a default speed limit of 100 km/h, respectively 80 km/h, for similar groups as for highways. Obviously, highway and motorway are mutually exclusive measure types. That is, no street can be both a highway and a motorway.



(a) Residential area

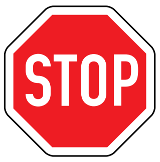


(b) Pedestrian zone

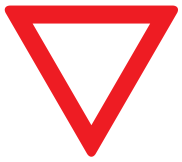


(c) Home zone

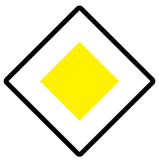
Figure 3.3: Traffic signs related to inhabited areas



(a) Stop



(b) Give Way



(c) Priority road

Figure 3.4: Traffic signs that regulate the right of way

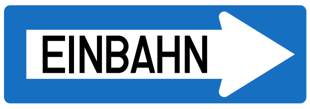
Residential areas, pedestrian zones and home zones. Residential areas imply a default speed limit of 50 km/h, unless there is an explicit different speed limit. The start sign of a residential area (Figure 3.3a) does not overrule an existing speed limit, except for the default speed limit in non-residential areas, which is 100 km/h.

Pedestrian zones (Figure 3.3b) can only be announced within residential areas and prohibit use for all road participants except pedestrians, public service vehicles and emergency vehicles. For allowed vehicles there is a maximum speed limit of 5 km/h. For home zones (Figure 3.3c) similar conditions hold.

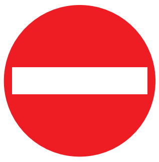
We observe that some traffic measures require others. There cannot be a pedestrian zone (or home zone) without a residential area at the same place. Also, pedestrian zones and home zones are mutually exclusive. In Austria, a highway or a motorway must not lead through a residential area. (And thus, as a logical consequence, highways and motorways cannot cross pedestrian zones or home zones.)

Stop, Give Way and priority road. The next group of traffic signs deals with the right of way. When arriving at a Stop sign (Figure 3.4a), usually before a junction, one must stop the car before continuing to drive in either direction. By stopping, one always gives way to other road users. A weaker form of signalling lower priority is done by the Give Way (or Yield) sign (Figure 3.4b). Here, road users do not necessarily have to stop the car but must give way to crossing traffic.

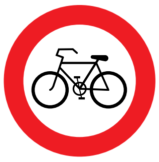
In contrast to this, by the traffic sign shown in Figure 3.4c one is informed about driving on a priority road. That is, when crossing other streets, road users of the other arms must give way. As a consequence, all incoming streets must have a Stop sign or a Give Way sign right before the junction. This example illustrates that traffic signs might imply the necessity of other traffic signs on *other* roads.



(a) One-way street

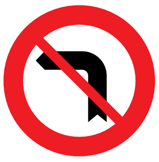


(b) No Entry

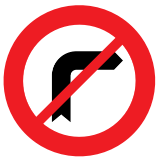


(c) No Bicycles

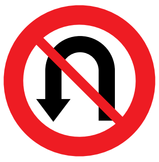
Figure 3.5: Traffic signs prohibiting traffic



(a) No Left Turn

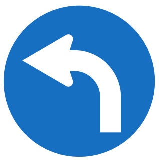


(b) No Right Turn

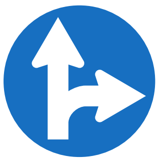


(c) No U-turn

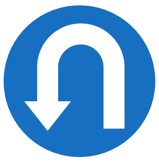
Figure 3.6: Turning bans



(a) Mandatory Left Turn



(b) Mandatory Straight or Right Turn



(c) Mandatory U-turn

Figure 3.7: Mandatory driving directions

One-way streets, No Entry and driving bans. One-way streets allow traffic in only one direction. After every junction, the according direction must be indicated by the sign shown in Figure 3.5a. The other direction needs to be banned for traffic, usually by a No Entry sign (Figure 3.5b) at the other end of the arm.

A round, white traffic sign with a red border, called No Vehicles, indicates a driving ban in both directions. This sign is often restricted to certain vehicles, properties of vehicles (like height or weight), or combinations thereof, by an according symbol within the main sign. Figure 3.5c shows the example of a driving ban for bicycles.

Note that highways are implicit one-way streets and must not be announced as such. Further, they also imply a driving ban for bicycles and other means of transport as mentioned earlier. Thus, we can view traffic signs as *groups* of restrictions, which in general could also be expressed by other signs. For instance, having a No Entry sign at all entry points to a street essentially yields the same restriction as posting a general driving ban. Thus, we distinguish between the traffic sign and its *effects*, which depend on the context, i.e., the street topology and the other signs.

Category	Examples
Vehicle	passenger car, truck, bicycle, motor vehicles
Attribute	gross vehicle weight, height, length, trailer
Owner	doctor on duty, supplier
Role	taxi, public service, emergency vehicle, school bus
Activity	access, loading, boarding
Time	Mondays, workdays, 9:00 a.m. – 5:00 p.m.
Weather	rain, snow, slippery road

Table 3.1: Categories of validity restrictions

Turning bans and mandatory driving directions. In a similar way as the former signs prohibit traffic for entire streets, turning bans (Figure 3.6) restrict the allowed driving directions at junctions. Their validity can also be restricted for specific road users by an according additional panel, e.g., "except bicycles" in addition to a No Entry sign. Dually, mandatory driving directions, as shown in Figure 3.7, indicate which directions are allowed for traffic.

Combinations with the aforementioned traffic signs can lead to logical inconsistencies. Imagine, for instance, the Mandatory Left Turn (Figure 3.7a) pointing to a street where a No Entry sign is posted. Furthermore, in areas with many one-way streets, one could easily create a loop, or at least complicated routes, by posting additional restrictions on driving directions. Moreover, one might be interested in higher-level goals associated with the flow of traffic, e.g., noise reduction in populated areas by means of permanent diversions for trucks. We aim to target such common sense and optimization issues, which will not be defined in legal texts, as well.

Validity restrictions. The validity of most regulations can be restricted to certain subsets of road users and conditions by adding an additional panel below the traffic sign. For common validity restrictions there exist designated traffic signs which symbolize the validity in the main sign, like noted before (Figure 3.5c). However, such traffic signs might also be restricted further by additional panels. Table 3.1 lists categories from which such restrictions are composed. Note that the applicability of these categories depend on the kind of traffic restriction. Speed limits, for instance, will be bound to vehicle types and attributes, or weather conditions, but not vehicle owners, roles or activities. Parking bans, on the other hand, are usually bound to weekdays and time intervals but will not refer to weather conditions.

The investigation of highways and motorways revealed that dealing with validity restrictions introduces a considerable source of additional complexity. In fact, even a formal model to represent which aspects of Table 3.1 may or may not be combined, or in which ways, is a challenging task on its own. To develop a systematic approach towards advanced inconsistency management tasks, we will for the purpose of this work assume a single modality of traffic. Validity restrictions are beyond the scope of this thesis and remain to be done in future work.

As a consequence, we do not model traffic regulations that require validity restrictions, such as “No Entry except bicycles,” or different speed limits for different road users on the same position, as implied e.g. by highways.

3.3 Inconsistencies

In the previous section we already pointed out potential inconsistencies which might arise due to bad traffic sign posting, respectively traffic measure combinations. We will now categorize different classes of errors.

Data faults. The simplest conflict is when a single traffic sign or traffic measure is faulty. First, the data assigned to the street map can contain errors. This occurs when a traffic sign or traffic measure has a bad position or a wrong extent. For instance, a Give Way sign must be posted directly before a junction and the shape of a residential area cannot be a point. Second, the relation between the input and the street graph can be flawed. As an example, imagine a No Left Turn before a junction that has no outgoing arm on the left-hand side.

Missing required restrictions. We mentioned above that pedestrian zones cannot occur outside residential areas. Similarly, a priority road requires that all incoming traffic is informed about the right of way. Consequently, before a junction where a priority road is crossed, there must be posted a Give Way sign or a Stop sign. Also, we cannot announce a one-way street in one direction without banning the opposing direction for traffic.

Overlap of mutually exclusive types. Certain kinds of restrictions or measure types (respectively sign types) cannot occur on the same position. For instance, in Austria it is not allowed that a highway leads through a residential area. Technically, a similar case is the overlap of a parking ban and a halting ban. Likewise, there cannot be a parking ban on a highway. Furthermore, on every lane, the maximum allowed speed must be unique. This may not be the case in traffic regulation databases, since a street might be wrongly associated with two speed limit measures of different values.

In Example 2 we have seen a similar case where the speed limit after a junction is ambiguous due to a missing repeated start sign. However, an explicit overlap of contradicting speed limits can only be expressed by traffic signs if two start signs for different speed limits are posted at the same location. Otherwise, the latter sign in driving direction always overrules the former.

Contradictory driving permission. A special case of mutually exclusive information is the logical contradiction between driving bans and driving permissions, particularly interesting at junctions. The traffic signs shown in Figures 3.6 and 3.7 give an instruction which turning directions are prohibited, respectively available. Each arm which is pointed to by a mandatory driving direction sign to must also be available for traffic, i.e., not prohibited by other signs, like No Entry. Besides such pairwise *clashes*, there is the more subtle case where all turning directions are banned due to multiple traffic signs. For instance, a No Left Turn sign also prohibits U-turns. Then, if the outgoing arms (other than the left one) are also banned for traffic by other signs, there is a (literal) dead-end.

Correspondence. We noted earlier that traffic signs materialize the intended restrictions of traffic measures. Thus, we can view traffic measures and traffic signs as two languages which shall describe the same restrictions, i.e., the same meaning. Consequently, an application in which both languages are administrated should provide assistance to ensure their *correspondence*. That is, whenever there is a traffic measure, there must be traffic signs expressing the same restrictions, and vice versa. Traffic measures for which no according signs are posted are not legally binding. We call such measures *unannounced*. Dually, there might be posted traffic signs which do not (or no longer) have a legal backing by according traffic measures. These *unjustified* traffic signs should also be detected.

Common sense and flexible criteria. Apart from strictly illegal cases, we have already discussed some common sense issues, like preventing loops. In practice, we also have many soft criteria, like aiming for low speed limits and driving bans for heavy trucks in streets with kindergartens.

As another example, imagine a traffic planner who wants to evaluate different traffic flow related properties or examine the implications of hypothetical new restrictions on road use. To avoid complicated routes, for instance, she could test before every junction the number of turns required to reach the street on the left-hand side. Without restrictions, this number is 1, but in urban areas with many one-way streets it might be 3 (right turns). However, any larger number is suspicious.

3.4 Use Cases

We introduced the meaning of prominent traffic signs and illustrated which problems might arise from their combined application on streets. We now identify different use cases around such inconsistencies from a user perspective.

3.4.1 Consistency Evaluation

Given an application for the administration of traffic measures and traffic signs, the first question is whether the database complies with a fixed specification like the Road Traffic Regulations. In case of conflicts, one wants to know which kinds of problems occur, and where.

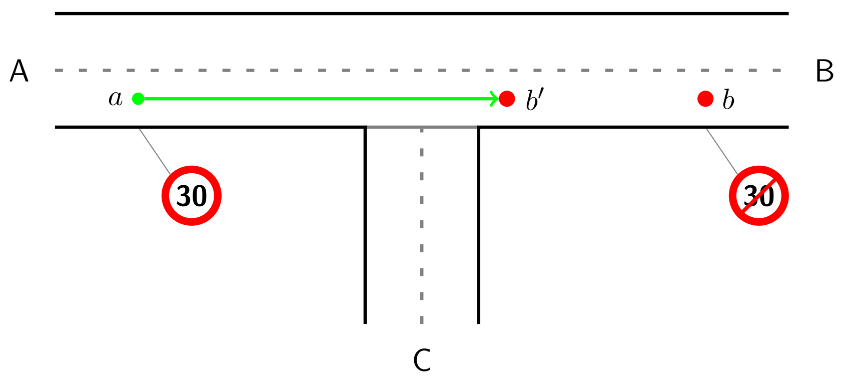


Figure 3.8: Insufficiently announced 30 km/h speed limit yields two conflicts

Example 10 Figure 3.8 shows the start sign and the end sign of a maximum speed limit of 30 km/h like in the examples of the previous chapter. However, the repeated start sign after the junction is missing. This leads to two conflicts as indicated by the red dots at points b' and b. As explained in Example 1 (page 2), traffic from arm C, turning right into arm B, is not informed about a speed limit. Since a speeding ticket could be challenged with this argument, the legally effective speed limit ends after the junction. However, speed limits cannot end without any explicit or implicit end sign. Thus, we have a conflict at point b'. Second, we have a conflict at point b, where no open restrictions exists to be ended by the posted end sign. ■

In this case, an expert will immediately recognize the problem. In real-world scenarios, however, which comprise a large number of traffic signs, it is not always apparent which traffic signs cause which conflicts. Hence, a diagnostic mechanism needs to be provided, relating inconsistencies with their causes.

3.4.2 Diagnosis

Given inconsistent traffic regulations, a government official shall be assisted in finding the causes for detected problems. What we view as explanations for according conflicts is inspired by the literature in the field of knowledge-based systems. In Chapter 2 we distinguished between consistency-based diagnosis and abductive diagnosis. We noted that abductive diagnosis is the stronger notion, in which we require that observable findings are a logical consequence of the diagnosis.

Moreover, we argued that abductive diagnosis fits domain knowledge with explicit fault models, or models where causes and effects are directly related. This meets our domain, where we aim to explicitly specify the conditions of illegal or undesirable situations. What we understand as a diagnosis (or explanation) for a set of conflicts is a set of traffic signs or measures causing it.

Example 11 (cont'd) In Figure 3.8, we have two conflicts. The problem at point b', that a restriction ends with an end sign, is caused by the start sign at point a. By removing only the start sign, this conflict is resolved. Independent of that, the other conflict at point b can be explained by the end sign alone. ∎

Diagnosis is useful as a preparatory step for a manual repair. However, we want to be able to provide a mechanism for generating repair proposals. In particular, diagnosis can only help to find data to be removed. As the example suggests, additions of new traffic signs shall also be considered.

3.4.3 Repair

Given an inconsistent traffic regulation scenario, it would be convenient to have an automatic repair mechanism, or to get a list of repair proposals to choose from. By a repair we mean (a preferably small number of) changes to the data base such that the result is consistent with the specification, i.e., free of conflicts.

Example 12 (cont'd) In Figure 1.2 (page 2) we have seen the correct sign posting for the 30 km/h speed limit from point a to point b. However, by banning the right turn from arm C to arm B with a No Right Turn sign, as shown previously in Figure 1.3, the scenario is also repaired. This alternative repair is also minimal in the sense that only a single traffic sign needs to be added to establish compliance with the Road Traffic Regulations. ∎

It is not intuitive to repair an inconsistency arising from speed limit signs by imposing a traffic ban. In order to avoid such unnatural repairs, we must also take into account the legal intentions behind traffic sign posting, which are captured by traffic measures. This brings us to our fourth major use case.

3.4.4 Correspondence and Strict Repair

Since we deal with both traffic measures and traffic signs, we must ensure that inputs in these two description languages *correspond*, i.e., that they express the same traffic restrictions. We get a stronger notion of repair, called *strict repair*, by requiring that traffic sign and traffic measure data are not only free of conflicts, but also correspond.

Example 13 (cont'd) Introducing a No Right Turn sign on arm C, like in Figure 1.3, is not a strict repair for the conflicts shown in Figure 3.8. If a speed limit measure (from a to b) is stored in the database, only the addition of the repeated start sign after the junction is a strict repair (Figure 1.2). The introduction of the No Right Turn sign on arm C would also require the addition of a corresponding measure. However, a suggestion service for repairs should not list such unintuitive, arbitrary modifications. ∎

We have investigated traffic regulations, inconsistencies and use cases how to deal with them. Next, we informally introduce the technical approach towards their realization.

3.5 Technical Approach

To tackle our logic-oriented problems in the presented traffic regulation domain, we first need a street model based on which we can express measures and signs. Generally, the data model should focus on the inconsistency management tasks and not be bound to specific software solutions or street maps. Therefore, we will view streets in a very general way as *directed graphs*, where edges represent the potential direction of traffic. Each edge will get a unique label to discern whether it represents a part of a lane, a turn over a junction or a U-turn.

To reflect measures and signs of a given scenario, we add further labels to the graph. For measures we use its type to label every edge it is associated with. Likewise, sign types are used to label nodes where according traffic signs are posted. During this domain analysis, we observed that measures and signs ought to represent the same restrictions. Hence, we introduce a third language of *effects*, which we likewise represented as edge labels. For instance, a Mandatory Left Turn sign has the effect that the outgoing edge with label *left* must be necessarily available for traffic, and all other outgoing edges from that node, including the U-turn, are prohibited for traffic. By labelling the left edge with *nec*, and the other ones with *ban*, we can encode the local meaning of the sign.

After an *effect mapping*, which creates such effect labels from measures and signs, we derive in a second mapping step the *conflict labels* (on nodes), which represent inconsistencies relative to a fixed conflict specification. The conflict labels may then be used to visualize inconsistencies on the respective street map, followed by user interaction related to diagnosis and repair.

3.5.1 Challenges

Given an inconsistent traffic regulation scenario, the computation of reasonably filtered and ranked repair proposals can be very difficult due to the evolved and context-sensitive nature of traffic regulations. An intelligent decision often depends on special conditions or needs common sense reasoning. Generic rules may likely fail to produce intuitive results. According to our experience, the development of suitable abstractions towards reliable and robust rules often requires many examples. Sometimes, the investigation of a new situation puts previous rules into question.

Example 14 Consider the traffic signs dealing with driving permissions and driving bans, such as mandatory driving directions, prohibited turns like No Right Turn, one-way street, No Entry, and so on. An effect mapping can intuitively label edges with a label *ban*, in case traffic is prohibited there, or *nec*, if the edge must be necessarily available for traffic due to a mandatory driving direction. A conflict specification will specify that no edge can be labelled both with *ban* and *nec*.

Now, consider that we wish to model home zones. In Austria it is not prohibited to drive within home zones, but to drive *through* them. Consider the case that we have a Mandatory Left Turn sign pointing towards a home zone, which shall be considered as conflict, for our consideration. If we use the *ban* label to reflect the special prohibition implied by the home zone, we can detect the conflict by the edge which gets labelled both with *ban* (for the implied restriction of the home zone) and with *nec* (for the mandatory turn). However, we may as well have mandatory driving direction signs within a home zone, for which the same conflict would then wrongly be reported. ∎

The example showed that there is a subtle but relevant difference between two kinds of driving bans, which have to be accounted for by the model. That is, we cannot use the same labels to model the effects of the driving restriction implied by home zones, as we would for turning restrictions and the like. However, naively using new effect types may not always work. On the other hand, working backwards from potential conflicts of interest turned out to be more fruitful.

Example 15 (cont'd) A closer look at the home zone example reveals that, without a clear use case, no reasonable choice can be made on how to deal with its special nature. In order to model the intended prohibition of the home zone, we need an unambiguous specification of what it means to *drive through* a certain zone. Is entering a zone and leaving it at the same junction also driving "through" it? Is leaving a home zone at a different street allowed if the car was parked intermediately? That is, do we need to model stops, i.e., time?

We see that there are different kinds of reachability issues and problems we may be interested in. A standard routing application should not propose a sequence of edges involving home zones, unless the start or the end node of such a query is within the zone. For this use case, the labelling approach above with *ban* and *nec* would work, since the question whether traffic is allowed along a given sequence of edges reduces to asking whether none of these edges is assigned a *ban* label.

For our inconsistency tasks, however, we may deal with the special nature of home zones with designated rules. That is, we can detect clashes, as exemplified above, with special tests for home zones, leaving the other generic rules untouched. ∎

To clarify such considerations, an implementation language which allows for rapid prototyping is useful. We can expect new traffic signs to be included over time and, more importantly, new consistency criteria to be tested for. These changes in the specifications should be possible in a flexible, modular way.

In Section 1.1, we pointed out different usage scenarios of an inconsistency management tool in the field of traffic regulations. Moreover, we listed different categories of inconsistencies in Section 3.3. This suggests that we will want to maintain *different* effect mappings, and *different* conflict specifications sharing a common, simple data model, which may then be applied flexibly based on the use case. Further, such conflict specifications should ideally be adaptable at runtime by domain experts.

Example 16 Imagine a government official observes a street with 30 km/h and 70 km/h speed limits in opposing directions. While this is not illegal, and probably not tested for by a standard implementation, she might wish to evaluate whether such a situation also occurs elsewhere. If the system can be extended, an expert might then add according rules to the application which are considered in forthcoming evaluations. In this case, it would be the check whether the difference in speed limits of parallel lanes exceeds a certain constant. ■

The implementation should allow for an intuitive encoding of the domain knowledge. This suggests using a declarative language, and in particular, a rule-based, logic-oriented language. This will give a readable code by allowing to directly express the logic itself, rather than control flow and instructions to compute it. Further, direct support for non-monotonic reasoning, especially default reasoning [43], is desirable for a natural representation of default cases that hold unless specific conditions apply. For instance, a speed limit of 50 km/h is implied by a residential area unless it is explicitly overruled. Similarly, traffic is generally permitted along a lane or turning direction unless it is explicitly prohibited.

These observations suggest that Answer Set Programming is a suitable choice for an implementation of our inconsistency management tasks.

3.5.2 Answer Set Programming

We now review the observed desiderata for an implementation language and argue how Answer Set Programming meets the requirements.

Automated reasoning. Our inconsistency management tasks deal with the logical implications of traffic signs and traffic measures on a digital street map. Hence, a logic-oriented language is a natural choice in the first place. That is, instead of writing algorithms to compute the logic, we want to encode rules as such, and let a solver interpret the meaning of input data according to these rules.

Prolog, being the most prominent logical programming language, however, has a significant drawback in this regard since it has no fully declarative semantics. This implies limitations on both modularity and readability, two major dimensions of maintainable software, which we discuss in more detail below.

Answer Set Programming, on the other hand, is purely declarative and allows for model building, instead of proof search. In particular, the stable model semantics [23] fits our problem domain well, which we will see in detail in forthcoming chapters. For instance, to compute diagnoses or repairs for conflicts, we will declare simple conditions and constraints in addition to the specification and immediately get the solutions as answer sets of the extended program. ASP is a powerful paradigm allowing to solve search and constraint satisfaction problems without writing algorithms, but only by specifying the properties of their solutions. For knowledge representation purposes, ASP has clear advantages over SAT solving in terms of expressivity; allowing for predicates, non-monotonic reasoning, defaults, and transitive closures.

Modularity. We have seen the importance of a highly flexible (i.e. modular) specification based on which consistency of traffic regulations is defined. The effect definitions of new measure and traffic sign types should, were possible in principle, not affect the rest of the implementation. Moreover, the conflict specification is supposed to evolve over time. Consequently, the introduction of new traffic signs, measures, or conflict definitions should in most cases only require to add new rules. There should be no need to change the system at large.

Such modularity can only be achieved by careful design, which will be the topic of the next chapter. However, modularity of software is supported to different degrees by the language being used. The semantics of Answer Set Programming assists modular composition, since programs are *sets* of rules. That is, if we have an existing program P, and the meaning of a new traffic sign is modelled in a new, independent set of rules Q, we get the extended program by taking the union $P \cup Q$. Likewise, new consistency criteria can be added and removed, provided a robust formal model for conceptual clarity.

Readability. We argued that the implementation should be *declarative* since a typical imperative way of programming rules will quickly lead to deeply nested conditionals with intransparent dependencies between various variables.

Further, imagine an object-oriented approach towards this traffic regulation logic, where rules and conditions would be modelled as objects of different classes. Any test will be composed of multiple such objects involving complicated references and considerable amounts of imperative boilerplate code. The central logical pattern “if X and not Y, then Z” will not be naturally reflected by the syntax of such programs.

This might be provided by functional languages and languages with macro features, in which domain specific languages can be created towards a syntax that fits the problem domain. However, by following this strategy, the computation of the logic must still be programmed manually.

Answer Set Programming, on the other hand, directly provides a natural syntax for expressing logic-oriented programs. With readily available solvers like DLV [30] and the Potassco suite [21], we get executable implementations by purely declarative specifications.

CHAPTER 4

Formal Model

In the previous chapter we analyzed the traffic regulation domain and discussed consistency related questions which arise in the context of traffic measures and traffic signs. In order to reason over such traffic regulation problems, we first need an underlying formal model, which we will describe next. First, we present a tailored street model based on which we can reflect measures and signs. We then show how the meaning of this traffic regulation data is captured by means of *effects* and discuss how they are obtained. Towards our central interest of inconsistency, we will introduce the notion of *conflicts*, which are explicit representations of undesired situations as defined by the considered traffic regulation.

4.1 Street Graph

In this section we formalize the data model representing streets. We model street maps as directed graphs, where edges represent potential flow of traffic. To reflect topological information, e.g., whether some edge is part of a lane or a turn over a junction, edges get according labels. Certain restrictions on the structure of these graphs and their labels then yield our street model, as we describe in the following.

Definition 4 (Street graph) *A* street graph*, or* graph *for short, is a connected, labelled, directed graph $G = (V, E, \ell)$ of nodes V, edges $E \subseteq V \times V$, and labelling function ℓ that assigns each edge $(v, w) \in E$ a unique label $\ell(v, w) \in \{left, right, straight, lane, uturn\}$, called the* street labels.

Towards our logic-oriented use cases and forthcoming definitions, we identify the street labels with a set T of constants and a street graph $G = (V, E, \ell)$ with the set a atoms of form $e(t, v, w)$. Every edge $(v, w) \in E$ is associated with such an atom, where $t \in \mathsf{T}$ is the street label $\ell(v, w)$. Throughout, we assume that a version of predicate logic $\mathcal{L}$ with negation is fixed, in which the desired specifications can be expressed (e.g., first-order logic or Answer Set Programming).

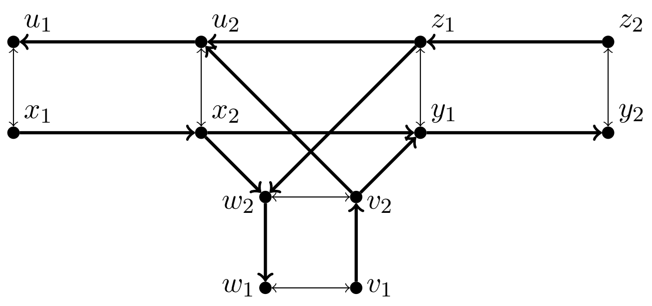


Figure 4.1: Street graph of a T-junction

A street graph has to satisfy certain structural requirements as described next.

Scope. All nodes must have an incoming edge and an outgoing edge with label *uturn*. A node which has only one outgoing edge, thus necessarily labelled *uturn*, is called *out-node*. Similarly, a node with only one incoming edge is called *in-node*. These two kinds of nodes model the end of the scope and are thus called *border nodes*, which form the starting points and end points for reachability considerations. In the T-junction of Figure 4.1, the edges (x_i, u_i), (y_i, z_i), (w_i, v_i) (with $i \in \{1, 2\}$, resp.) and their symmetric counterparts are U-turns, modelled with atoms of form $e(\mathit{uturn}, X, Y)$. The nodes u_1, w_1 and y_2 are out-nodes, and x_1, v_1 and z_2 are in-nodes.

The following restrictions on different kinds of nodes partially do not apply in case they are border nodes.

Junctions. We model junctions by means of edges with labels *left*, *right* and *straight*, referred to as *junction labels*, the edges are referred to as *junction edges*. Additional turning directions like "half left" could be introduced analogously. In Figure 4.1, these edges are given by $e(\mathit{right}, x_2, w_2)$, $e(\mathit{straight}, x_2, y_1)$, $e(\mathit{left}, v_2, u_2)$, $e(\mathit{right}, v_2, y_1)$, $e(\mathit{left}, z_1, w_2)$, and $e(\mathit{straight}, z_1, u_2)$.

Lanes. Each node is either a *lane start*, a *lane end*, or *within a lane*. A node is said to be a *lane start*, or a *node after a junction*, if there exists an incoming junction edge. Such a node cannot have an incoming edge labelled *lane* and must have exactly two outgoing edges, one labelled *uturn* and one labelled *lane*. In the example, u_2, w_2 and y_1 are lane starts.

A node is said to be a *lane end*, or a *node before a junction*, if it has at least one outgoing edge with a junction label. Such a node must have exactly one incoming edge with label *lane*, must not have any incoming edge with a junction label and must not have an outgoing edge labelled *lane*. Nodes x_2, v_2 and z_1 are lane ends.

A node is said to be *within a lane*, if there exists both an incoming and an outgoing edge with label *lane*, which must be unique, respectively. Such a node can have neither an incoming nor an outgoing edge with a junction label. In general, nodes within a lane are not modelled. We remain agnostic about the nature of border nodes in this regard. For instance, we do not care whether x_1 and u_1 are nodes after (resp. before) a junction, or whether they are nodes within a lane.

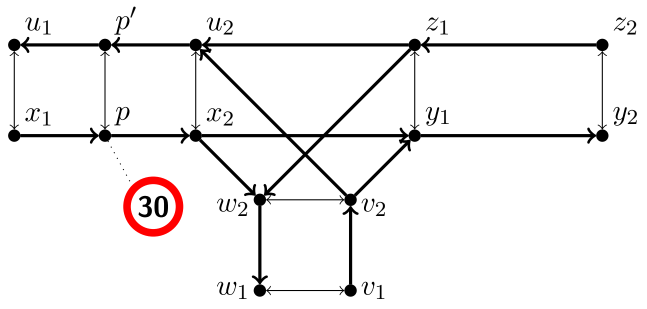


Figure 4.2: Street graph of a T-junction with split lanes in the left arm

Symmetry. We say two nodes are *parallel*, if they are connected by a U-turn. For each pair of parallel nodes, we require a U-turn in both directions. We say two edges are *parallel* if the respective pairs of start and end node are parallel. For instance, (x_1, x_2) and (u_2, u_1) are parallel edges, since x_1 and u_1 are parallel nodes, as well as x_2 and u_2. We require the graph to be set up in such a way that parallel edges represent street parts of equal lengths. The lengths themselves do not play a role in our application and are therefore not modelled. Similarly, we are not interested in the angle between crossing streets, the width of lanes, or any other metrics.

If some point of interest p needs to be modelled within a lane, say, between x_1 and x_2, the lane must be given in (at least) two parts, i.e., edges (x_1, p) and (p, x_2) with label *lane*. Then, also the opposite direction must consist of two *lane* edges with respective parallel points with according U-turns. Figure 4.2 shows such split lanes with new parallel nodes p and p' in order to allow the modelling of a traffic sign along the lane (x_1, x_2).

Unless we need to model traffic signs, or starts or ends of measures somewhere between junctions, i.e., not directly before or after junctions, we need only one edge with label *lane* per direction to connect two junctions. U-turns can only connect parallel edges both of type *lane* or *straight*, i.e., edges modelling opposite directions of non-turning traffic. Every street has to be modelled with edges for both directions. In case only one direction is allowed for traffic, the other one has to be explicitly prohibited for traffic by means of traffic signs and measures.

Our data model is designed for inconsistency management and is less concerned with accuracy of street modelling. Aiming for simplicity, we do not deal with more complex streets involving multiple lanes per driving direction, or roundabouts. However, our definitions may be adapted to suit more sophisticated graphs, if needed.

4.2 Traffic Measures and Traffic Signs

In the formulation of measures in traffic regulation orders, concepts like street names, addresses and cardinal points are used in natural language descriptions to specify the intended topological dimensions. We assume that for the description at hand, a prepro-

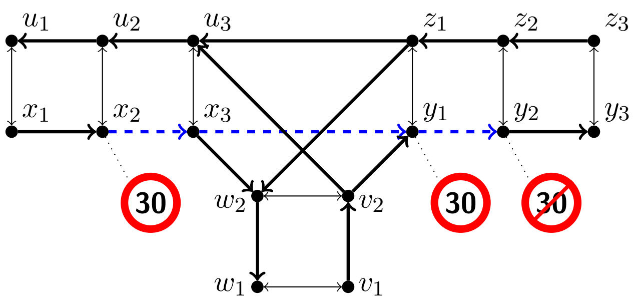


Figure 4.3: Correct traffic sign posting for a 30 km/h speed limit measure

cessing maps this scope to edges of a graph meeting the requirements above.

To describe traffic measures and traffic signs we build upon sets of ground terms M, called the *measure types*, and S, called the *sign types*. These types are used to label the edges, resp. nodes, on which measures, resp. signs, occur. For instance, M may contain a set of terms $\mathit{spl}(k)$ for each speed limit value k that is needed, e.g., $\mathit{spl}(10)$ up to $\mathit{spl}(130)$ in Austria. The set S includes constants like $\mathit{no\text{-}entry}$, and the terms $\mathit{start}(t)$ and $\mathit{end}(t)$ for all measure types $t \in \mathsf{M}$ that have explicit start and end points, i.e., types that have a topological stretch in form of lines or zones.

Definition 5 (Measures, Signs, Input) *Given a street graph $G = (V, E, \ell)$, and sets of ground terms M (measure types) and S (sign types), we define the following sets of atoms:*

- Measures $M_G = \{m(t, v, w) \mid t \in \mathsf{M}, (v, w) \in E\}$
- Signs $S_G = \{s(t, v) \mid t \in \mathsf{S}, v \in V\}$
- Input $I_G = M_G \cup S_G$

It is thus our aim to associate traffic measures with edges and traffic signs with nodes. That is, to reflect a measure of type $t \in \mathsf{M}$, all respective edges (v, w) must be labelled with t. To distinguish between the original measure information and its representation, we also call the atoms in M_G *atomic measures*. Likewise, to represent a traffic sign of type $t \in \mathsf{S}$ posted at a node v, the latter must be labelled with t. Similar to street labels, these labels are encoded as ground atoms of form $m(t, v, w)$ and $s(t, v)$, respectively.

Example 17 In Figure 4.3, the dashed blue path from x_2 to y_2 symbolizes a 30 km/h speed limit measure which we formalize as a set of the following atomic measures:

$$M = \{m(\mathit{spl}(30), x_2, x_3), m(\mathit{spl}(30), x_3, y_1), m(\mathit{spl}(30), y_1, y_2)\} \subseteq M_G$$

The traffic signs are defined similarly:

$$S = \{s(\mathit{start}(\mathit{spl}(30)), x_2), s(\mathit{start}(\mathit{spl}(30)), y_1), s(\mathit{end}(\mathit{spl}(30)), y_2)\} \subseteq S_G \qquad ■$$

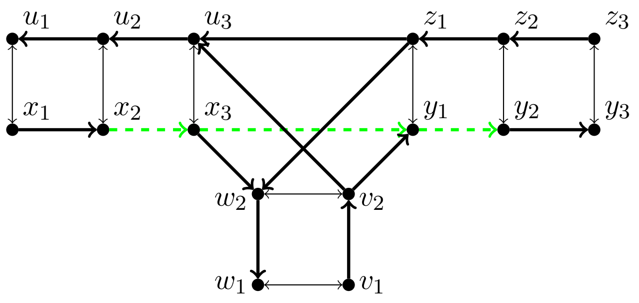


Figure 4.4: Intended *max-speed*(30) effect of measures and signs in Figure 4.3

Sets of measures and signs on a street graph form the syntax for our investigations.

Definition 6 (Scenario) *Let G be a street graph, $M \subseteq M_G$ be a set of measures on G, and $S \subseteq S_G$ be a set of signs on G. Then, a* scenario *is a tuple $Sc = (G, M, S)$.*

We note that such node and edge labels may be similarly used to model potentially further input languages. In particular, we do not explicitly cover road markings, which can informally be seen as traffic signs but may be represented along edges like measures. Finally, we note that any data source, from which a scenario can be generated, can serve as basis for inconsistency management as described in the sequel.

4.3 Effects and Conflicts

The meaning of both measures and signs on a given street graph is captured by a mapping of the according languages to a common target language of effects, which will in turn be used to define conflicts. We use the constants ℓ_m and ℓ_s to represent the *measure language* and the *sign language*, respectively. Similar to Definition 5, we introduce another two sets of labels, ground terms F and C, called the *effect types* and *conflict types*, respectively.

Definition 7 (Effects, Conflicts) *Let $G = (V, E, \ell)$ be a street graph and F (effect types) and C (conflict types) be sets of ground terms. We define the following sets of atoms:*

- Effects $F_G = \{f(t, v, w) \mid t \in \mathsf{F}, (v, w) \in E\}$
- Conflicts $C_G = \{c(t, v) \mid t \in \mathsf{C}, v \in V\}$

Atoms not covered by Definition 4, 5 and 7, i.e., those with a predicate symbol not included in $\{e, m, s, f, c\}$, are called *helper atoms* or *helper predicates.*

In their form, effects and conflicts resemble measures and signs. Before we explain how they are obtained, we consider a few examples.

Example 18 (cont'd) In Example 17 we considered a 30 km/h speed limit measure with an according traffic sign posting. Both kinds of input express the same road use restrictions, i.e., the same *effects*, which are depicted in Figure 4.4. To express the meaning of both measure and sign data, the respective edges must be assigned the effect type $\mathit{max\text{-}speed}(30) \in \mathsf{F}$. This is represented by atoms $f(\mathit{max\text{-}speed}(30), V, W)$, where $(V, W) \in \{(x_2, x_3), (x_3, y_1), (y_1, y_2)\}$.

We will deal with different kinds of conflicts, which we do not need to distinguish formally. Some conflicts will arise from measures only, some can be caused only by traffic signs, and some may be caused by both languages.

For the latter case, we use the respective constant ℓ_m or ℓ_s, to reflect where the problem originates. For instance, to represent potential (prohibited) overlaps of motorways and residential areas, we might include $\mathit{overlap}(L, \mathit{motorway}, \mathit{residential\text{-}area})$ in C for $L \in \{\ell_m, \ell_s\}$. To detect that this conflict occurs on a node v due to traffic sign data, the atom $c(\mathit{overlap}(\ell_s, \mathit{motorway}, \mathit{residential\text{-}area}), v) \in C_G$ will then be used. Similarly, $c(\mathit{no\text{-}way\text{-}out}(\ell_m), v)$ expresses the fact that one is caught in a dead end or loop at node v due to measure information. ■

The following two definitions will assist establishing a two-stage mapping approach.

Definition 8 (Closed world operator) *Let Y be a set of ground atoms and $X \subseteq Y$. We define*

$$\overline{X}_Y = X \cup \{\neg x \mid x \in Y \setminus X\}$$

as the closed world operator *applied to X relative to the* base set *Y.*

We always use the according base set of Definitions 5 and 7, and thus omit the subscript. For instance, for a set measures M on G, $\overline{M}$ abbreviates $\overline{M}_{M_G}$. The base set assumed for the completion $\overline{G}$ of any graph G is the set of all atoms $e(t, v, w)$. We introduce another operator Cn_G that maps between atoms on a graph G. In the first step, it will be used to associate measure and sign atoms with effect atoms. Then, those effect atoms will be related with conflict atoms.

Definition 9 (Cn_G) *Let X and Y be sets of atoms on G and let T be a set of formulas in a fixed predicate logic $\mathcal{L}$. The Y*-consequences of T and X (on G) *is the set of atoms*

$$Cn_G(T, X, Y) = \{y \in Y \mid T \cup \overline{G} \cup \overline{X} \models y\}.$$

Here, $\models$ is the consequence relation in the underlying logic $\mathcal{L}$. Informally, the Y-consequences of T and X are the set of atoms in Y entailed by a description or theory T, given a graph G and some additional facts X. The closed world operator makes sure that atoms that are not explicitly given are set to false and thus ensure that valuations of atoms in Y are unique. When using answer set programming, which employs closed world reasoning, we will not need this operator. However, we will leave open the exact logic used, so this restriction will enable a general two-stage approach computing graph labels, as described next.

4.4 Effect Mapping

Now we introduce the mechanism by which the meaning of both traffic measures and traffic signs are computed.

Definition 10 (Effect mapping) *An* effect mapping *relative to a street graph G is a set P of formulas in $\mathcal{L}$ that associates with each input $I \subseteq I_G$ the set*

$$\mathcal{F}_G^P(I) = Cn_G(P, I, F_G)$$

of atoms, called effects of I (on G)*.*

That is, given a scenario $Sc = (G, M, S)$ and $I = M \cup S$, an effect mapping P determines a set $\mathcal{F}_G^P(I) \subseteq F_G$ of effects that captures the meaning of I. We implicitly assume that effect mappings are well-designed, i.e., they do not add new graph elements or new input and that the ranges of terms are used appropriately.

When using answer set programs, $\models$ will denote cautious consequence, unless stated otherwise. That is to say, $P \models q$ holds if q is true in *all* answer sets of P, i.e., $q \in \bigcap \mathcal{AS}(P)$. The notation $\mathcal{AS}_Q^{\cap}(P)$ abbreviates $\bigcap \mathcal{AS}(P) \cap Q$, i.e., the set $\{q \in Q \mid P \models q\}$. Consequently, when using answer set programs, $Cn_G(T, X, Y) = \mathcal{AS}_Y^{\cap}(T \cup G \cup X)$.

Example 19 The first-order sentence

$$\forall k, x, y\, (m(spl(k), x, y) \quad \supset \quad f(max\text{-}speed(k), x, y)) \tag{4.1}$$

of an effect mapping captures the meaning of (atomic) speed limit (*spl*) measures. Similarly, we can model the effect of a speed limit with ASP in a single rule:

$$f(max\text{-}speed(K), X, Y) \quad \leftarrow \quad m(spl(K), X, Y) \tag{4.2}$$

Note that in this case, the ASP rule closely resembles the first-order formula. All variables, written in upper case letters, are implicitly universally quantified.

To represent that some effect type F is the consequence of some measure type T on a given edge, we can take a more generic approach, using helper predicates with symbol *m2f*. Second, in order to keep track which effects stem from the language of measures (ℓ_m), and which from the language of signs (ℓ_s), we will employ a helper predicate symbol f'. We use the following ASP rules plus a set of helper atoms $speed(K)$ for $K \in \{5, 10, 20, \ldots, 130\}$ as effect mapping P.

$$\begin{aligned} f(F, X, Y) &\leftarrow f'(L, F, X, Y). && (4.3)\\ f'(\ell_m, F, X, Y) &\leftarrow m(T, X, Y), m2f(T, F). && (4.4)\\ m2f(spl(K), max\text{-}speed(K)) &\leftarrow speed(K). && (4.5) \end{aligned}$$

The first rule says that every *measure effect* ($L = \ell_m$) and every *sign effect* ($L = \ell_s$) is an effect. The second rule directly translates from measure labels to effect labels on the same edge, as defined by *m2f*. The third rule specifies that a speed limit measure of value K translates to a maximum speed restriction of value K. ■

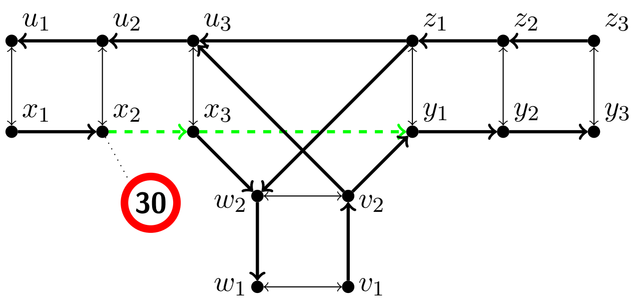


Figure 4.5: Effect of the 30 km/h restriction start sign at x_2

We will now examine how a speed limit is determined by means of traffic signs.

Example 20 Whenever a road user passes a start sign of a speed limit, the according effect holds as long as she does not reach an end sign. The start of another speed limit implicitly ends the former. In non-zonal speed limits, which we consider here, the effect also automatically ends when the driver leaves the street on which it was announced (by turning left or right into a different one). That is, a linear speed limit effect is only propagated in direction of traffic. In our data model, these directions are captured by the edge labels *lane* and *straight*. We specify some additional information in the effect mapping P:

$$\mathit{in\text{-}dir}(X,Y) \leftarrow e(\mathit{lane},X,Y) \tag{4.6}$$

$$\mathit{in\text{-}dir}(X,Y) \leftarrow e(\mathit{straight},X,Y) \tag{4.7}$$

In Figure 4.5, consider road users coming from node v_2, turning right into the arm starting at y_1. Those drivers would not know about the speed limit expressed at x_2. Therefore, if a node after a junction is reachable from a different arm, there has to be another start sign to inform incoming traffic about the restriction. (U-turns are ignored for practical reasons.) We formalize these observations and extend P, describing how the $\mathit{max\text{-}speed}(K)$ effect labels are obtained by means of traffic signs.

If, at a node X, the start of the effect type F is expressed ($\mathit{expr\text{-}start}(F,X)$) then the effect holds on the next edge in direction of traffic ($\mathit{in\text{-}dir}(X,Y)$).

$$f'(\ell_s,F,X,Y) \leftarrow \mathit{expr\text{-}start}(F,X), \mathit{in\text{-}dir}(X,Y) \tag{4.8}$$

If there are two consecutive edges in direction of traffic ($\mathit{in\text{-}dir}(X,Y), \mathit{in\text{-}dir}(Y,Z)$), where the first is labelled with effect type F, then the effect also holds on the second, *unless* propagation of F is blocked at node Y. We note that such transitive effect propagation cannot be expressed in first-order logic.

$$\begin{aligned} f'(\ell_s,F,Y,Z) \leftarrow\ & f'(\ell_s,F,X,Y), \mathit{in\text{-}dir}(X,Y), \mathit{in\text{-}dir}(Y,Z), \\ & \text{not } \mathit{block\text{-}prop}(F,Y) \end{aligned} \tag{4.9}$$

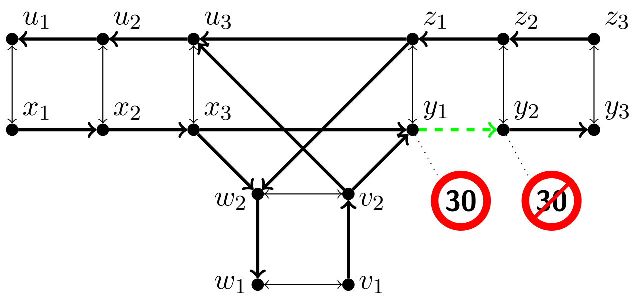


Figure 4.6: 30 km/h restriction from y_1 to y_2

An effect type F is blocked for propagation at node Y, if the end of F is expressed there.

$$\textit{block-prop}(F, Y) \leftarrow f'(\ell_s, F, X, Y), \textit{in-dir}(Y, Z), \textit{expr-end}(F, Y) \tag{4.10}$$

The effect propagation is also blocked at a note Y after a junction ($e(\textit{straight}, X, Y)$), if Y is permitted for incoming traffic according to signs ($\textit{has-perm-inc}(\ell_s, Y)$) but the (repeated) start of the effect is not expressed there (not $\textit{expr-start}(F, Y)$).

$$\begin{aligned}\textit{block-prop}(F, Y) \leftarrow\ & f'(\ell_s, F, X, Y), e(\textit{straight}, X, Y),\\ & \textit{has-perm-inc}(\ell_s, Y), \text{not } \textit{expr-start}(F, Y)\end{aligned} \tag{4.11}$$

A node Y is permitted for incoming traffic according to a language L, if there is an edge $e(T, X, Y)$ with label *left* or *right*, for which a traffic ban cannot be derived. We express this by atoms $\textit{lang}(\ell_s)$ and $\textit{lang}(\ell_m)$ plus two rules:

$$\textit{has-perm-inc}(L, Y) \leftarrow \textit{lang}(L), e(\textit{left}, X, Y), \text{not } f'(L, \textit{ban}, X, Y) \tag{4.12}$$

$$\textit{has-perm-inc}(L, Y) \leftarrow \textit{lang}(L), e(\textit{right}, X, Y), \text{not } f'(L, \textit{ban}, X, Y) \tag{4.13}$$

So far we have not said anything about the predicates *expr-start* and *expr-end*. The start (resp. end) of an effect of type F is expressed at node X, if a traffic sign is posted there, whose type T is defined to start (resp. end) the effect type F.

$$\textit{expr-start}(F, X) \leftarrow s(T, X), \textit{start-of}(T, F) \tag{4.14}$$

$$\textit{expr-end}(F, X) \leftarrow s(T, X), \textit{end-of}(T, F) \tag{4.15}$$

The effect type F is started by a start sign of the according measure type T.

$$\textit{start-of}(\textit{start}(T), F) \leftarrow \textit{m2f}(T, F) \tag{4.16}$$

In general, the end of an effect can be expressed explicitly or implicitly. We will see the need for this distinction below.

$$\textit{end-of}(T, F) \leftarrow \textit{expl-end-of}(T, F) \tag{4.17}$$

$$\textit{end-of}(T, F) \leftarrow \textit{impl-end-of}(T, F) \tag{4.18}$$

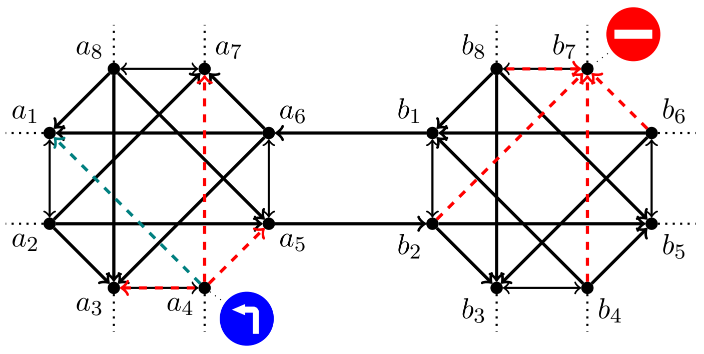


Figure 4.7: Signs that ban traffic along certain edges

An end sign for a measure type T explicitly ends the associated effect F.

$$\mathit{expl\text{-}end\text{-}of}(\mathit{end}(T), F) \quad \leftarrow \quad \mathit{m2f}(T, F) \tag{4.19}$$

The start sign for a speed limit restriction of K km/h implicitly ends the effect of any other speed limit J.

$$\mathit{impl\text{-}end\text{-}of}(\mathit{start}(\mathit{spl}(K)), \mathit{max\text{-}speed}(J)) \quad \leftarrow \quad \mathit{speed}(K), \mathit{speed}(J), K \neq J. \tag{4.20}$$

We use the graph G of the running example as set of edge facts and as input only the start sign of Figure 4.5, i.e., $I = \{s(\mathit{start}(\mathit{spl}(30)), x_2)\}$. With the effect mapping P, the program $P \cup G \cup I$ has a unique answer set, and we get as effects

$$\mathcal{F}^P_G(I) = \mathcal{AS}^{\cap}_{F_G}(P \cup G \cup I) = \{f(\mathit{max\text{-}speed}(30), x_2, x_3), f(\mathit{max\text{-}speed}(30), x_3, y_1\}.$$

Using the second start sign $I' = \{s(\mathit{start}(\mathit{spl}(30)), y_1)\}$, we get an effect labelling for the next two edges: $\mathcal{F}^P_G(I') = \{f(\mathit{max\text{-}speed}(30), y_1, y_2), f(\mathit{max\text{-}speed}(30), y_2, y_3\}$. If we additionally use the end sign at y_2, i.e. $I' \cup \{s(\mathit{end}(\mathit{spl}(30)), y_2)\}$, we get only the effect $f(\mathit{max\text{-}speed}(30), y_1, y_2)$, as shown in Figure 4.6. ∎

Measures and signs that restrict potential driving directions are of special interest. We present an example including mandatory driving directions and No Entry signs.

Example 21 Consider the graph of Figure 4.7, where two junctions are connected by a street with parallel lanes from a_5 to b_2 and b_1 to a_6. There is a Mandatory Left Turn at node a_4 and a No Entry sign at b_7, which we reflect as $s(\mathit{mand\text{-}turn}(\mathit{left}), a_4)$ and $s(\mathit{no\text{-}entry}, b_7)$, respectively. The effect of a Mandatory T Turn at a node X is that the edge in direction T must be necessarily available for traffic (nec) and all other edges are banned (ban).

$$f'(\ell_s, \mathit{nec}, X, Y) \quad \leftarrow \quad s(\mathit{mand\text{-}turn}(T), X), e(T, X, Y) \tag{4.21}$$

$$f'(\ell_s, \mathit{ban}, X, Y) \quad \leftarrow \quad s(\mathit{mand\text{-}turn}(T), X), e(E, X, Y), T \neq E \tag{4.22}$$

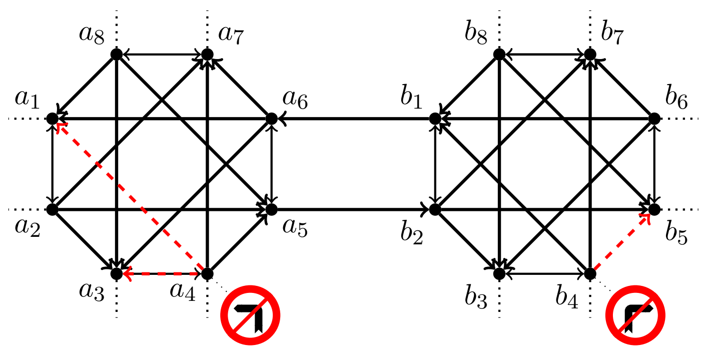


Figure 4.8: The effects of No Left Turn and No Right Turn signs

Recall that in rule (4.9) a general effect propagation was defined. Since we do not want a propagation of these effects, we explicitly prohibit their propagation.

$$\begin{aligned} node(X) &\leftarrow e(T,X,Y) && (4.23)\\ node(Y) &\leftarrow e(T,X,Y) && (4.24)\\ block\text{-}prop(ban,X) &\leftarrow node(X) && (4.25)\\ block\text{-}prop(nec,X) &\leftarrow node(X) && (4.26)\end{aligned}$$

We omit dealing with mandatory turn signs allowing for multiple directions. In rule (4.22) we would need to additionally test whether T is an edge label. In addition, further rules are required for signs of type *mand-turn*(T), where T does not is not an edge label. For instance, *mand-turn*(*left-or-straight*) would require to label both *left* and *straight* as necessarily available, and ban directions *right* and *uturn*.

The (unique) answer set for the set of edge atoms G and an effect mapping P including rules (4.21), (4.22) and (4.3) gives us exactly four effects for the Mandatory Left Turn, $I_1 = \{s(mand\text{-}turn(left), a_4)\}$, as depicted.

$$\mathcal{F}_G^P(I_1) = \{f(ban, a_4, a_3), f(ban, a_4, a_5), f(ban, a_4, a_7), f(nec, a_4, a_1)\}$$

The meaning of a No Entry sign at a node Y can be captured by banning all incoming edges to Y.

$$f'(\ell_s, ban, X, Y) \leftarrow s(no\text{-}entry, Y), e(T, X, Y) \qquad (4.27)$$

Adding this rule to program P and using the No Entry sign at b_7, $I_2 = \{s(no\text{-}entry, b_7)\}$, leads to the expected effects:

$$\mathcal{F}_G^P(I_2) = \{f(ban, b_2, b_7), f(ban, b_4, b_7), f(ban, b_6, b_7), f(ban, b_8, b_7)\} \qquad \blacksquare$$

The effect of turning bans can be represented in a similar way by labelling the prohibited edges with *ban*.

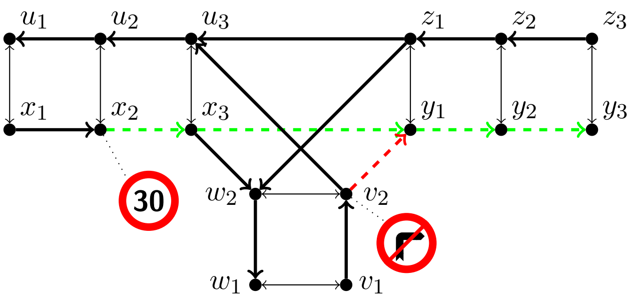


Figure 4.9: Propagation of a speed limit restriction due to a No Right Turn

Example 22 Figure 4.8 shows a No Left Turn at node a_4 and a No Right Turn at b_4. In case of No Left Turns, the U-turn is also banned.

$$f'(\ell_s, \mathit{ban}, X, Y) \leftarrow s(\mathit{no\text{-}turn}(T), X), e(T, X, Y) \tag{4.28}$$
$$f'(\ell_s, \mathit{ban}, X, Y) \leftarrow s(\mathit{no\text{-}turn}(\mathit{left}), X), e(\mathit{uturn}, X, Y) \tag{4.29}$$

With these rules in P input $I = \{s(\mathit{no\text{-}turn}(\mathit{left}), a_4), s(\mathit{no\text{-}turn}(\mathit{right}), b_4)\}$ yields

$$\mathcal{F}_G^P(I) = \{f(\mathit{ban}, a_4, a_1), f(\mathit{ban}, a_4, a_3), f(\mathit{ban}, b_4, b_5)\}.$$ ■

To describe corresponding measures for these signs we will pragmatically use only two measure types, *traffic* and *no-traffic*. These labels shall be directly assigned to edges which are necessarily available, resp. banned for traffic.

Example 23 A corresponding measure information for the Mandatory Left Turn in Figure 4.7 is given by the following atoms.

$$I = \{m(\mathit{traffic}, a_4, a_1)\} \cup \{m(\mathit{no\text{-}traffic}, a_4, \alpha) \mid \alpha \in \{a_3, a_5, a_7\}\}$$

With the general rule (4.4), we can extend the effect mapping P simply by two new facts: $\mathit{m2f}(\mathit{traffic}, \mathit{nec})$ and $\mathit{m2f}(\mathit{no\text{-}traffic}, \mathit{ban})$. ■

Note that nothing opposes using other measure types than *traffic* and *no-traffic* and more complex rules to derive *nec* and *ban* labels. Without a specific data source in mind it is not useful to speculate about various data modelling possibilities. Here, we want to focus more on high-level principles for which it suffices to use simple measure representations as shown. While the interpretation of traffic measures can be assumed to be given in some form of direct mapping between measure types and effect types, the meaning of traffic signs is intrinsically context dependent.

Example 24 Figure 4.9 depicts the situation in which the speed limit restriction is propagated after a junction without another start sign. The No Right Turn at node v_2 now bans incoming traffic from the only incoming street. Since U-turns are ignored for practical reasons, all (considered) drivers arriving at node y_1 are aware of the speed

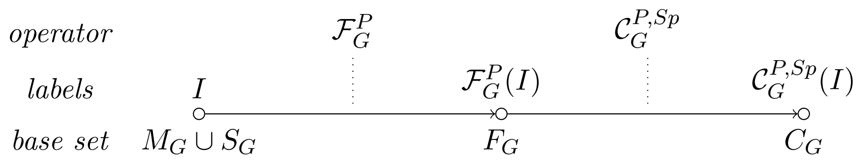


Figure 4.10: Illustration of the two-stage label mapping approach

limit restriction expressed by the start sign at x_2. A *ban* effect will be derived for edge (v_2, y_1) by rule (4.28), and so $\textit{has-perm-inc}(\ell_s, y_1)$ can no longer be concluded. As a consequence, the propagation is not blocked anymore and continues through y_1. The effects of $I = \{s(\textit{start}(\textit{spl}(30)), x_2), s(\textit{no-turn}(\textit{right}), v_2)\}$ thus are

$$\begin{aligned}\mathcal{F}_G^P(I) \;\; &= \;\; \{f(\textit{ban}, v_2, y_1), f(\textit{max-speed}(30), x_2, x_3), f(\textit{max-speed}(30), x_3, y_1),\\ &\qquad f(\textit{max-speed}(30), y_1, y_2), f(\textit{max-speed}(30), y_2, y_3)\}.\end{aligned}$$

Now we can describe and derive the meaning of measures and signs. What remains to be done is to capture traffic regulations for such scenarios, i.e., specifications by which we can distinguish consistent scenarios from inconsistent ones.

4.5 Conflict Specification

Similar to an effect mapping, defining the meaning of traffic measures and traffic signs, we will use another mapping to interpret effect labels.

Definition 11 (Conflict specification) *Let P be an effect mapping relative to a street graph G. A* conflict specification over P *is a set Sp of formulas in $\mathcal{L}$ that associates with each input $I \subseteq I_G$ the set*

$$\mathcal{C}_G^{P,Sp}(I) = Cn_G(Sp, \mathcal{F}_G^P(I), C_G)$$

of atoms, called the conflicts of I (on G).

We assume that Sp is designed in such a way that an empty scenario $(G, \emptyset, \emptyset)$ is free of conflicts, i.e., that $\mathcal{C}_G^{P,Sp}(I) \neq \emptyset$ implies $I \neq \emptyset$.

The setup to compute conflicts based on effects, given a conflict specification Sp, is the same as computing effects from measures and signs, given an effect mapping P. The first stage builds a context-dependent model of the input, the second stage establishes the basis for reasoning tasks. Figure 4.10 illustrates the composition of those mappings. However, when using ASP, we can conveniently compute these two stages in one step.

Example 25 Figure 4.11 shows the speed limit restriction scenario, where only the first start sign at x_2 and the end sign at y_2 are given, and no measure information.

$$I = \{s(\textit{start}(\textit{spl}(30)), x_2), s(\textit{end}(\textit{spl}(30)), y_2)\}$$

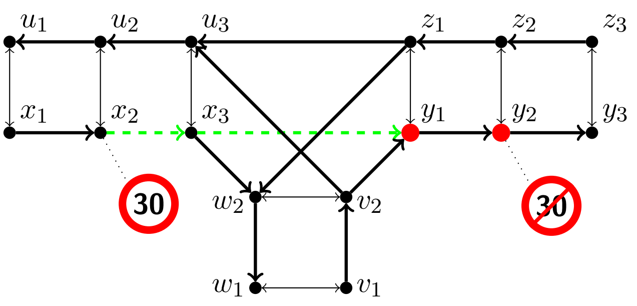


Figure 4.11: Conflicts arising from missing start sign of a 30 km/h speed limit

The red dots at y_1 and y_2 indicate two conflicts. The problem at y_1 is that an effect ends without any traffic sign expressing this end. We label nodes with this kind of conflict with type *bad-end*. (Note that a dual conflict *bad-start* cannot exist, since a start sign always expresses its own first effect label.) At node y_2 there is an (explicit) end sign for an effect that does not exist on the previous edge in direction of traffic. In such a case we want to derive the conflict type *cant-end*. We specify these undesired situations in a new answer set program Sp as follows.

$$\begin{aligned} c(\mathit{bad\text{-}end}(F), Y) &\leftarrow \mathit{in\text{-}dir}(X, Y), \mathit{in\text{-}dir}(Y, Z), && (4.30)\\ &\quad f'(\ell_s, F, X, Y), \text{not } f'(\ell_s, F, Y, Z), &&\\ &\quad \mathit{needs\text{-}expr\text{-}end}(F), \text{not } \mathit{expr\text{-}end}(F, Y) &&\\ c(\mathit{cant\text{-}end}(F), Y) &\leftarrow s(T, Y), \mathit{expl\text{-}end\text{-}of}(T, F), \mathit{in\text{-}dir}(X, Y), && (4.31)\\ &\quad \text{not } f'(\ell_s, F, X, Y) && \end{aligned}$$

Since some effects automatically end, e.g., at the next junction, we use the predicate $\mathit{needs\text{-}expr\text{-}end}(F)$ to test whether the effect F needs to be ended by a traffic sign. This is the case for speed limits, so we additionally add the following rule.

$$\mathit{needs\text{-}expr\text{-}end}(\mathit{max\text{-}speed}(K)) \leftarrow \mathit{speed}(K) \qquad (4.32)$$

These rules specify errors in traffic sign posting, so we explicitly test for sign effects. If we were to test for "not $f(F, X, Y)$" instead of "not $f'(\ell_s, F, X, Y)$" in rule (4.30), also a measure's effect of type F might prevent the rule from firing. In our example, adding $m(\mathit{spl}(30), y_1, y_2)$ would lead to an effect $f(\mathit{max\text{-}speed}(30), y_1, y_2)$ and the *bad-end* conflict could not be derived. The example makes clear that we want to separately test the consistency of traffic measures and traffic signs. For this reason, we use the helper predicate symbol f'.

Again, we use the graph as a set of edge atoms G and the effect mapping P developed in the previous examples. For the input $I = \{s(\mathit{start}(\mathit{spl}(30), x_2)), s(\mathit{end}(\mathit{spl}(30)), y_2)\}$ we get the following effects.

$$\mathcal{F}_G^P(I) = \{f(\mathit{max\text{-}speed}(30), x_2, x_3), f(\mathit{max\text{-}speed}(30), x_3, y_1)\}$$

We recall that, in case of Answer Set Programming, computing $Cn_G(T, X, Y)$ amounts to filtering atoms of the cautious consequence of the program $T \cup G \cup X$ which are in Y, i.e., $\mathcal{AS}^{\cap}_{Y}(T \cup G \cup X)$. Hence, we get the conflicts $\mathcal{C}^{P,Sp}_{G}(I) = Cn_G(Sp, \mathcal{F}^{P}_{G}(I), C_G)$ as $\mathcal{AS}^{\cap}_{C_G}(Sp \cup G \cup \mathcal{F}^{P}_{G}(I))$. The effect mapping leads to a unique answer set and since P is well-designed in the sense that no effect is dependent on conflicts, the latter amounts to $\mathcal{AS}^{\cap}_{C_G}(Sp \cup P \cup G \cup I)$. The fact that we can put Sp and P together allows us to use input atoms and helper predicates from P also in rules of Sp, making the conflict specification more readable and intuitive. The program $Sp \cup P \cup G \cup I$ also has a single answer set, including only the expected conflicts:

$$\mathcal{C}^{P,Sp}_{G}(I) = \{c(\mathit{bad\text{-}end}(\mathit{max\text{-}speed}(30)), y_1), c(\mathit{cant\text{-}end}(\mathit{max\text{-}speed}(30)), y_2)\}$$

Now we also see why it is necessary to ask for an *explicit* end in rule (4.31). Since a start sign for a speed limit K is also an (implicit) end sign for all other speed limits $J \neq K$, every speed limit start sign would lead to incorrect conflicts. That is, if we replace $\mathit{expl\text{-}end\text{-}of}(T, F)$ by $\mathit{end\text{-}of}(T, F)$ in rule (4.31), i.e., using instead rule

$$\begin{array}{rcl} c(\mathit{cant\text{-}end}(F), Y) & \leftarrow & s(T, Y), \mathit{end\text{-}of}(T, F), \mathit{in\text{-}dir}(X, Y), \\ & & \text{not } f'(\ell_s, F, X, Y)\,, \end{array}$$

every start sign for a speed limit K on a node x would result in a conflict predicate $c(\mathit{cant\text{-}end}(\mathit{max\text{-}speed}(J)), x)$ for each speed J that was not the maximum speed on the previous edge. ∎

To support query answering on top of conflicts, aspects of interest may be encoded in the conflict specification, using designated conflicts and formulas defining them in Sp, or using a separate conflict specification Q instead.

Given an input I on a graph G, querying for a certain conflict type (or aspect of interest) $t \in \mathsf{C}$ amounts to computing the set $\{c(t, v) \in \mathcal{C}^{P,Sp}_{G}(I) \mid v \in V\}$, resp. $\mathcal{C}^{P,Q}_{G}(I)$. In our Answer Set Programming setting, the first approach with Sp amounts to computing $\mathcal{AS}^{\cap}_{C^t_G}(Sp \cup P \cup G \cup I)$, where C^t_G is the subset of C_G with conflict type t. With a separate specification Q, we get the intended query result by $\mathcal{AS}^{\cap}_{C_G}(Q \cup P \cup G \cup I)$.

Example 26 Figure 4.12 shows inadmissible combinations of traffic signs. First, observe the Mandatory Left Turn at node a_4, which requires the edge (a_4, a_1) to be trafficable and bans all other outgoing edges. Due to the No Entry sign posted at a_1, we have a contradiction since it bans all incoming edges, including (a_4, a_1). To detect such contradictory labels we add to the conflict specification Sp the following rules.

$$c(\mathit{overlap}(L, F_1, F_2), X) \leftarrow f'(L, F_1, X, Y), f'(L, F_2, X, Y), \mathit{contr}(F_1, F_2) \quad (4.33)$$

$$\mathit{contr}(\mathit{ban}, \mathit{nec}) \quad (4.34)$$

The first rule says two contradicting effect types F_1 and F_2 overlap at a node X due to the language L, if L expresses effects of both types on some edge starting at X. We

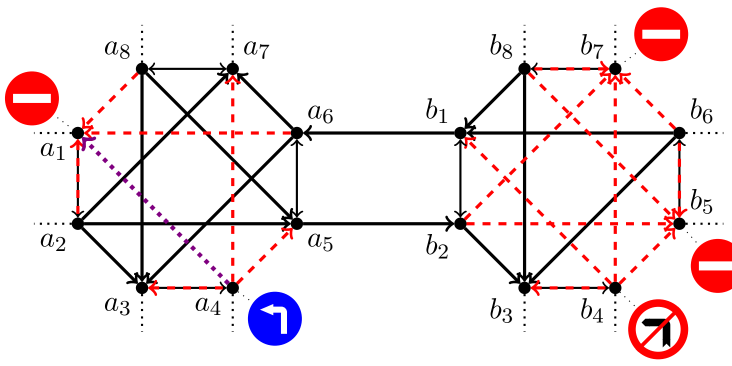


Figure 4.12: Contradicting traffic bans

can specify other such contradictions, like ambiguous speed limits or motorways leading through residential areas.

$$contr(max\text{-}speed(K), max\text{-}speed(J)) \leftarrow speed(K), speed(J), K < J \qquad (4.35)$$
$$contr(motorway, residential\text{-}area) \qquad (4.36)$$

The discussed traffic signs are given by $I = \{s(no\text{-}entry, a_1), s(mand\text{-}turn(left), a_4)\}$.

$$\mathcal{C}_G^{P,Sp}(I) = \{c(overlap(\ell_s, ban, nec), a_4)\}\,.$$

However, there is a second problem at a_4, which also occurs at b_4. If a driver arrives at any of these nodes, she cannot drive any further, since all outgoing edges are banned. More general, whenever there is a sign at a node X, there must be a way out, i.e., some out-node must be reachable.

$$c(no\text{-}way\text{-}out(\ell_m), X) \leftarrow m(T, X, Y), \text{not } way\text{-}out(\ell_m, X) \qquad (4.37)$$
$$c(no\text{-}way\text{-}out(\ell_s), X) \leftarrow s(T, X), \text{not } way\text{-}out(\ell_s, X) \qquad (4.38)$$
$$way\text{-}out(L, X) \leftarrow lang(L), out\text{-}node(X) \qquad (4.39)$$
$$way\text{-}out(L, X) \leftarrow lang(L), reach(L, X, Y), way\text{-}out(L, Y) \qquad (4.40)$$
$$reach(L, X, Y) \leftarrow lang(L), e(T, X, Y), \text{not } f'(L, ban, X, Y) \qquad (4.41)$$
$$reach(L, X, Z) \leftarrow reach(L, X, Y), reach(L, Y, Z) \qquad (4.42)$$

In our example, a_1, a_3, a_7, and b_3, b_5, b_7 are out-nodes. None of these are (ℓ_s-)reachable from a_4 and b_4. With $I' = I \cup \{s(no\text{-}turn(left), b_4), s(no\text{-}entry, b_5), s(no\text{-}entry, b_7)\}$:

$$\mathcal{C}_G^{P,Sp}(I') = \{c(no\text{-}way\text{-}out(\ell_s), a_4), c(no\text{-}way\text{-}out(\ell_s), b_4), c(overlap(\ell_s, nec, ban), a_4)\}$$

Suppose we are interested only in the conflict of type $no\text{-}way\text{-}out(\ell_s)$. A query for this kind of problem amounts to $\mathcal{AS}_C^{\cap}(Sp \cup P \cup G \cup I)$, where C is the subset of C_G with conflict type $no\text{-}way\text{-}out(\ell_s)$. Alternatively, we may use a new program Q including rules (4.38)-(4.42) and get the conflicts of interest:

$$\mathcal{C}_G^{P,Q}(I') = \{c(no\text{-}way\text{-}out(\ell_s), a_4), c(no\text{-}way\text{-}out(\ell_s), b_4)\} \qquad ■$$

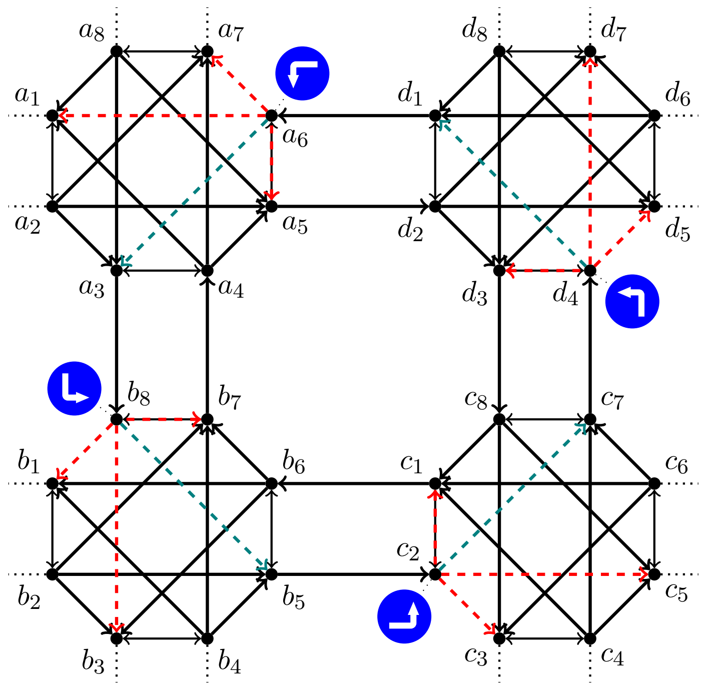


Figure 4.13: Loop caused by four Mandatory Left Turns

The following example demonstrates that formally plausible rules are not necessarily practicable. With experience, such specifications may often be refined, which can conveniently be done with ASP rules.

Example 27 Figure 4.13 shows another scenario with different kinds of dead ends. In this case, the four Mandatory Left Turn signs create a loop. Note that in principle, a driver can escape this loop by using a U-turn after a junction, e.g. (a_3, a_4), or in reality, at any point along the lanes where we do not model U-turns, unless needed. Nevertheless, if such a situation occurs, it is certainly not intended and worth detecting. Therefore, we pragmatically replace rule (4.41) by the following two:

$$\mathit{reach}(L, X, Y) \leftarrow \mathit{lang}(L), e(T, X, Y), \text{not } f'(L, \mathit{ban}, X, Y), T \neq \mathit{uturn} \tag{4.43}$$

$$\mathit{reach}(L, X, Y) \leftarrow e(\mathit{uturn}, X, Y), f'(L, \mathit{nec}, X, Y) \tag{4.44}$$

■

The first rule disregards U-turns as part of reachable ways. However, this is neither intuitive nor practical for Mandatory U-turns. The second rule accounts for this case.

Note that if we replace in the second rule the constant *uturn* by a variable, the conflict $c(\mathit{no\text{-}way\text{-}out}(\ell_s), a_4)$ of the previous example will not be derivable anymore. It is a matter of taste how to interpret such a case of contradicting permissions on some edge, so both variants can be argued for. For the loop example we use again according edges G

to represent the graph and the input $I = \{s(\mathit{mand\text{-}turn}(\mathit{left}), \alpha) \mid \alpha \in \{a_6, b_8, c_2, d_4\}\}$. We get the following conflicts.

$$\begin{aligned} \mathcal{C}_G^{P,Sp}(I) \quad = \quad & \{c(\mathit{no\text{-}way\text{-}out}(\ell_s), a_6), c(\mathit{no\text{-}way\text{-}out}(\ell_s), b_8), \\ & c(\mathit{no\text{-}way\text{-}out}(\ell_s), c_2), c(\mathit{no\text{-}way\text{-}out}(\ell_s), d_4)\} \end{aligned}$$

Given an effect mapping and a conflict specification, a traffic regulation as viewed in this work is sufficiently defined.

Definition 12 (Traffic regulation problem) *Let $Sc = (G, M, S)$ be a scenario, P an effect mapping relative to G and let Sp be a conflict specification over P. Then, the pair $\Pi = (P, Sp)$ is called a* traffic regulation *and the pair (Π, Sc) a* traffic regulation problem*.*

Technically, a conflict specification Sp is not limited to the traffic regulation in the narrow sense of the word. Already the loop example showed the usefulness to include additional aspects which may not be found in legal documents. Moreover, our high-level approach is suitable to detect any point of interest which arises due to the constellation of traffic regulation data on a street map, as discussed in Chapter 3. In this chapter we have developed a specific effect mapping P and a conflict specification Sp, which gives us a traffic regulation $\Pi = (P, Sp)$. Thus, all presented examples of this chapter yield a traffic regulation problem by the according scenario $Sc = (G, M, S)$. We are now prepared to discuss inconsistency management in form of reasoning tasks based on these examples in the next chapter.

CHAPTER 5

Reasoning Tasks

In the previous chapter we developed a formal model of street graphs, the two considered input languages of traffic measures and traffic signs, and a logic-based traffic regulation in form of an effect mapping and a conflict specification. On top of this, we will now present different practically relevant reasoning tasks to inspect data inconsistencies and define mechanisms to eliminate them. In particular, we will investigate the following problems, given a traffic regulation problem $\mathcal{T}$.

- Consistency Evaluation. Determine whether $\mathcal{T}$ is inconsistent w.r.t. the traffic regulation, i.e., whether it has any conflicts.
- Diagnoses. For a given set of conflicts C of $\mathcal{T}$, find explanations, i.e., parts of the input that causes C.
- Correspondence. Determine whether measures and signs in $\mathcal{T}$ express the same restrictions on road use, i.e., the same effects.

In case we have an inconsistent traffic regulation problem $\mathcal{T}$, the most important question is how to establish consistency by modifications to the input. We distinguish four different kinds of such data updates.

- Repairs. Which measures and signs must be added or deleted such that the resulting traffic regulation problem has no conflicts?
- Strict Repairs. For an inconsistent $\mathcal{T}$ comprising both measure and sign data, find a repair that also ensures correspondence.
- Adjustment. If among the two languages, measures and signs, data of only one kind is consistent, find a strict repair that adapts the other language's data towards correspondence.

- GENERATION. Given consistent data of only one language, generate corresponding data of the other language from scratch.

Throughout this chapter, $\mathcal{T}$ we will always denote a traffic regulation problem (Π, Sc) with the traffic regulation $\Pi = (P, Sp)$ developed throughout the examples of the previous chapter. We will use different scenarios, always denoted by $Sc = (G, M, S)$ with traffic measures M and traffic signs S on a graph G, and again call the respective set $I = M \cup S$ the scenario's input. Depending on context, the notion *input* will also be used for subsets $J \subseteq I_G$ of arbitrary measures and signs on the graph.

5.1 Consistency Evaluation

The first reasoning task is central to our approach of inconsistency management. In Chapter 4 we introduced conflicts, which are entailed whenever the input $I = M \cup S$ contains illegal constellations as formalized by the conflict specification Sp (over an effect mapping P). We define inconsistency as the existence of conflicts.

Definition 13 (Inconsistency) *The* conflicts of *a traffic regulation problem $\mathcal{T}$ with input I are given by $\mathcal{C}(\mathcal{T}) = \mathcal{C}_G^{P,Sp}(I)$. If $\mathcal{C}(\mathcal{T}) \neq \emptyset$, we call $\mathcal{T}$, its scenario Sc, and I* inconsistent *(w.r.t.* Π*).*

We define the first reasoning task by requiring the computation of $\mathcal{C}(\mathcal{T})$.

Problem:	CONSISTENCY EVALUATION (CONSEVAL)
Instance:	A traffic regulation problem $\mathcal{T}$
Output:	The set of conflicts $\mathcal{C}(\mathcal{T})$

Accordingly, we also write CONSEVAL($\mathcal{T}$) or $\mathcal{C}(\mathcal{T})$. In the previous chapter we have already seen examples of inconsistent traffic regulation problems. Note that the presented ASP-based definition of Π gave a declarative implementation for CONSEVAL. That is, by instantiating the formal model with formulas in ASP we already have the means to determine the conflicts of a traffic regulation problem.

In this chapter, we deal with two major questions concerning the conflicts of an inconsistent $\mathcal{T}$: how they are explained and how they can be removed.

5.2 Diagnosis

Given an inconsistent traffic regulation problem $\mathcal{T}$, we are interested in the part of the input that causes its conflicts. In terms of abductive diagnosis [39, 42], the input serves as set of hypotheses from which we need to explain the observed conflicts based on the traffic regulation and the street graph as underlying theory.

Definition 14 (Diagnosis) *Let $\mathcal{T}$ be an inconsistent traffic regulation problem with input I. A* diagnosis *for a (non-empty) set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$ is a set $J \subseteq I$, such that $C \subseteq \mathcal{C}_G^{P,Sp}(J)$. The set of all diagnoses for C (with fixed $\mathcal{T}$) is denoted by $\mathcal{D}_{\mathcal{T}}(C)$.*

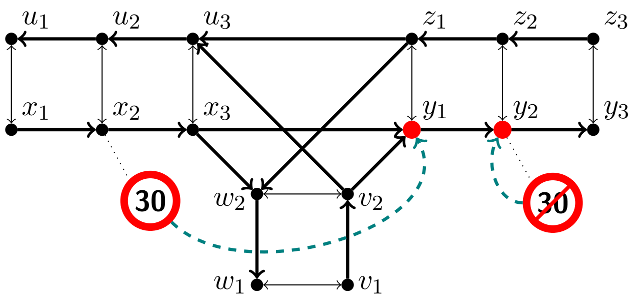


Figure 5.1: Traffic signs causing conflicts due to a missing repeated start sign at y_1

We assume that Sp is defined such that $\mathcal{C}_G^{P,Sp}(\emptyset) = \emptyset$. Consequently, each diagnosis (for $C \neq \emptyset$) is non-empty. The respective reasoning task is defined as follows.

Problem:	DIAGNOSES
Instance:	A traffic regulation problem $\mathcal{T}$ and a set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$
Output:	The set of all diagnoses $\mathcal{D}_\mathcal{T}(C)$ for C

We also write DIAGNOSES$(\mathcal{T}, C)$ for $\mathcal{D}_\mathcal{T}(C)$. For a single conflict $c \in \mathcal{C}(\mathcal{T})$, we abbreviate $\mathcal{D}_\mathcal{T}(\{c\})$ by $\mathcal{D}_\mathcal{T}(c)$. In particular, we are interested in *minimal* diagnoses with respect to cardinality or subset inclusion. The sets of these diagnoses are denoted by $\mathcal{D}_\mathcal{T}^{\#}(C)$ and $\mathcal{D}_\mathcal{T}^{\subseteq}(C)$, respectively. If we speak only of minimal diagnoses, we mean $\subseteq$-minimal.

Example 28 Recall the traffic regulation problem $\mathcal{T}$ with the speed limit scenario depicted again in Figure 5.1, where the missing repeated start sign for the 30 km/h restriction leads to $\mathcal{C}(\mathcal{T}) = C_1 \cup C_2$, where

$$\begin{aligned} C_1 &= \{c(\mathit{bad\text{-}end}(\mathit{max\text{-}speed}(30)), y_1)\}, \text{ and} \\ C_2 &= \{c(\mathit{cant\text{-}end}(\mathit{max\text{-}speed}(30)), y_2)\}. \end{aligned}$$

The entire input I is the only diagnosis for $\mathcal{C}(\mathcal{T})$, i.e., $\{I\} = \mathcal{D}_\mathcal{T}^{\#}(\mathcal{C}(\mathcal{T})) = \mathcal{D}_\mathcal{T}^{\subseteq}(\mathcal{C}(\mathcal{T}))$. The single minimal diagnosis for C_1 is $\{s(\mathit{start}(\mathit{spl}(30)), x_2)\}$, since the incorrectly ended $\mathit{max\text{-}speed}(30)$ effect stems from this sign. Independently, the end sign at y_2 leads to the other conflict, i.e., C_2 is minimally explained by $\{s(\mathit{end}(\mathit{spl}(30)), y_2)\}$. ■

We formalize this idea of independence between sets of conflicts and subsets of the input.

Definition 15 (Independence) *Let $\mathcal{T}$ be an inconsistent traffic regulation problem with input I. We say a set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$ is* independent *of a set $Y \subseteq I$, if for each diagnosis J for C and each $Y' \subseteq Y$, the set $J \setminus Y'$ is also a diagnosis for C.*

The notion of independence captures which part of the input is irrelevant with respect to the explanation for a given set of conflicts.

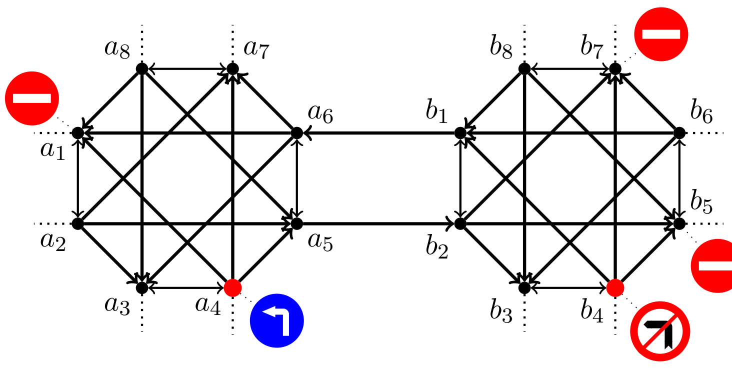


Figure 5.2: Conflicts caused by sets of two, respectively three traffic signs

Example 29 The input I of the scenario depicted in Figure 5.2 is given by the union of the following sets of signs:

$$\begin{aligned} I_a &= \{s(\mathit{no\text{-}entry}, a_1), s(\mathit{mand\text{-}turn}(\mathit{left}), a_4)\} \\ I_b &= \{s(\mathit{no\text{-}turn}(\mathit{left}), b_4), s(\mathit{no\text{-}entry}, b_5), s(\mathit{no\text{-}entry}, b_7)\} \end{aligned}$$

This traffic regulation problem has the set of conflicts $\mathcal{C}(\mathcal{T}) = C_a \cup C_b$, where

$$\begin{aligned} C_a &= \{c(\mathit{no\text{-}way\text{-}out}(\ell_s), a_4), c(\mathit{overlap}(\ell_s, \mathit{ban}, \mathit{nec}), a_4)\}, \text{ and} \\ C_b &= \{c(\mathit{no\text{-}way\text{-}out}(\ell_s), b_4)\}. \end{aligned}$$

By a set $I_{x_n} \subseteq I$ we denote the singleton comprising the sign posted at node x_n. The diagnoses for each non-empty set of conflicts $C'_a \subseteq C_a$ are then given as follows:

$$\begin{aligned} D_a^1 &= I_a \cup I_{b_4} \cup I_{b_5} \cup I_{b_7} \\ D_a^2 &= I_a \cup I_{b_4} \cup I_{b_5} \\ D_a^3 &= I_a \cup I_{b_4} \cup I_{b_7} \\ D_a^4 &= I_a \cup I_{b_5} \cup I_{b_7} \\ D_a^5 &= I_a \cup I_{b_4} \\ D_a^6 &= I_a \cup I_{b_5} \\ D_a^7 &= I_a \cup I_{b_7} \\ D_a^8 &= I_a \end{aligned}$$

Hence, each C'_a is independent of I_b. Likewise, C_b is independent of I_a, as we see by enumerating the diagnoses for C_b:

$$\begin{aligned} D_b^1 &= I_b \cup I_{a_1} \cup I_{a_4} \\ D_b^2 &= I_b \cup I_{a_1} \\ D_b^3 &= I_b \cup I_{a_4} \\ D_b^4 &= I_b \end{aligned}$$

We get a dual view on independence by the observation that a set of conflicts C occurs due to a *context*, i.e., a maximal subset $X \subseteq I$, such that C is not independent of X.

Definition 16 (Context) *A subset $X \subseteq I$ of the input is called a* context *of a set of conflicts C, if the following conditions hold:*

(i) C is independent of $I \setminus X$, and

(ii) C is not independent of any non-empty $X' \subseteq X$.

Using the scenario with the speed limit signs shown in Figure 5.1, and the notation of Example 28, $\{s(start(spl(30)), x_2)\}$ is a context of C_1, and $\{s(end(spl(30)), y_2)\}$ is a context of C_2. In Example 29, we get a context I_a for C_a, and I_b is a context for C_b. As these examples suggest, the context of a set of conflicts is always unique.

Proposition 1 *Each set $C \subseteq \mathcal{C}(\mathcal{T})$ of conflicts has a unique context.*

Proof. Let I be the input of a traffic regulation problem $\mathcal{T}$ and let X, Y be two contexts of a set $C \subseteq \mathcal{C}(\mathcal{T})$ of conflicts. For the sake of contradiction, we assume that $X \neq Y$. Without loss of generality, we can assume some $y \in Y$, such that $y \notin X$. Since X is a context, by Definition 16 (i), C is independent of $I \setminus X$ and thus in particular, independent of $\{y\}$. However, since Y is a context, by Definition 16 (ii), C is not independent of any non-empty $Y' \subseteq Y$, and thus in particular, not independent of $\{y\}$, which gives the contradiction. □

In the sequel, the context for a set C of conflicts is denoted by $ctx(C)$. The $\subseteq$-minimal parts capture the essence of a diagnosis. In a similar way, the $\subseteq$-minimal diagnoses themselves capture the essence of a context.

Proposition 2 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts. Then all $\subseteq$-minimal diagnoses are contained in the context, i.e., $\bigcup \mathcal{D}^{\subseteq}_{\mathcal{T}}(C) \subseteq ctx(C)$.*

Proof. Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts and let $j \in \bigcup \mathcal{D}^{\subseteq}_{\mathcal{T}}(C)$. Towards a contradiction, assume that $j \notin ctx(C)$. Since C is independent of $I \setminus ctx(C)$ it must hold that, whenever J is a diagnosis for C and $Y \subseteq I \setminus ctx(C)$, then $J \setminus Y$ is also a diagnosis for C. Let J be any $\subseteq$-minimal diagnosis containing j. By assumption, $\{j\} \subseteq I \setminus ctx(C)$, but $J \setminus \{j\}$ is not a diagnosis for C, which yields the contradiction. □

Examples 28 and 29 suggest that the context of a set of conflicts is determined by its minimal diagnoses. Our next example shows that this is not always the case. Thus, in general, the converse of Proposition 2 does not hold.

Example 30 Figure 5.3 shows the correct announcement of a 30 km/h speed limit restriction from x_2 to x_3. However, at x_4 there is another end sign, causing a single

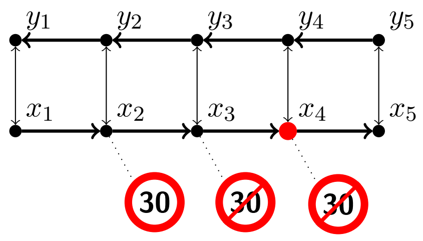


Figure 5.3: A repeated end sign causes a conflict at x_4

conflict $C = \{c(\mathit{bad\text{-}end}(\mathit{max\text{-}speed}(30)), x_4)\}$. To list its diagnoses, we denote by s_n the respective traffic sign at node x_n.

$$\begin{aligned} D_1 &= \{s_4\} \\ D_2 &= \{s_3, s_4\} \\ D_3 &= \{s_2, s_3, s_4\} \end{aligned}$$

We observe that s_4 must be in the context due the $\subseteq$-minimality of D_1. The only diagnosis in which s_2 appears is D_3. Since $D_3 \setminus \{s_2\}$ is also a diagnosis for C, C is independent of $\{s_2\}$. However, if we remove s_3 from D_3, we do not obtain a diagnosis. Consequently, the context contains s_3, which is not part of a minimal diagnosis. ■

In the previous example we saw that the context may contain more elements than those contained in minimal diagnoses. Towards a full characterization of a context, we introduce intervals and the notion of convexity.

Definition 17 (Interval) *Let S and S' be two sets such that $S \subseteq S'$. The* interval from S to S' *is defined as* $[S, S'] = \{S'' \mid S \subseteq S'' \subseteq S'\}$.

By this distinguishing property of a collection of sets we define convexity.

Definition 18 (Convexity) *Let $\mathcal{X}$ be a collection of sets and let S and S' be two sets such that $S \subseteq S'$. The pair (S, S') is called* convex in $\mathcal{X}$ *if $[S, S'] \subseteq \mathcal{X}$. A pair of sets (S, S') is* maximal convex *in $\mathcal{X}$ if (S, S') is convex in $\mathcal{X}$, and for every pair (T, T') where $T \subset S$ or $T' \supset S'$, (T, T') is not convex in $\mathcal{X}$. By $\Diamond\mathcal{X}$ we denote the collection of maximal convex pairs of sets in $\mathcal{X}$.*

In Example 30, the set of diagnoses for C was $\mathcal{D}_\mathcal{T}(C) = \{D_1, D_2, D_3\}$. The set of maximal convex pairs of sets of this collection is given by $\Diamond\mathcal{D}_\mathcal{T}(C) = \{(D_1, D_2), (D_2, D_3)\}$.

If there are two different diagnoses J and $J \setminus \{x\}$ for C and (J, J') is maximal convex in $\mathcal{D}_\mathcal{T}(C)$, then $J \neq J'$, i.e., the interval $[J, J']$ contains at least two sets.

Lemma 3 *Let (J, J') be maximal convex in a set of diagnoses $\mathcal{D}_\mathcal{T}(C)$ and let $x \in J$. If $J \setminus \{x\} \in \mathcal{D}_\mathcal{T}(C)$, then J is a proper subset of J', i.e., $J \subset J'$.*

Proof. Let $(J, J') \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$, $x \in J$ and suppose that $J \setminus \{x\} \in \mathcal{D}_{\mathcal{T}}(C)$. For the sake of contradiction, assume $J \not\subset J'$, i.e., $J = J'$. That is, (J, J) is maximal convex in $\mathcal{D}_{\mathcal{T}}(C)$. However, both J and $J \setminus \{x\}$ are diagnoses for C and thus $(J \setminus \{x\}, J)$ is convex in $\mathcal{D}_{\mathcal{T}}(C)$. Consequently, (J, J) cannot be maximal convex in $\mathcal{D}_{\mathcal{T}}(C)$, giving the contradiction. □

We now characterize the context of a set of conflicts by the union of all $\subseteq$-minimal sets in maximal convex pairs of its diagnoses.

Theorem 4 *Let $\mathcal{T}$ be an inconsistent traffic regulation problem and let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts. Then, $ctx(C) = \bigcup\{J \mid (J, J') \in \Diamond\mathcal{D}_{\mathcal{T}}(C)\}$.*

Proof. Let I be the input of a traffic regulation problem $\mathcal{T}$, $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts and let $\Diamond\mathcal{D}_{\mathcal{T}}(C)$ be the collection of maximal convex pairs (J, J') in $\mathcal{D}_{\mathcal{T}}(C)$. That is, given a pair (J, J') of diagnoses in $\Diamond\mathcal{D}_{\mathcal{T}}(C)$ and a set J'' for C such that $J \subseteq J'' \subseteq J'$ holds, then $J'' \in \mathcal{D}_{\mathcal{T}}(C)$, i.e., J'' is also a diagnosis for C.

Let $X = \bigcup\{J \mid (J, J') \in \Diamond\mathcal{D}_{\mathcal{T}}(C)\}$. We show that X is the context of C. To see that condition (i) in Definition 16 holds, we assume for the sake of contradiction that C is not independent of $I \setminus X$. That is, there exists a diagnosis J for C and some $Y \subseteq I \setminus X$ such that $J \setminus Y$ is not a diagnosis for C. Let $(J_1, J_2) \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$ such that $J_1 \subseteq J \subseteq J_2$. Since $Y \cap X = \emptyset$ and $J_1 \subseteq X$, we have $J_1 \subseteq J \setminus Y$ and thus $J_1 \subseteq J \setminus Y \subseteq J \subseteq J_2$. By convexity of (J_1, J_2) in $\mathcal{D}_{\mathcal{T}}(C)$, we conclude that $J \setminus Y \in \mathcal{D}_{\mathcal{T}}(C)$. Consequently, $J \setminus Y$ is a diagnosis, which yields the contradiction.

To prove condition (ii), we must show that for each non-empty $X' \subseteq X$ there exists a diagnosis J for C and some $X'' \subseteq X'$ such that $J \setminus X''$ is not a diagnosis for C. Let $X' \subseteq X$ be non-empty. We take some pair of diagnoses $(J_1, J_2) \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$ such that $J_1 \cap X' \neq \emptyset$. Let $x \in J_1 \cap X'$. If $J_1 \setminus \{x\}$ is not a diagnosis, we are done. Hence, we suppose that $D = J_1 \setminus \{x\}$ is a diagnosis for C. By Lemma 3, we have $J_1 \subset J_2$. Since $J_1 = D \cup \{x\}$ is minimal in $[J_1, J_2]$, every J in $[J_1, J_2]$ contains x. We can write J_2 as $D \cup \{x\} \cup Y$, where $Y = \bigcup_{i=1}^{n}\{y_i\}$ for some set of atoms $y_1, \ldots, y_n$, $n \geq 1$, such that $(D \cup \{x\}) \cap Y = \emptyset$. Thus, for each $Y' \subseteq Y$ it holds that $D \cup \{x\} \cup Y' \in [J_1, J_2]$. Since (J_1, J_2) is maximal convex in $\mathcal{D}_{\mathcal{T}}(C)$, there exists a set $S \in [J_1 \setminus \{x\}, J_2] = [D, J_2]$ that is not a diagnosis for C. We conclude that S must be of form $D \cup Y'$ for some $Y' \subseteq Y$. Consequently, we get a diagnosis $J = S \cup \{x\}$ such that $J \setminus \{x\}$ is not a diagnosis. □

Consider a pair of diagnoses $(J_1, J_2) \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$. By Theorem 4, $J_1 \subseteq ctx(C)$. Then, any $(J_1', J_2') \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$, where $J_1' \subseteq J_1$, will not introduce further elements to $ctx(C)$. Thus, to determine the context for C, it suffices to consider those $(J_1, J_2) \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$ where J_1 is maximal in the sense that for all pairs $(J, J') \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$ it holds that $J \not\supset J_1$.

Example 31 (cont'd) To illustrate this characterization, we recall the scenario of Example 30. We had two maximal convex pairs of diagnoses given by $\Diamond\mathcal{D}_{\mathcal{T}}(C) = \{\mathcal{D}_1, \mathcal{D}_2\}$, where $\mathcal{D}_1 = (\{s_4\}, \{s_3, s_4\})$ and $\mathcal{D}_2 = (\{s_3, s_4\}, \{s_2, s_3, s_4\})$. By Theorem 4,

$$\begin{aligned} ctx(C) &= \bigcup\{J \mid (J, J') \in \{(\{s_4\}, \{s_3, s_4\}), (\{s_3, s_4\}, \{s_2, s_3, s_4\})\}\} \\ &= \{s_4\} \cup \{s_3, s_4\} \\ &= \{s_3, s_4\}. \end{aligned}$$

According to the note above, $\mathcal{D}_1$ is redundant in the characterization of $ctx(C)$ in the sense that $\mathcal{D}_2$ implies the inclusion of s_3 and s_4 in $ctx(C)$ and $\mathcal{D}_1$ requires only s_4. ■

We call a collection $\mathcal{X}$ of sets *convex*, if for all $S, S' \in \mathcal{X}$, $S \subseteq S'$ implies that (S, S') is convex in $\mathcal{X}$. If the entire set of diagnoses $\mathcal{D}_{\mathcal{T}}(C)$ for a set C of conflicts is convex, then $(J, J') \in \Diamond\mathcal{D}_{\mathcal{T}}(C)$ implies that J is a $\subseteq$-minimal diagnosis for C. As a consequence, we get the following corollary.

Corollary 5 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts. If $\mathcal{D}_{\mathcal{T}}(C)$ is convex, then the context is given by the union of $\subseteq$-minimal diagnoses, i.e., $ctx(C) = \bigcup\mathcal{D}^{\subseteq}_{\mathcal{T}}(C)$.*

In particular, $\mathcal{D}_{\mathcal{T}}(C)$ is always convex if the employed logic is monotonic, i.e., the property that for all sets of formulas T and T' and atoms α, $T \models \alpha$ implies $T \cup T' \models \alpha$. Then, for subsets J and J' of the input I, $J \subseteq J'$ implies that $\mathcal{C}^{P,Sp}_{G}(J) \subseteq \mathcal{C}^{P,Sp}_{G}(J')$. As a consequence, if J is a diagnosis for C, then every J' such that $J \subseteq J' \subseteq I$ holds is also a diagnosis for C.

Example 32 In Example 29, we named two conflict sets C_a and C_b and listed their respective diagnoses. Both $\mathcal{D}_{\mathcal{T}}(C_a)$ and $\mathcal{D}_{\mathcal{T}}(C_b)$ are convex, and as a consequence, the $\subseteq$-minimal diagnoses determine the respective context. There, the entire context $ctx(C_k)$ is required to diagnose the respective set C_k ($k \in \{a, b\}$). Later, we will see examples with convex sets of diagnoses where $\subseteq$-minimal diagnoses are proper subsets of the context. ■

To see the relation of diagnosis with the usual notion of abductive diagnosis [35], we adjust the definition of the latter, taking into account the closed world operator. A *complete abductive diagnosis (CAD) for* a diagnostic problem $\langle H, T, O\rangle$ is a set of *assumptions* $\Delta \subseteq H$, such that $T \cup \overline{\Delta} \not\models \bot$ and $T \cup \overline{\Delta} \models O$. We immediately obtain the following proposition.

Proposition 6 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be the conflicts of an inconsistent traffic regulation problem $\mathcal{T}$ with input I and $J \subseteq I$. Then, J is a CAD for $\langle I, P \cup \overline{G}, \mathcal{F}^{P}_{G}(J)\rangle$ and $\mathcal{F}^{P}_{G}(J)$ is a CAD for $\langle F_G, Sp \cup \overline{G}, \mathcal{C}^{P,Sp}_{G}(J)\rangle$.*

By definition, a set of measures and signs determines effects given P, which in turn determine conflicts given Sp. Thus, we can conclude the following relation between our notion of diagnosis and complete abductive diagnosis.

Corollary 7 *$J \subseteq I$ is a diagnosis for $C \subseteq \mathcal{C}(\mathcal{T})$ iff $\mathcal{F}^{P}_{G}(J)$ is a CAD for $\langle F_G, Sp \cup \overline{G}, C\rangle$.*

A system that explains the cause of conflicts is useful if a user wants to correct traffic regulation data faults manually. However, we also want to provide semi-automatic support for repairing inconsistent traffic regulation problems.

5.3 Repair

Complementary to diagnoses explaining the cause of inconsistency, an important question is how to repair an inconsistent traffic regulation problem, i.e., by means of which deletions and additions of measures and signs consistency can be established.

Definition 19 (Update) *An* update for $\mathcal{T}$ *with input I is a pair (I^-, I^+), where $I^- \subseteq I$ and $I^+ \subseteq I_G \setminus I$, called the* deletions *and* additions*, respectively. The* updated traffic regulation problem (due to (I^-, I^+))*, denoted by $\mathcal{T}[I^-, I^+]$, is obtained by replacing I with $I' = (I \ \setminus \ I^-) \cup I^+$ in $\mathcal{T}$.*

When applying an update (I^-, I^+) to $\mathcal{T}$, we say I^- is *deleted from* $\mathcal{T}$ and I^+ is *added to* $\mathcal{T}$ to obtain $\mathcal{T}[I^-, I^+]$. In general, we might delete and add both measures and signs.

Definition 20 (Repair) *An update (I^-, I^+) for an inconsistent $\mathcal{T}$ is a* (local) repair for conflicts $C \subseteq \mathcal{C}(\mathcal{T})$ (in $\mathcal{T}$)*, if $C \cap \mathcal{C}(\mathcal{T}[I^-, I^+]) = \emptyset$, i.e., if none of these conflicts can be derived in the updated traffic regulation problem due to (I^-, I^+). An update (I^-, I^+) is a* repair for $\mathcal{T}$*, if $\mathcal{C}(\mathcal{T}[I^-, I^+]) = \emptyset$.*

We observe that a repair for $\mathcal{C}(\mathcal{T})$ is not necessarily a repair for $\mathcal{T}$, since updates (I^-, I^+) may introduce new conflicts. Second, we note that an update which deletes the entire input, i.e. $(I, \emptyset)$, is always a trivial, but rarely desired repair. Further, it seems impractical to compute all possible repairs analogous to DIAGNOSES. Suppose we are given an intuitive repair (I^-, I^+) for $\mathcal{T}$. As long as the updated traffic regulation problem remains consistent, we can remove further items from $I \setminus I^-$ and add arbitrary additional ones from $I_G \setminus (I \cup I^+)$ and get a lot of repairs involving unnecessary modifications. In fact, Definition 20 allows the addition of random elements from I_G, as long as the resulting $\mathcal{T}'$ is consistent.

Example 33 One repair for the traffic regulation problem $\mathcal{T}$ of Figure 5.1 is the addition of another 30 km/h start sign at y_1, i.e., $(\emptyset, I^+)$, where $I^+ = \{s(\mathit{start}(\mathit{spl}(30)), y_1)\}$. However, every random addition to I^+ which does not introduce new conflicts, is also a repair for $\mathcal{T}$. For instance, we could additionally augment the scenario by a No Entry sign at u_3, which is totally unrelated to the problem. ■

This observation suggests that a reasoning task for repairs shall compute only a subset of all possible repairs as defined by an *adequacy criterion.*

Problem:	REPAIRS (for $\mathcal{T}$ under adq)
Instance:	An inconsistent traffic regulation problem $\mathcal{T}$ and an adequacy criterion adq for updates (I^-, I^+)
Output:	The set of repairs (I^-, I^+) for $\mathcal{T}$ such that $\mathit{adq}(I^-, I^+)$ holds

By REPAIRS($\mathcal{T}$, adq) we denote the set of repairs for $\mathcal{T}$ under adq. Similarly, for a fixed set $C \subseteq \mathcal{C}(\mathcal{T})$, REPAIRS($\mathcal{T}, C, \mathit{adq}$) is the set of *(local) repairs for C (in $\mathcal{T}$ under adq)*. Usually, we are interested in establishing consistency by as few modifications as possible, i.e., by *minimal* repairs according to some preference. Without further domain knowledge, we will simply count the number of *changes to the input* $I^- \cup I^+$ as the *(unit) cost* of repairs, i.e., $|I^- \cup I^+|$. We define an adequacy criterion *ucost* which holds for a repair (I^-, I^+) for $\mathcal{T}$, if no repair for $\mathcal{T}$ has lower unit cost, i.e., $|I^- \cup I^+| \leq |J^- \cup J^+|$ for each repair (J^-, J^+) for $\mathcal{T}$.

Example 34 (cont'd) We apply *ucost* as adequacy criterion on Repairs for the traffic regulation problem $\mathcal{T}$ of Figure 5.1. The minimal update $(\emptyset, I^+)$ adds only one sign, hence $ucost(\emptyset, I^+)$ holds. Any repair (J^-, J^+) for $\mathcal{T}$, where $|J^-| > 1$ or $|J^+| > 1$, shall not to be computed by Repairs for $\mathcal{T}$ under *ucost*. Since no repair $(J^-, \emptyset)$ for $\mathcal{T}$ such that $|J^-| = 1$ exists, none of the two given traffic signs will be deleted. ■

Next, we deal with *overconstrained* scenarios in the sense that some input needs to be deleted to establish consistency.

Example 35 The loop example of Figure 4.13 represents a traffic regulation problem $\mathcal{T}$ with too many driving bans. In principle, we could repair this scenario by adding a Mandatory U-turn somewhere along a lane, e.g., between nodes d_1 and a_6. However, this would require to introduce two new nodes and two new edges (labelled *uturn*) to the graph. To avoid such unintuitive repairs, we only consider fixed graphs comprising only those nodes which are necessary to describe the initial input I. In this example, for every non-empty $J \subseteq I$, the update $(J, \emptyset)$ is a repair for $\mathcal{T}$, but $J \in \text{Repairs}(\mathcal{T}, ucost)$ only if $|J| = 1$. Thus, the reasoning task Repairs for this $\mathcal{T}$ under *ucost* will suggest only those repairs which delete exactly one of the four Mandatory Left Turn signs. ■

The theory of consistency-based diagnosis by Reiter [44] deals with fault observations for a fixed system of components, where each component can behave normally or abnormally. Given a set of observations, a *conflict set* denotes a subset of the components that cannot all be assumed to be behave normally without contradicting the system description. Theorem 4.4 in [44] states that minimal consistency-based diagnoses (of according diagnostic problems) can be identified with minimal hitting sets for the collection of conflict sets.

Definition 21 (Hitting set) *Let $\mathcal{S}$ be a collection of sets $X_1, \ldots, X_n$. Then, a subset $H \subseteq \bigcup_{i=1}^{n} X_i$ is a* hitting set *for $\mathcal{S}$, if H* meets *every set in $\mathcal{S}$, i.e., $H \cap X \neq \emptyset$ for each $X \in \mathcal{S}$. A hitting set H is* minimal, *if no $H' \subset H$ is a hitting set.*

We draw upon this idea to investigate the connection between diagnoses and *delete-only repairs*, i.e., repairs of form $(I^-, \emptyset)$. Assume we are given some set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$. If we delete a subset $J \subseteq I$ that meets every diagnosis of C, then all causes of C will be eliminated.

Proposition 8 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts and $\mathcal{D}_{\mathcal{T}}(C)$ be the set of its diagnoses. If a subset $J \subseteq I$ of the input is a hitting set for $\mathcal{D}_{\mathcal{T}}(C)$, then $C \not\subseteq \mathcal{C}(\mathcal{T}[J, \emptyset])$.*

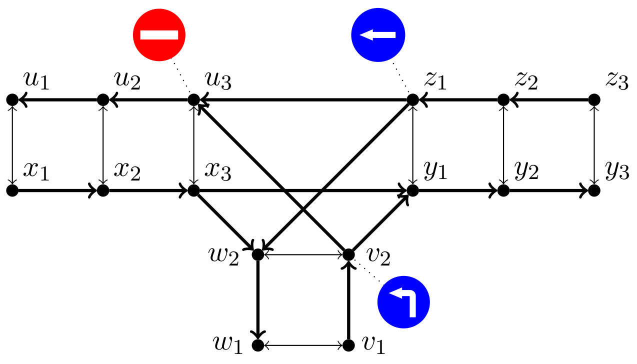


Figure 5.4: Contradictory information for mandatory driving directions

Proof. Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts and let $J \subseteq I$ be a hitting set for $\mathcal{D}_{\mathcal{T}}(C)$, i.e., $J \cap D \neq \emptyset$ for each diagnosis D for C in $\mathcal{T}$. Towards a contradiction, further assume that $C \subseteq \mathcal{C}(\mathcal{T}[J, \emptyset])$, i.e., $C \subseteq \mathcal{C}_G^{P,Sp}(I \setminus J)$. Consequently, $I \setminus J$ is a diagnosis for C, i.e., $(I \setminus J) \in \mathcal{D}_{\mathcal{T}}(C)$. Since $J \cap (I \setminus J) = \emptyset$, J cannot be a hitting set for $\mathcal{D}_{\mathcal{T}}(C)$. It follows that $C \not\subseteq \mathcal{C}(\mathcal{T}[J, \emptyset])$. □

In general, this hitting set condition of an input's subset J w.r.t. the diagnoses of a set of conflicts C is not enough to guarantee that $(J, \emptyset)$ is a repair for C, i.e., $C \cap \mathcal{C}(\mathcal{T}[J, \emptyset]) = \emptyset$.

Example 36 Consider Figure 5.4, where mandatory driving directions point towards a No Entry sign. The input I is given by the following parts.

$$\begin{aligned} I_{u_3} &= \{s(\mathit{no\text{-}entry}, u_3)\} \\ I_{v_2} &= \{s(\mathit{mand\text{-}turn}(\mathit{left}), v_2)\} \\ I_{z_1} &= \{s(\mathit{mand\text{-}turn}(\mathit{straight}), z_1)\} \end{aligned}$$

The set of conflicts $\mathcal{C}(\mathcal{T})$ comprises the following sets:

$$\begin{aligned} C_{v_2} &= \{c(\mathit{overlap}(\ell_s, \mathit{ban}, \mathit{nec}), v_2), c(\mathit{no\text{-}way\text{-}out}(\ell_s), v_2)\} \\ C_{z_1} &= \{c(\mathit{overlap}(\ell_s, \mathit{ban}, \mathit{nec}), z_1), c(\mathit{no\text{-}way\text{-}out}(\ell_s), z_1)\} \end{aligned}$$

The $\subseteq$-minimal diagnoses for C_{v_2} and C_{z_1} are $I_{u_3} \cup I_{v_2}$ and $I_{u_3} \cup I_{z_1}$, respectively. The only diagnosis for $C = \mathcal{C}(\mathcal{T})$ is I, i.e., $\mathcal{D}_{\mathcal{T}}(C) = \{I\}$. Consequently, each non-empty $J \subseteq I$ is a hitting set for $\mathcal{D}_{\mathcal{T}}(C)$. The update $(I_{u_3}, \emptyset)$ to delete only the No Entry sign is a minimal repair for C, but for $J = I_{v_2}$ (respectively $J = I_{z_1}$), $(J, \emptyset)$ is not a repair for C, since $C \cap \mathcal{C}(\mathcal{T}[J, \emptyset])$ equals C_{z_1} (respectively C_{v_2}). ■

In order to get a delete-only repair for a set of conflicts C from its diagnoses, we need to ensure that no conflict in C can be derived in the updated traffic regulation problem. To this end, we observe the following special case of Proposition 8 for single conflicts.

Corollary 9 *Let $c \in \mathcal{C}(\mathcal{T})$ be a single conflict. If $J \subseteq I$ is a hitting set for $\mathcal{D}_{\mathcal{T}}(c)$, then $\{c\} \not\subseteq \mathcal{C}(\mathcal{T}[J, \emptyset])$, i.e., $(J, \emptyset)$ is a repair for $\{c\}$.*

Consequently, given a set C of conflicts, if $J \subseteq I$ meets all diagnoses of all conflicts in C, we get a delete-only repair $(J, \emptyset)$ for C.

Corollary 10 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts. If $J \subseteq I$ is a hitting set for $\mathcal{D}_{\mathcal{T}}(c)$ for each $c \in C$, then $C \cap \mathcal{C}(\mathcal{T}[J, \emptyset]) = \emptyset$, i.e., $(J, \emptyset)$ is a repair for C.*

Proof. Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts and let $J \subseteq I$ be a hitting set for $\mathcal{D}_{\mathcal{T}}(c)$ for each $c \in C$. By Corollary 9, $\{c\} \cap \mathcal{C}(\mathcal{T}[J, \emptyset]) = \emptyset$ for each $c \in C$. Thus, $C \cap \mathcal{C}(\mathcal{T}[J, \emptyset]) = \emptyset$, and therefore, $(J, \emptyset)$ is a repair for C. □

We will now emphasize two special cases of Corollary 9, which will be illustrated below. For a collection $\mathcal{S}$ of sets, $\bigcap \mathcal{S} = \bigcap_{X \in \mathcal{S}} X$ denotes its intersection. The respective set differences, $\Delta\mathcal{S} = \{X \setminus \bigcap \mathcal{S} \mid X \in \mathcal{S}\}$, are called the *non-shared parts (of $\mathcal{S}$)*. For instance, if $\mathcal{S} = \{\{a, b\}, \{b, c\}, \{b\}\}$, then $\bigcap \mathcal{S} = \{b\}$ and $\Delta\mathcal{S} = \{\{a\}, \{c\}, \emptyset\}$. Note that a non-empty $\bigcap \mathcal{S}$ is a special case of a hitting set for $\mathcal{S}$.

Corollary 11 *Let $c \in \mathcal{C}(\mathcal{T})$ and $J \subseteq I$ be non-empty. Then $(J, \emptyset)$ is a repair for $\{c\}$, (i) if $J \supseteq \bigcap \mathcal{D}_{\mathcal{T}}(c)$; or (ii) if $J \subseteq I \setminus \bigcap \mathcal{D}_{\mathcal{T}}(c)$ is a hitting set for $\Delta\mathcal{D}_{\mathcal{T}}(c)$.*

By this corollary, we can obtain different intuitive classes of delete-only repairs by diagnoses for single conflicts, as we show in the next example. In particular, given a conflict $c \in \mathcal{C}(\mathcal{T})$, by item *(i)* we get a minimal repair $(\{j\}, \emptyset)$ for each $j \in \bigcap \mathcal{D}_{\mathcal{T}}(c)$.

Example 37 Figure 5.5 shows a traffic regulation problem $\mathcal{T}$ which adds to the former loop example three additional No Entry signs at nodes a_1, d_5 and d_7, and another Mandatory Left Turn at a_4. We partition the input I as follows:

$$\begin{aligned}
I_{a_{14}} &= \{s(\textit{no-entry}, a_1), s(\textit{mand-turn}(\textit{left}), a_4)\} \\
I^{3ml} &= \{s(\textit{mand-turn}(\textit{left}), \alpha) \mid \alpha \in \{a_6, b_8, c_2\}\} \\
I^{ml}_{d_4} &= \{s(\textit{mand-turn}(\textit{left}), d_4)\} \\
I^{ne}_{d_5} &= \{s(\textit{no-entry}, d_5)\} \\
I^{ne}_{d_7} &= \{s(\textit{no-entry}, d_7)\}
\end{aligned}$$

This traffic regulation problem has conflicts $\mathcal{C}(\mathcal{T}) = C_a \cup C_\ell$, where

$$\begin{aligned}
C_a &= \{c(\textit{no-way-out}(\ell_s), a_4), c(\textit{overlap}(\ell_s, \textit{ban}, \textit{nec}), a_4)\}, \text{ and} \\
C_\ell &= \{c(\textit{no-way-out}(\ell_s), \alpha) \mid \alpha \in \{a_6, b_8, c_2, d_4\}\}.
\end{aligned}$$

For each $c \in C_a$, the diagnoses for $\{c\}$ are

$$\mathcal{D}_{\mathcal{T}}(c) = \{I_{a_{14}} \cup X \mid X \subseteq I \setminus I_{a_{14}}\}.$$

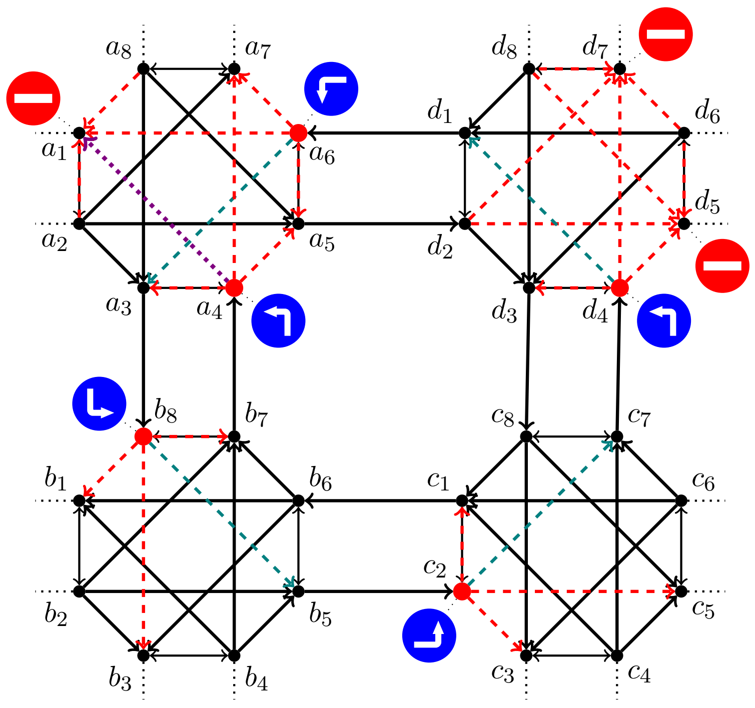


Figure 5.5: Scenario with independent conflicts and repair variants

In particular, $\bigcap \mathcal{D}_{\mathcal{T}}(c) = I_{a_{14}}$ and thus $I_{a_{14}}$ is the unique minimal diagnosis for each of these conflicts. Since each set of respective diagnoses $\mathcal{D}_{\mathcal{T}}(c)$ is convex, we have

$$ctx(\{c(\textit{no-way-out}(\ell_s), a_4)\}) = ctx(\{c(\textit{overlap}(\ell_s, \textit{ban}, \textit{nec}), a_4)\}) = ctx(C_a) = I_{a_{14}}.$$

By Corollary 11, we get two minimal repairs for these $\{c\} \subseteq C_a$: $(\{s(\textit{no-entry}, a_1)\}, \emptyset)$ and $(\{s(\textit{mand-turn}(\textit{left}), a_4)\}, \emptyset)$.

Complementary, for each non-empty $C \subseteq C_\ell$ it holds that $I_{a_{14}}$ is independent of C, since for each $c \in C_\ell$, the diagnoses for $\{c\}$ are given by

$$\mathcal{D}_{\mathcal{T}}(c) = \bigcup_{1 \leq k \leq 5} \{D_k \cup Y \mid Y \subseteq I_{a_{14}}\}, \text{ where}$$

$$\begin{array}{lcl}
D_1 & = & I^{3ml} \cup I^{ml}_{d_4} \cup I^{ne}_{d_5} \cup I^{ne}_{d_7} \\
D_2 & = & I^{3ml} \cup I^{ml}_{d_4} \cup I^{ne}_{d_5} \\
D_3 & = & I^{3ml} \cup I^{ml}_{d_4} \cup I^{ne}_{d_7} \\
D_4 & = & I^{3ml} \cup I^{ml}_{d_4} \\
D_5 & = & I^{3ml} \cup I^{ne}_{d_5} \cup I^{ne}_{d_7}.
\end{array}$$

Each of these conflict sets $C \subseteq C_\ell$ has two $\subseteq$-minimal diagnoses, namely D_4 and D_5. Since $\mathcal{D}_{\mathcal{T}}(C)$ is convex, we get $ctx(C) = I^{3ml} \cup I^{ml}_{d_4} \cup I^{ne}_{d_5} \cup I^{ne}_{d_7} = I \setminus I_{a_{14}}$. Thus, C_a

and C_ℓ are maximal independent sets of conflicts. The intersection of the diagnoses for each conflict in C_ℓ is I^{3ml}, i.e., $\bigcap \mathcal{D}_\mathcal{T}(c) = I^{3ml}$ for each $c \in C_\ell$. Hence, by Corollary 11, we get a repair $(J, \emptyset)$ for each non-empty $J \subseteq I^{3ml}$. That is to say, deleting at least one Mandatory Left Turn sign at nodes a_6, b_8 and c_2 gives a repair for C_ℓ, and if we delete exactly one, i.e. $|J| = 1$, we get a minimal repair.

If we do not delete any of these three signs in $I^{3ml} = \bigcap \mathcal{D}_\mathcal{T}(C_\ell)$, we certainly have to delete the fourth Mandatory Left Turn at d_4. By removing only this sign, the loop would still exist due to the No Entry signs at d_5 and d_7, from which at least one has to be deleted. These observations are expressed by the $\subseteq$-minimality of the diagnoses D_4 and D_5. Corollary 11 also captures these plausible repairs. In contrast to C_a, the set of non-shared parts of diagnoses for C_ℓ does not contain the empty set, i.e. $\emptyset \notin \Delta\mathcal{D}_\mathcal{T}(C_\ell)$. For each $c \in C_\ell$, the non-shared parts of $\mathcal{D}_\mathcal{T}(c)$ are

$$\Delta\mathcal{D}_\mathcal{T}(c) = \bigcup_{1 \leq k \leq 5} \{D_k^\Delta \cup Y \mid Y \subseteq I_{a_{14}}\}, \text{ where}$$

$$\begin{aligned}
D_1^\Delta &= I_{d_4}^{ml} \cup I_{d_5}^{ne} \cup I_{d_7}^{ne} \\
D_2^\Delta &= I_{d_4}^{ml} \cup I_{d_5}^{ne} \\
D_3^\Delta &= I_{d_4}^{ml} \cup I_{d_7}^{ne} \\
D_4^\Delta &= I_{d_4}^{ml} \\
D_5^\Delta &= I_{d_5}^{ne} \cup I_{d_7}^{ne}.
\end{aligned}$$

We can find sets $J \subseteq I \setminus \bigcap \mathcal{D}_\mathcal{T}(c)$ that are hitting sets for $\Delta\mathcal{D}_\mathcal{T}(c)$. By picking $J = I_{d_4}^{ml}$, we have a hitting set for D_k^Δ, where $1 \leq k \leq 4$. In order to meet also D_5^Δ, we must add $I_{d_5}^{ne}$ or $I_{d_7}^{ne}$ (or both). We get the additional repairs $(I_{d_4}^{ml} \cup I_{d_5}^{ne}, \emptyset)$ and $(I_{d_4}^{ml} \cup I_{d_7}^{ne}, \emptyset)$ for all $c \in C_\ell$ (and those delete-only repairs with according supersets).

For $k \in \{a, \ell\}$, and $c \in C_k$, a repair $(R_k^c, \emptyset)$ for $\{c\}$ is also a repair for any $C'_k \subseteq C_k$. If $(R_k, \emptyset)$ is a repair for the respective set of conflicts C_k, then $(R_a \cup R_\ell, \emptyset)$ is also a repair for $\mathcal{T}$. ∎

In the previous example, we could repair the traffic regulation problem by combining local repairs for independent sets of conflicts. In general, however, local repairs may have *side effects*, i.e., introduce new conflicts.

Example 38 Figure 5.6 shows the T-junction with speed limit signs as seen in the previous chapter. Similar as in Figure 5.4, the Mandatory Left Turn sign at v_2 and the No Entry sign at u_3 express contradictory permissions. The input I of this traffic regulation problem $\mathcal{T}$ is given as the union of the following sets:

$$\begin{aligned}
I_1 &= \{s(\mathit{start}(\mathit{spl}(30)), x_2), s(\mathit{end}(\mathit{spl}(30)), y_2)\}, \\
I_2 &= \{s(\mathit{mand\text{-}turn}(\mathit{left}), v_2), s(\mathit{no\text{-}entry}, u_3)\}.
\end{aligned}$$

The set I_2 is the unique minimal diagnosis (and thus the context) of the traffic regulation problem's conflicts $\mathcal{C}(\mathcal{T}) = \{c(\mathit{no\text{-}way\text{-}out}(\ell_s), v_2), c(\mathit{overlap}(\ell_s, \mathit{ban}, \mathit{nec}), v_2)\}$. For

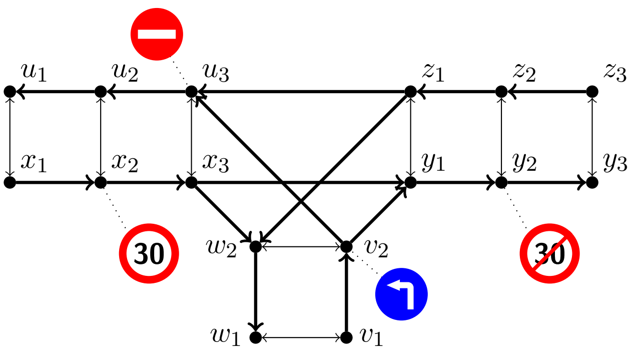


Figure 5.6: Mandatory Left Turn sign pointing in an impossible direction

each $c \in \mathcal{C}(\mathcal{T})$, Corollary 11 suggests two minimal repairs:

$$\begin{aligned} R_1 &= (\{s(\textit{no-entry}, u_3)\}, \emptyset) \\ R_2 &= (\{s(\textit{mand-turn}(\textit{left}), v_2)\}, \emptyset) \end{aligned}$$

Since R_1 does not have side effects it is also a repair for $\mathcal{T}$. However, removing (only) the Mandatory Left Turn from the input introduces new conflicts

$$\{c(\textit{bad-end}(\textit{max-speed}(30)), y_1), c(\textit{cant-end}(\textit{max-speed}(30)), y_2)\},$$

as discussed in Example 25. As a consequence, R_2 not a repair for $\mathcal{T}$. ■

To see why in the last example the update R_2 has side effects, we analyze the specification of the involved conflicts. The according rules (4.30) and (4.31) (page 46) test for the *absence* of an effect; the sign effect $f'(\ell_s, \textit{max-speed}(30), y_1, y_2)$ in this case. Since we remove the Mandatory Left Turn in R_2, this effect is not derived anymore by the effect mapping and therefore both rules fire.

In the examples before, where local repairs did not introduce new conflicts, the involved rules in the conflict specification test only for *positive* occurrences of effects. In principle, however, the removal of signs or measures may also lead to additional effects. Therefore, a detailed analysis under which circumstances local repairs (and merged local repairs) are free of side effects, is a topic on its own.

We return to the original speed limit example to discuss its repairs.

Example 39 To repair the speed limit scenario of Figure 4.11, we have several options, other than deleting both signs. One option would be to move the end sign from node y_2 to y_1, as shown in Figure 5.7, i.e., the repair $(\{s(\textit{end}(\textit{spl}(30)), y_2)\}, \{s(\textit{end}(\textit{spl}(30)), y_1)\})$. Similarly, we could move the start sign from x_2 to y_1, as shown earlier in Figure 4.6. A repair with unit cost 1 is $(\emptyset, \{s(\textit{mand-turn}(\textit{left}), v_2)\})$, i.e., adding a Mandatory Left Turn at node v_2 as presented in the previous example. Likewise, the addition of a No Right Turn at v_2, as depicted in Figure 4.9 is a minimal repair.

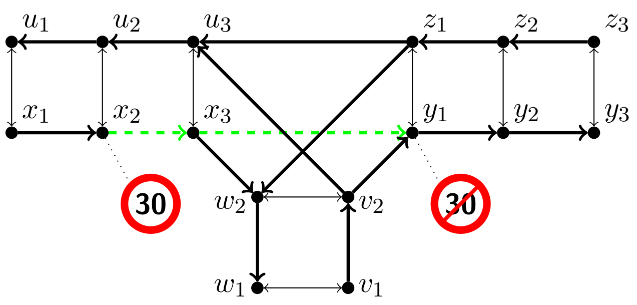


Figure 5.7: Repair variant for Figure 4.3: Moving the end sign from y_2 to y_1

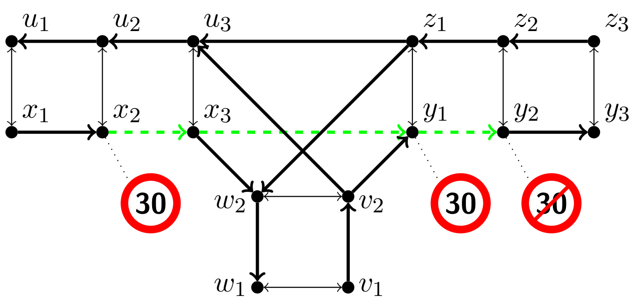


Figure 5.8: Result of the only reasonable repair variant for Figure 4.3, if a speed limit measure information is given from x_2 to y_2

Finally, putting another 30 km/h start sign at y_1, i.e. $(\emptyset, \{s(\mathit{start}(\mathit{spl}(30)), y_1)\})$, is also a minimal repair, leading to the scenario shown in Figure 5.8. If we also consider the according speed limit measure from x_2 to y_2, as in the initial example of Figure 4.3, then only this repair is reasonable. ■

This example suggests that we need a stronger variant of repair in case we are given input of both languages.

5.4 Correspondence

Given a traffic regulation problem which includes both measures and signs, consistency is not a sufficient criterion for data correctness. Additionally, we are interested in the following condition.

Definition 22 (Correspondence) *A set of measures M and a set of signs S* correspond *with respect to an effect mapping P and a street graph G, if they express the same effects, i.e., $\mathcal{F}_G^P(M) = \mathcal{F}_G^P(S)$.*

We naturally get an according decision problem.

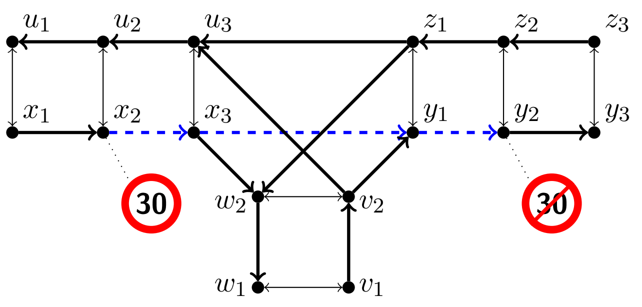


Figure 5.9: Missing repeated start sign for a speed limit measure from x_2 to y_2

Problem:	CORRESPONDENCE (CORR)
Instance:	A traffic regulation problem $\mathcal{T}$ with measures M and signs S
Output:	Decide whether M and S correspond

We call a measure *unannounced*, if no sign expresses one of its effects. Likewise, a sign is called *unjustified*, if some effect expressed by it is not expressed by any measure. We extend these two notions to *unannounced effects*, given by $\mathcal{F}^P_G(M) \setminus \mathcal{F}^P_G(S)$, and *unjustified effects*, given by $\mathcal{F}^P_G(S) \setminus \mathcal{F}^P_G(M)$.

Example 40 Assume we are given the scenario $Sc = (G, M, S)$ of Figure 5.9, where

$$\begin{aligned} M &= \{m(\mathit{spl}(30), x_2, x_3), m(\mathit{spl}(30), x_3, y_1), m(\mathit{spl}(30), y_1, y_2)\}, \text{ and} \\ S &= \{s(\mathit{start}(\mathit{spl}(30)), x_2), s(\mathit{end}(\mathit{spl}(30)), y_2)\}, \end{aligned}$$

and the graph G is given as set of atoms of form $e(T, X, Y)$, representing edges, as usual. As explained before, S is not consistent. However, M is consistent, and consequently, S and M cannot correspond. We have

$$\begin{aligned} \mathcal{F}^P_G(S) \setminus \mathcal{F}^P_G(M) &= \emptyset, \\ \mathcal{F}^P_G(M) \setminus \mathcal{F}^P_G(S) &= \{f(\mathit{max\text{-}speed}(30), y_1, y_2)\}. \end{aligned}$$

We see that there are no unjustified signs, but the *max-speed*(30) effect on edge (y_1, y_2) is unannounced. ■

The correspondence criterion is the key to obtain a suitable repair task for traffic regulation problems that contain both measure and sign data.

5.5 Strict Repair

By requiring correspondence in addition to consistency, we get a strengthened notion of repair, suitable for scenarios (G, M, S), where both M and S are non-empty.

Definition 23 (Strict repair) *A repair (I^-, I^+) for $\mathcal{T}$ is called a* strict repair*, if the set of measures M' and the set of signs S' in the updated traffic regulation problem $\mathcal{T}[I^-, I^+]$ correspond.*

In analogy to Repairs, we define an according reasoning task. Again, we are not interested in all possible strict repairs, but only a relevant subset as defined by a predicate *adq* on strict repairs.

Problem:	Strict Repairs (StrRepairs) (for $\mathcal{T}$ under *adq*)
Instance:	An inconsistent traffic regulation problem $\mathcal{T}$ and an adequacy criterion *adq* for updates (I^-, I^+)
Output:	The set of strict repairs (I^-, I^+) for $\mathcal{T}$ such that $adq(I^-, I^+)$ holds

By StrRepairs($\mathcal{T}$, *adq*) we denote the set of strict repairs for $\mathcal{T}$ under *adq*. We omit a discussion of local strict repairs for sets of conflicts.

Example 41 (cont'd) All repairs for the scenario of Figure 4.11 as listed in Example 39 are also repairs for the scenario of Example 40 (Figure 5.9), but only the addition of the repeating start sign at y_1, i.e. $(\emptyset, \{s(start(spl(30)), y_1)\})$, is a strict repair, which is the only strict repair for $\mathcal{T}$ under *ucost*.

Moving the start sign or the end sign would leave some measures unannounced. In order to get a strict repair from these variants, the according measure would need to be deleted as well, leading to unit cost 2. Adding the No Right Turn at v_2 would introduce an unjustified *ban* effect on edge (v_2, y_1). Likewise, putting a Mandatory Left Turn at v_2 would ban traffic along the edges (v_2, u_3) and (v_2, w_2) without justification by a measure. To obtain a strict repair from these sign additions, at least one measure needs to be added, causing a non-optimal unit cost of at least 2. ■

For Strict Repairs, a good choice for the minimality (adequacy) criterion is context dependent. Since a strict repair potentially deletes and adds measures and signs, and moreover, of different respective types, there are many ways to define a preference relation over strict repairs. So far, we have only considered unit cost, i.e., a simple count of changes to the input regardless of language or type. In practice, there is no adequacy condition which equally applies to every situation.

For instance, consider the case that measures were imported from an unreliable data source, or via some heuristic algorithm. Then we might prefer to change measures before adding or deleting traffic signs. However, if measures are collected manually, the opposite preference is plausible. Traffic measures reflect the legal intention, are supposed to be deliberately enacted and undergo official authorization. Traffic signs, on the other hand, may be sloppily placed or forgotten to be removed. If authorized government officials manually create traffic measure data, we will prefer changes of traffic signs.

Example 42 Figure 5.10 depicts a 30 km/h speed limit measure from x_2 to y_2 plus its start sign at x_2, and a 40 km/h measure from y_1 to y_3 plus its end sign at y_3.

$$\begin{aligned} M &= \{m(spl(30), x_2, x_3), m(spl(30), x_3, y_1), m(spl(30), y_1, y_2), \\ &\qquad m(spl(40), y_1, y_2), m(spl(40), y_2, y_3)\} \\ S &= \{s(start(spl(30)), x_2), s(end(spl(40)), y_3)\} \end{aligned}$$

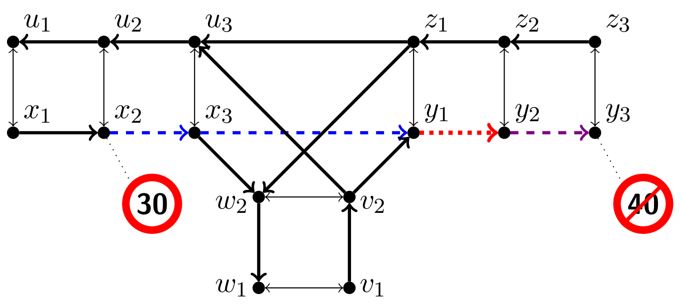


Figure 5.10: Overlapping speed limit measures, 30 km/h and 40 km/h, on edge (y_1, y_2)

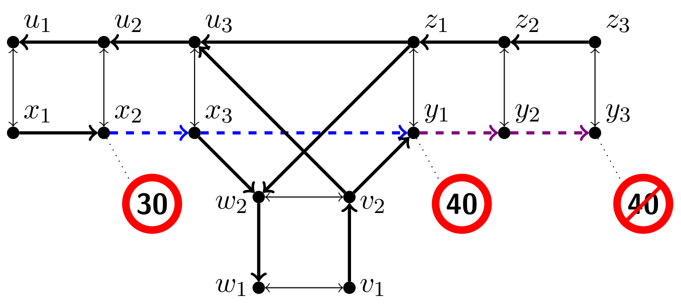


Figure 5.11: Strict repair for Figure 5.10 in favour of the 40 km/h restriction

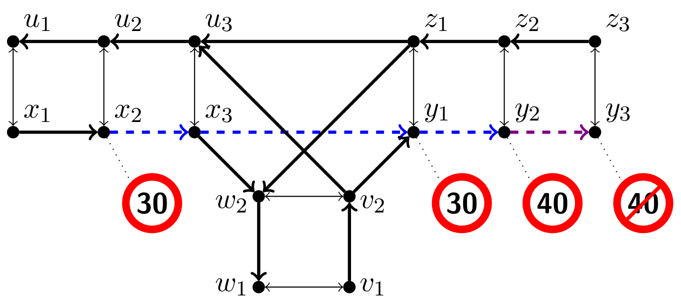


Figure 5.12: Strict repair for Figure 5.10 in favour of the 30 km/h restriction

We have have three conflicts with respective unique minimal diagnoses:

k	conflict singleton C_k	$\subseteq$-minimal diagnosis J_k
1	$\{c(\mathit{bad\text{-}end}(\mathit{max\text{-}speed}(30)), y_1)\}$	$\{s(\mathit{start}(\mathit{spl}(30)), x_2)\}$
2	$\{c(\mathit{cant\text{-}end}(\mathit{max\text{-}speed}(40)), y_3)\}$	$\{s(\mathit{end}(\mathit{spl}(40)), y_3)\}$
3	$\{c(\mathit{overlap}(\ell_m, \mathit{max\text{-}speed}(30), \mathit{max\text{-}speed}(40)), y_1)\}$	$\{m(\mathit{spl}(30), y_1, y_2),$ $m(\mathit{spl}(40), y_1, y_2)\}$

Three effects are unannounced:

$$\begin{aligned}\mathcal{F}^P_G(M) \setminus \mathcal{F}^P_G(S) \;=\; & \{f(\mathit{max\text{-}speed}(30), y_1, y_2), f(\mathit{max\text{-}speed}(40), y_1, y_2),\\ & f(\mathit{max\text{-}speed}(40), y_2, y_3)\}\end{aligned}$$

Let us first examine only the traffic sign information. Again, there is a bad end of a 30 km/h restriction after the junction. According to signs, a speed limit of 40 km/h is supposed to end along this lane. Hence, we can repair the sign information by adding a 40 km/h start sign at y_1.

Concerning measures, we have contradictory information on the edge (y_1, y_2). We need to delete one of the atomic measures $\{m(\mathit{spl}(K), y_1, y_2) \mid K \in \{30, 40\}\}$ to establish consistency of the measure information. Thus, we get two minimal repairs

$$(\{m(\mathit{spl}(K), y_1, y_2)\}, \{s(\mathit{start}(\mathit{spl}(40)), y_1)\}),$$

where $K \in \{30, 40\}$. However, only the repair with $K = 30$ is also a strict repair. The updated traffic regulation problem due to this repair is shown in Figure 5.11. In order to get a strict repair where the 40 km/h measure on (y_1, y_2) is deleted, we need to add two signs as shown in Figure 5.12: in addition to the 40 km/h start sign at y_1, a repeated 30 km/h start at y_1 is required. Hence, this variant, which is equally plausible, is not minimal with respect to unit cost, since one more sign needs to be added. ■

We conclude that it is desirable to have flexibility in the adequacy condition applied to (strict) repairs. In the demonstrated case, we might want to count only modifications of measures, regardless of how many traffic signs need to be changed as a consequence of minimal measure changes.

Two specific constraints on strict repairs are discussed next, where only measures or signs can be updated, but not both.

5.6 Adjustment and Generation

Motivated by data import settings and data correction of only one language, we will investigate two practically relevant special cases of strict repairs. We say a repair (I^-, I^+) is *X-based*, if $I^- \cup I^+ \subseteq X$.

Definition 24 (Adjustment) *Let $\mathcal{T}$ be a traffic regulation problem with measures M and signs S. If M is consistent, a* sign adjustment *for $\mathcal{T}$ is an S_G-based strict repair (S^-, S^+) for $\mathcal{T}$. Similarly, if S is consistent, a* measure adjustment *for $\mathcal{T}$ is an M_G-based strict repair (M^-, M^+) for $\mathcal{T}$.*

That is, whenever only signs (respectively measures) are changed, we speak of a sign (respectively measure) adjustment. Adjustment is useful if one kind of information shall serve as basis for the repair of the other.

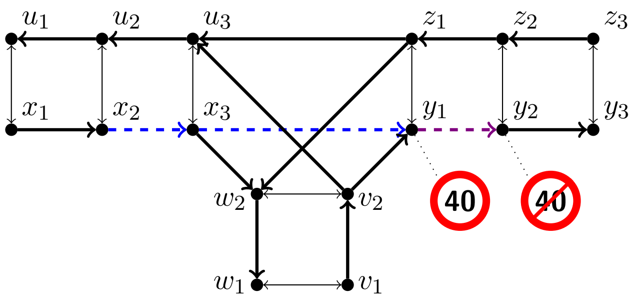


Figure 5.13: Consecutive 30 km/h and 40 km/h speed limit measures

Problem:	Sign Adjustment (S-Adj) (for $\mathcal{T}$ under *adq*)
Instance:	An inconsistent traffic regulation problem $\mathcal{T}$ with consistent M and an adequacy criterion *adq* for updates (I^-, I^+)
Output:	The set of sign adjustments (I^-, I^+) for $\mathcal{T}$ such that $adq(I^-, I^+)$ holds

The task Measure Adjustment (M-Adj) is similarly defined.

Example 43 Consider Figure 5.13, in which a 30 km/h speed limit measure ranges from x_2 to y_1, immediately followed by a 40 km/h speed limit measure until node y_2. While the latter is correctly announced and the 40 km/h start sign is a proper (implicit) end for the previous measure, the start sign at x_2 is missing. The sets M and S are consistent but do not correspond, since

$$\mathcal{F}_G^P(M) \setminus \mathcal{F}_G^P(S) = \{f(\mathit{max\text{-}speed}(30), x_2, x_3), f(\mathit{max\text{-}speed}(30), x_3, y_1)\}.$$

A minimal strict repair for this scenario is $(\emptyset, \{s(\mathit{start}(\mathit{spl}(30)), x_2)\})$, which is also a sign adjustment. If we know that the traffic sign data is complete, the proper task is Measure Adjustment, which is unique: $(\emptyset, \{m(\mathit{spl}(30), x_2, x_3), m(\mathit{spl}(30), x_3, y_1)\})$. ■

Another relevant restriction related to data import is the generation of measures or signs, given consistent data of the other kind.

Definition 25 (Generation) *Let $\mathcal{T}$ be a traffic regulation problem with measures M and signs S. If M is consistent and $S = \emptyset$, an S_G-based strict repair $(\emptyset, S^+)$ is called a* sign generation *for $\mathcal{T}$. Similarly, if S is consistent and $M = \emptyset$, an M_G-based strict repair $(\emptyset, M^+)$ for $\mathcal{T}$ is called a* measure generation *for $\mathcal{T}$.*

Adjustment requires one of the two languages in the scenario, measures or signs, to be consistent and furthermore constrains the strict repair to suggest updates only for the other language. Generation imposes the additional restriction on the scenario to comprise consistent data of only one language.

Problem:	Sign Generation (S-Gen) (for $\mathcal{T}$ under *adq*)
Instance:	A consistent traffic regulation problem $\mathcal{T}$ with empty S and an adequacy criterion *adq* for updates $(\emptyset, S^+)$
Output:	The set of sign generations $(\emptyset, S^+)$ for $\mathcal{T}$ such that $adq(\emptyset, S^+)$ holds

Again, the dual task MEASURE GENERATION (M-GEN) is similarly defined.

Example 44 The 30 km/h speed limit scenario $(G, \emptyset, S)$ of Figure 5.8 has three unjustified effects:

$$\begin{aligned}\mathcal{F}_G^P(S) \setminus \mathcal{F}_G^P(M) \;=\; &\{f(\mathit{max\text{-}speed}(30), x_2, x_3), f(\mathit{max\text{-}speed}(30), x_3, y_1),\\ &f(\mathit{max\text{-}speed}(30), y_1, y_2)\}\end{aligned}$$

There are two minimal strict repairs: Either all of these three signs are deleted, or three atomic *spl*(30) measures along these edges are added. To force the latter solution, we can apply MEASURE GENERATION. In a dual scenario $(G, M, \emptyset)$, comprising only these three speed limit measures, we can similarly use SIGN GENERATION to prevent the measure labels from being removed. ■

Summary. In this chapter we have defined major goals in inconsistency management for traffic regulations based on explicit conflicts. We have extensively studied diagnoses and repairs and investigated independence between conflicts and subsets of the input, and dually, unique contexts due to which inconsistencies arise. We gave a full characterization of contexts by means of diagnoses. For repairs, we have studied several practically relevant variants and pointed out some difficulties arising from the problem domain. In particular, we illustrated some challenges for an implementation regarding its maintainability and flexibility.

If the programming language or methodology allows to modularly add (or activate and deactivate) constraints, we need no separate implementation of adjustment and generation (or other such restrictions), given one exists for the strict repair task. Moreover, if correspondence checking (CORR) is reduced to CONSEVAL, one can uniformly approach repairs and strict repairs. In Chapter 7 we will take these conceptual ideas even further and present an implementation with Answer Set Programming that realizes all presented reasoning tasks as slight variations on top of a uniform approach. Before that, we will now investigate the computational complexity of decision problems associated with our reasoning tasks.

CHAPTER 6

Computational Complexity

In this chapter, we analyze the computational complexity of decision problems associated with the reasoning tasks of Chapter 5, by presenting in more detail the results established in [3]. In particular, we consider for a given traffic regulation problem $\mathcal{T} = (\Pi, Sc)$, with traffic regulation $\Pi = (P, Sp)$ and scenario $Sc = (G, M, S)$, the following decision problems:

- Cons: Decide whether $\mathcal{T}$ is consistent, i.e., $\mathcal{C}(\mathcal{T}) = \mathcal{C}_G^{P,Sp}(I) = \emptyset$.
- UMinDiag: Decide, given a set $C \subseteq \mathcal{C}(\mathcal{T})$ of conflicts, whether C has a unique $\subseteq$-minimal diagnosis, i.e., a single minimal $J \subseteq I$ such that $C \subseteq \mathcal{C}_G^{P,Sp}(J)$.
- Corr: Decide whether M and S correspond, i.e., $\mathcal{F}_G^P(M) = \mathcal{F}_G^P(S)$.
- Repair: Decide, given that $\mathcal{T}$ is inconsistent, whether some admissible repair exists, i.e., some $I^-, I^+ \subseteq I_G$ such that $\mathcal{C}_G^{P,Sp}((I \setminus I^-) \cup I^+) = \emptyset$ and a polynomial-time admissibility predicate $\mathcal{A}(I^-, I^+)$ holds.

We consider different logics $\mathcal{L}$ for the traffic regulation Π and assume that the same logic is used for the effect mapping P and the conflict specification Sp over it. In particular, we consider

(i) first-order logic under domain closure (FOL+DCA), i.e., an axiom $\forall x. \bigvee_{i=1}^{n}(x = c_i)$, where $c_1, \ldots, c_n$ is the finite set of constant symbols; and

(ii) Answer Set Programs.

Here, we consider different syntactic classes of function-free programs. Function symbols were used in the previous chapters for convenient presentation of P and Sp, but are not essential for the domain.

Logic $\mathcal{L}$	general case	BPA
FOL+DCA	co-NEXP	PSPACE
$\text{ASP}^{\neg_s}$	EXP	P^{NP}
$\text{ASP}^{\neg}$	co-NEXP	Π_2^p
$\text{ASP}^{\vee,\neg}$	co-$\mathsf{NEXP}^{\mathsf{NP}}$	Π_3^p

Table 6.1: Complexity classes for Impl in different logics: completeness results for the general case and with bounded predicate arities (BPA)

The classes of interest are the following.

- *Stratified programs* ($\text{ASP}^{\neg_s}$)
- *Normal programs* ($\text{ASP}^{\neg}$)
- *Disjunctive programs* ($\text{ASP}^{\vee,\neg}$)

We recall that disjunctive programs are sets of rules of form

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

where $k, m, n \geq 0$, and all a_i and b_j are literals. Normal programs disallow disjunctive heads, i.e. $k = 1$, but still allow arbitrary negation. Stratified programs constrain normal programs by allowing only stratified negation, i.e., no recursion through negation.

To obtain the complexity results for the listed decision problems, we make use of results for the entailment problem, which is reviewed next.

6.1 Entailment

The entailment problem is defined as follows.

Problem:	Entailment (Impl)
Instance:	A set T of formulas (or rules) and an atom α in a fixed logic $\mathcal{L}$
Output:	Decide whether $T \models \alpha$ holds

For the ASP cases, we are interested in cautious consequence, i.e., whether α is true in all answer sets of T. Table 6.1 summarizes complexity results for Impl of first-order theories under domain closure (FOL+DCA), respectively ASP, for the general case, and when the predicate arities are bounded by some constant.

For the FOL+DCA case, a countermodel of $T \models \alpha$ (of at most exponential size) can be guessed and verified in polynomial space (in the size of T and α). This puts the problem into co-NEXP. Hardness follows, e.g., from the complexity of satisfiability of the Bernays-Schönfinkel fragment of FOL [31]. Under the assumption of bounded predicate arities (BPA), the model guess has polynomial size, and thus the whole countermodel check is feasible in polynomial space. PSPACE-hardness is inherited from the evaluation of a given FOL formula over a finite structure, which is PSPACE-hard already for monadic predicates. This can be shown by encodings of quantified boolean formulas (QBFs). For the ASP^X languages mentioned above, we refer to [10, 13].

Throughout this chapter, we will refer to the complexity classes for Impl by

$$C_{\text{Impl}} = \{\mathsf{EXP}, \mathsf{PSPACE}, \mathsf{P}^{\mathsf{NP}}, \text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}\,. \tag{6.1}$$

6.2 Consistency

We recall the definitions how effects are established by means of an effect mapping P and how conflicts are derived by the conflict specification Sp. By employing the closed world operator, we will always get a unique model. We assume that both P and Sp are well-defined in the sense that a model always exists.

$$\begin{array}{lllll} \mathcal{F}_G^P(I) & = & Cn_G(P, I, F_G) & = & \{f(t,v,w) \in F_G \mid P \cup \overline{G} \cup \overline{I} \models f(t,v,w)\} \\ \mathcal{C}_G^{P,Sp}(I) & = & Cn_G(Sp, \mathcal{F}_G^P(I), C_G) & = & \{c(t,v) \in C_G \mid Sp \cup \overline{G} \cup \overline{\mathcal{F}_G^P(I)} \models c(t,v)\} \end{array}$$

We define Consistency as the problem to decide whether conflicts exist.

Problem:	Consistency (Cons)
Instance:	A traffic regulation problem $\mathcal{T}$
Output:	Decide whether $\mathcal{C}(\mathcal{T})$ is empty

Proposition 12 *For a given logic, let $O \in C_{\text{Impl}}$ be the complexity of* Impl. *Then, the problem* Cons *is in $\mathsf{P}^O_{\|[2]}$, i.e., the class of polynomial time computations with 2 rounds of parallel oracle calls for* Impl.

Proof. To decide Cons, we must determine whether some conflict atom $c(t,v) \in C_G$ is derivable on a street graph $G = (V, E)$. Recall that C is the fixed set of conflict types. Thus, there are at most $|C_G| = |V| \cdot |\mathsf{C}|$ such atoms to test. Likewise, we have polynomially many possible effects, since $|F_G| = |E| \cdot |\mathsf{F}|$, where F are the effect types. For each $f(t,v,w) \in F_G$, we can test whether $f(t,v,w) \in \mathcal{F}_G^P(I)$ with an oracle for Impl, and then likewise decide for each $c(t,v) \in C_G$, whether $c(t,v) \in \mathcal{C}_G^{P,Sp}(I)$. This is a polynomial time computation with two consecutive rounds of parallel evaluation of oracle queries with respective Impl complexity O. This puts the problem Cons in the complexity classes $\mathsf{P}^O_{\|[2]}$. □

Lemma 13 *For complexity classes $O \in \{\mathsf{EXP}, \mathsf{PSPACE}, \mathsf{P}^{\mathsf{NP}}\}$, it holds that $\mathsf{P}^O_{\|[2]} = O$.*

Proof (Sketch). Oracle calls in the computation of an $\mathsf{P}^O_{\|[2]}$ machine can be simulated by the machine for $O \in \{\mathsf{EXP}, \mathsf{PSPACE}\}$, so no oracle is needed; for $O = \mathsf{P}^{\mathsf{NP}}$, they can be calculated with an NP oracle in polynomial time. □

The next lemma shows that, for the remaining classes of Impl, multiple consecutive parallel oracle queries can be reduced to one.

Lemma 14 *For complexity classes $O \in \{\mathsf{NEXP}, \mathsf{NEXP}^{\mathsf{NP}}\} \cup \{\Sigma_i^p \mid i \geq 1\}$, and constant k, it holds that $\mathsf{P}^{O}_{\|[k]} = \mathsf{P}^{O}_{\|[1]} = \mathsf{P}^{O}_{\|}$.*

Proof (Sketch). By definition $\mathsf{P}^{O}_{\|[1]} = \mathsf{P}^{O}_{\|}$, and thus it remains to consider $\mathsf{P}^{O}_{\|[k]} = \mathsf{P}^{O}_{\|[1]}$. (For $O = \Sigma_i^p$, this was shown by Wagner [46].) We prove the result by induction on k. For the induction step, we assume that $\mathsf{P}^{O}_{\|[k-1]} = \mathsf{P}^{O}_{\|[1]}$ holds, i.e., $k-1$ consecutive rounds of parallel oracle queries can be compiled into into 1 round. We observe that every polynomial computation with k rounds of queries can be seen as one with $k-1$ rounds, which produces for a fixed k in polynomial time an input for a computation with one round of queries. Thus, we have 2 rounds of queries which can be reduced to 1 by the result for $k = 2$.

To see that $\mathsf{P}^{O}_{\|[2]} = \mathsf{P}^{O}_{\|[1]}$ holds, suppose we are given a Turing machine M solving a problem A that works on input I in polynomial time with two consecutive rounds of parallel queries. For this $\mathsf{P}^{O}_{\|[2]}$-machine M, we construct a Turing machine M' that works as follows. M' constructs (in polynomial time) the queries $Q_1^1, \dots, Q_{m_1}^1$ of the first round of m_1 parallel queries that M makes on input I. Let $b = b_1, \dots, b_{m_1}$ be the query result of the first round, where each b_i is the answer of the query Q_i^1. Furthermore, suppose that $Q_1^2(b), \dots, Q_{m_2}^2(b)$ are the queries that M would construct for the second round of queries, given b. Then, M' constructs queries $Q_{1,1}^2, \dots, Q_{i,j}^2, \dots, Q_{m_2,m_1}^2$, where $Q_{i,j}^2$ is the following query: "Are there j queries $Q_{h_1}^1, \dots, Q_{h_j}^1$ among $Q_1^1, \dots, Q_{m_1}^1$ such that the query $Q_i^2(b)$, where $b = b_1, \dots, b_{m_1}$ is the answer string such that $b_\ell =$ 'yes' iff $\ell \in \{h_1, \dots, h_j\}$, and all queries $Q_{h_1}^1, \dots, Q_{h_j}^1$ have the answer 'yes'."

Proper queries $Q_{h_1}^1, \dots, Q_{h_j}^1$, along with "proofs" for a 'yes' answer, can be nondeterministically generated and then be checked in polynomial time, using an available oracle in case of $O = \mathsf{NEXP}^{\mathsf{NP}}$ or $O = \Sigma_i^p, i > 1$. Thus, for the considered complexity classes O, answering query $Q_{i,j}^2$ is itself in class O. After constructing and evaluating all queries $Q_i^1, Q_{i,j}^2$ in one round, M' counts the number n_1 of 'yes' answers among the queries Q_i^1, i.e., the correct answer string b of M of the first round of queries. It then constructs the answer string $b' = b'_1, \dots, b'_{m_2}$ for the second round of queries by letting b'_i be the answer to Q_{i,n_1}^2. After that, M' simulates M and uses the precalculated answer strings for the first and second round of queries.

Overall, M' makes a polynomial number of queries in one round, and runs in polynomial time in the size of the input I. Thus, the problem A is solvable in $\mathsf{P}^{O}_{\|[1]}$. Note that the degree of the polynomial bounding the number of oracle queries and the running time of M' grows, and only for a constant number of times this reduction of the number of rounds stay within overall polynomial time. □

We recall that for every class C, $\mathsf{P}^{C} = \mathsf{P}^{\text{co-}C}$, where co-$C$ is the complementary class of C, and thus $\mathsf{P}^{\Sigma_i^p} = \mathsf{P}^{\Pi_i^p}$, $\mathsf{P}^{\text{co-}\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$ and $\mathsf{P}^{\text{co-}\mathsf{NEXP}^{\mathsf{NP}}} = \mathsf{P}^{\mathsf{NEXP}^{\mathsf{NP}}}$. Hence, for the classes $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$ of considered Impl complexities, we obtain by Lemma 14, $\mathsf{P}^{O}_{\|[2]} = \mathsf{P}^{O}_{\|}$.

To derive $\mathsf{P}_{\|}^{O}$-hardness of Cons, we reduce the following $\mathsf{P}_{\|}^{O}$-complete problem Even to Cons.

Problem:	Even
Instance:	Complexity class O and instances $I_0, \ldots, I_m$ ($m \geq 0$) of a fixed O-complete decision problem D
Output:	Decide whether the number of yes-instances is even

Lemma 15 *Problem* Even *is* $\mathsf{P}_{\|}^{O}$*-complete, for* $O \in \{\mathsf{NP}^{\Sigma_i^p}, \mathsf{NEXP}^{\Sigma_i^p} \mid i \geq 0\}$*. Furthermore, the problem is* $\mathsf{P}_{\|}^{O}$*-hard even if the instances* $I_0, \ldots, I_m$ *are ordered such that* I_i *is a no-instance only if* I_{i+1} *is a no instance, for all* $i \geq 0$*, i.e., all yes-instances precede all no-instances.*

Proof (Sketch). For $O = \mathsf{NP}^{\Sigma_0^p} = \mathsf{NP}$, this has been shown by Wagner [46], and using techniques by Krentel [29], we can give a short proof sketch here for all classes O that we consider.

For the membership part, we observe that all queries I_i can be evaluated in parallel and the result is then easily calculated from the answers.

For the hardness part, we exploit that $\mathsf{P}_{\|}^{O} = P^O[\log n]$, i.e., the class where at most an (order of) logarithmically many oracle queries in the size $|I|$ of the input can be made. (For "$\supseteq$" the proof is easy, as only polynomially many queries exist in all possible runs of a $P^O[\log n]$ machine in input I; for the "$\subseteq$" direction, one uses standard census techniques (cf. [46]) to compute in binary search the number of the parallel queries that have a 'yes' answer, and uses then this information to decide the problem with one further call to an oracle in O).

So let M be an $P^O[\log n]$ machine that decides the input I with queries $Q_1, \ldots, Q_{f(n)}$, where $n = |I|$. Without loss of generality, the answer to the last query $Q_{f(n)}$ is the result of the computation, i.e., accept or reject. Call any bit string $w = w_1, \ldots, w_{f(n)}$, where $w_i \in \{0, 1\}$, an answer string for M on I. We say w is compatible on I, if the computation of M on I, in which the answer to Q_i (i.e., the i-th query constructed) is replaced by w_i, is such that for each query Q_i where $w_i = 1$ the answer is 'yes'. Let $\text{int}(w)$ be the integer encoded by binary string w. Then it holds that the correct computation of M on I is reflected in the answer string w_{opt} such that the number $\text{int}(w_{\text{opt}})$ is maximal among all compatible answer strings w. That is, $w_{\text{opt}} = \arg\max_{w \in \text{cmp}(I)} \text{int}(w)$, where $\text{cmp}(I)$ is the set of all compatible strings w on I.

Now consider the following problem Comp, whose instances are (M, I, k), where M is a $P^O[\log n]$ machine as described, I is an input of M, and $k \geq 0$ is an integer. The instance is a yes-instance iff M has on I some compatible answer string w such that $\text{int}(w) \geq k$. The problem Comp is in the complexity class O (and in fact complete for O). Now we construct instances $I_i = (M, I, i), i = 0, \ldots, 2|I| + 1$. Clearly, here a no-instance I can be followed only by no-instances, and the maximal i such that I_i is a yes-instance gives us the result of the computation of M on input I; if I is a yes-instance, the last oracle query $Q_{f(n)}$ has answer 'yes' (i.e., $w_{f(n)} = 1$), so i is odd; as we start from index 0, we thus have an even number of yes-instances among all I_i; if I is a no-instance, then i is even and thus we have an odd number of yes-instances among the instances I_i.

The result for the particular O-complete problem COMP follows from this. We can transform the instances I_i of COMP to instances of an arbitrary O-complete problem D in polynomial time, and so obtain hardness for an arbitrary D-complete problem. □

Proposition 16 *Let $O \in C_{\text{IMPL}}$ be the complexity of* IMPL*. Then, the problem of deciding* CONS *is $\mathsf{P}_{\|}^{O}$-hard.*

Proof. We show this by a reduction for the problem EVEN, where we can assume, by Lemma 15, that the instances I_j are ordered such that all yes-instances precede all no-instances, and w.l.o.g $m = 2n + 1$ and I_0 is a yes-instance. That is, there exists an index k, where $0 \leq k \leq 2n + 1$, such that I_j is a yes-instance iff $j < k$. We use IMPL as the decision problem D with according complexity O and associate with each instance I_j an effect ϕ_j, such that I_j is a yes-instance iff $P \models \phi_j$ for a given effect mapping P. Thus, we can consider IMPL instances $I_j : T^{(j)} \models \alpha_j$ such that their answers amount to $P \models \phi_j$.

Then, we design the conflict specification Sp such that for a distinguished conflict χ it holds that $\chi \in \mathcal{C}_G^{P,Sp}(I)$ if and only if the maximum index of a yes-instance is even. For the ASP cases we use the following set of rules:

$$\{\chi \leftarrow \phi_{2j}, \text{not}\, \phi_{2j+1} \mid 1 \leq j \leq n\} \tag{6.2}$$

If the maximum index k of a yes-instance I_k is even, i.e., $k = 2j$ for some $0 \leq j \leq n$, then ϕ_k is derivable, but ϕ_{k+1} is not. Consequently, the rule

$$\chi \leftarrow \phi_k, \text{ not } \phi_{k+1} \tag{6.3}$$

fires and χ is derived. Contrary, if the maximum index k of yes-instance I_k is odd, i.e., $k = 2j + 1$ for some $0 \leq j \leq n$, then no rule in (6.2) can fire, since for all derivable effects ϕ_ℓ with even $\ell < k$, also the effect $\phi_{\ell+1}$ is entailed.

For the first-order case, we use the following formula:

$$\chi \leftrightarrow \bigvee_{j=0}^{n} \phi_{2j} \wedge \neg\phi_{2j+1} \tag{6.4}$$

Similarly as in the ASP case, χ holds iff for some even index $k = 2j$ the effect ϕ_k is derivable and ϕ_{k+1} is not, i.e., iff the maximum index of yes-instances I_k is even. □

From these results, we obtain the following theorem.

Theorem 17 *For* CONS*, the complexity results in Table 6.2 hold.*

Next, we investigate minimal diagnoses.

Logic $\mathcal{L}$	general case	BPA
FOL+DCA	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}}$	PSPACE
$\text{ASP}^{\neg_s}$	EXP	P^{NP}
$\text{ASP}^{\neg}$	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}}$	$\mathsf{P}_{\parallel}^{\Sigma_2^p}$
$\text{ASP}^{\vee,\neg}$	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}^{\mathsf{NP}}}$	$\mathsf{P}_{\parallel}^{\Sigma_3^p}$

Table 6.2: Complexity results for CONS (completeness results)

6.3 Unique Minimal Diagnosis

We now study the complexity of deciding whether a unique minimal diagnosis exists.

Problem:	UMINDIAG
Instance:	A traffic regulation problem $\mathcal{T}$ and a set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$
Output:	Decide whether C has a unique $\subseteq$-minimal diagnosis

Lemma 18 *For given $J \subseteq I$, deciding whether $C \subseteq \mathcal{C}_G^{P,Sp}(J)$ holds is in $\mathsf{P}_{\parallel}^{O}$ for respective* IMPL *classes $O \in C_{\text{IMPL}}$.*

Proof (Sketch). To decide this problem, we must determine whether each conflict in C is derivable by J. Similarly as for CONS, we can test in a first round of parallel oracles queries for each $f(t,v,w) \in F_G$ whether $f(t,v,w,) \in \mathcal{F}_G^P(J)$ with an oracle for IMPL. Then, in the second round, we check for each $c(t,v) \in C$ whether $c(t,v) \in \mathcal{C}_G^{P,Sp}(J)$. The machine answers 'yes' iff every query of the second round answers 'yes'. This puts this problem in the complexity classes $\mathsf{P}_{\parallel[2]}^{O}$, and by Lemmas 13 and 14, in the classes $\mathsf{P}_{\parallel}^{O}$, for respective IMPL complexity classes O. □

Proposition 19 *Suppose that* IMPL *is in class $O \in C_{\text{IMPL}}$. Then, deciding* UMINDIAG *is in $\mathsf{P}_{\parallel}^{O}$ in the general case. With bounded predicate arities,* UMINDIAG *is decidable in PSPACE for FOL+DCA, and in $\mathsf{P}_{\parallel}^{\mathsf{NP}^O}$ for the ASP classes.*

Proof (Sketch). Let $C \subseteq \mathcal{C}_G^{P,Sp}(I)$ be a set of conflicts. By definition, there always exists a diagnosis. For $O \in \{\mathsf{EXP}, \mathsf{PSPACE}\}$, an algorithm can cycle through all $J \subseteq I$ and compute two minimal diagnoses in exponential time (respectively polynomial space), provided two exist.

For the other IMPL complexity classes $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \mathsf{P}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$, the following more involved method works. Like for CONS, we first compute with a round of parallel O oracle calls the effects $\mathcal{F}_G^P(I)$. Every cardinality-minimal diagnosis is a $\subseteq$-minimal diagnosis. Therefore, in a second round, we ask oracles for $k = 0, \ldots, |I|$, whether for all $J \subseteq I$ of size $|J| = k$, it holds that $C \not\subseteq \mathcal{C}_G^{P,Sp}(J)$. For the considered $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}\}$, each oracle call is in O, since exponentially many subsets J of I can be considered without complexity increase, while for the other classes $O \in \{\mathsf{P}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$ it is in NP^O. The smallest k for which the oracle answers "no" is the size k^* of a *smallest* (in terms of cardinality) diagnosis, and thus of some $\subseteq$-minimal diagnosis.

In a further round, we ask an oracle whether for all $J_1, J_2 \subseteq I$ such that $|J_1| = k^*$ and $J_1 \not\subseteq J_2$ it holds that not both sets are diagnoses for C, i.e., either (i) $C \not\subseteq \mathcal{C}_G^{P,Sp}(J_1)$ or (ii) $C \not\subseteq \mathcal{C}_G^{P,Sp}(J_2)$. The answer will be 'yes' if and only if a unique minimal diagnosis exists. Overall, the method uses three rounds of O (respectively NP^O) oracle calls, which puts the problem in the class $\mathsf{P}^O_{\|[3]}$ (resp. $\mathsf{P}^{\mathsf{NP}^O}_{\|[3]}$). By Lemma 14, this class coincides with $\mathsf{P}^O_{\|}$ (resp. $\mathsf{P}^{\mathsf{NP}^O}_{\|}$), and for $O = \mathsf{P}^{\mathsf{NP}}$ we have $\mathsf{NP}^{\mathsf{P}^{\mathsf{NP}}} = \mathsf{NP}^{\mathsf{NP}}$, thus $\mathsf{P}_{\|}^{\mathsf{NP}^{\mathsf{P}^{\mathsf{NP}}}} = \mathsf{P}_{\|}^{\Sigma_2^p}$. □

Proposition 20 *Let $O \in C_{\text{IMPL}}$ the complexity of* IMPL. *Then, deciding* UMINDIAG *is $\mathsf{P}^O_{\|}$-hard in the general case. With bounded predicate arities,* UMINDIAG *is* PSPACE-*hard for FOL+DCA, and* co-NP^O-*hard for the ASP classes.*

Proof (Sketch). The hardness results for $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}\}$ are obtained by a reduction of the complement of the EVEN problem (which is also $\mathsf{P}^O_{\|}$-hard) that is a variant of the reduction for the CONS problem. The conflict rules are extended to

$$\{\chi \leftarrow \phi_{2j}, \text{not}\, \phi_{2j+1}, in_1 \mid 1 \leq j \leq n\}\,, \tag{6.5}$$

and a further rule is added:

$$\chi \leftarrow in_0\,. \tag{6.6}$$

Here, in_0 and in_1 are fresh input facts. Then, $J = \{in_0\}$ is a minimal diagnosis, and it is the single one iff $J = \{in_1\}$ is not a diagnosis, which is the case iff the EVEN instance is a no-instance.

For $O \in \{\mathsf{EXP}, \mathsf{PSPACE}\}$, hardness follows directly from IMPL, which is already hard for these classes. For the remaining cases, i.e. $O \in \{\mathsf{P}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$, one can show hardness for co-NP^O by a reduction from evaluation of suitable QBFs. Hardness for NP^O, let alone for $\mathsf{P}_{\|}^{\mathsf{NP}^O}$, however, is not apparent. □

Theorem 21 *For* UMINDIAG, *the complexity results in Table 6.3 hold.*

Next, we study the complexity of deciding correspondence.

6.4 Correspondence

Our next problem is to decide whether measures and signs express the same effects, i.e., whether $\mathcal{F}_G^P(M) = \mathcal{F}_G^P(S)$ holds.

Problem:	CORRESPONDENCE (CORR)
Instance:	A traffic regulation problem $\mathcal{T}$ with measures M and signs S
Output:	Decide whether M and S correspond

Logic $\mathcal{L}$	general case	BPA
FOL+DCA	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}}$	PSPACE
ASP$^{\neg_s}$	EXP	in $\mathsf{P}_{\parallel}^{\Sigma_2^p}$, Π_2^p-hard
ASP$^{\neg}$	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}}$	in $\mathsf{P}_{\parallel}^{\Sigma_3^p}$, Π_3^p-hard
ASP$^{\vee,\neg}$	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}^{\mathsf{NP}}}$	in $\mathsf{P}_{\parallel}^{\Sigma_4^p}$, Π_4^p-hard

Table 6.3: Complexity results for UMinDiag (completeness results by default)

Proposition 22 *Let $O \in C_{\text{Impl}}$ be the complexity class of* Impl. *Then, deciding* Corr *is in $\mathsf{P}_{\parallel}^{O}$.*

Proof (Sketch). Without loss of generality, we assume separate effect mappings P^M, P^S for measures and signs, respectively. With an oracle for Impl of complexity O in the underlying logic, we can then determine in the first round of parallel oracle queries the effects $f(t,v,w) \in F_G$ for which "$f(t,v,w) \in \mathcal{F}_G^P(I)$?" holds, where $P = P^M$. This computes the effects of the measures. Then, in a second round, we likewise compute the effects of the signs by taking $P = P^S$. We compare the answers of these two consecutive rounds of oracle queries in polynomial time, which yields membership in $\mathsf{P}_{\parallel[2]}^{O}$. This coincides with $\mathsf{P}_{\parallel}^{O}$, for $O \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{EXP}, \mathsf{PSPACE}\}$ by Lemma 13; and for the remaining classes of Impl, i.e. $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$, by Lemma 14.

Proposition 23 *Let $O \in C_{\text{Impl}}$ be the complexity of* Impl. *Then, the problem of deciding* Corr *is $\mathsf{P}_{\parallel}^{O}$-hard.*

Proof (Sketch). The hardness results for $O \in \{\mathsf{EXP}, \mathsf{PSPACE}\}$ again follow from O-hardness of Impl. For the other classes $O \in \{\mathsf{P}^{\mathsf{NP}}, \text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$, hardness for $\mathsf{P}_{\parallel}^{O}$ is shown similarly as for problem Cons by a reduction from the Even problem. We use effect mappings P^M and P^S, which consist of P from there with additional rules (respectively implications for FOL+DCA) as follows: P^M contains

$$\phi'_j \leftarrow \phi_0, \ldots, \phi_{2j+1} \tag{6.7}$$

and P^S contains

$$\phi'_j \leftarrow \phi_0, \ldots, \phi_{2j}\,, \tag{6.8}$$

for $0 \leq j \leq m$ where ϕ'_j are new effect atoms. Again, $P \models \phi_j$ indicates that I_j is a yes-instance of Impl. If the maximum index k of a yes-instance is even, i.e., $k = 2j$ for some $0 \leq j \leq n$, then both $\mathcal{F}_G^{P^M}(I)$ and $\mathcal{F}_G^{P^S}(I)$ contain all atoms $\phi'_0, \ldots, \phi'_{2j}$. If k is odd, i.e., $k = 2j{+}1$, then $\mathcal{F}_G^{P^M}(I)$ will include $\phi'_k = \phi'_{2j+1}$, but $\mathcal{F}_G^{P^S}(I)$ will not. Assuming, without loss of generality, that I_{2n+1} is a no-instance, the specifications will correspond, i.e., $\mathcal{F}_G^{P^M}(I) = \mathcal{F}_G^{P^S}(I)$, iff the number of yes-instances among $I_0, \ldots, I_{2n+1}$ is even. □

Logic $\mathcal{L}$	general case	BPA
FOL+DCA	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}}$	PSPACE
$\mathrm{ASP}^{\neg_s}$	EXP	P^{NP}
$\mathrm{ASP}^{\neg}$	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}}$	$\mathsf{P}_{\parallel}^{\Sigma_2^p}$
$\mathrm{ASP}^{\vee,\neg}$	$\mathsf{P}_{\parallel}^{\mathsf{NEXP}^{\mathsf{NP}}}$	$\mathsf{P}_{\parallel}^{\Sigma_3^p}$

Table 6.4: Complexity results for CORR (completeness results)

By the previous two propositions, we get the following results for CORRESPONDENCE.

Theorem 24 *For* CORR, *the complexity results in Table 6.4 hold.*

Among our major reasoning tasks, repair remains to be investigated.

6.5 Repair

Finally, we investigate the complexity of the related decision problem for the REPAIRS reasoning task as defined in Section 5.3. There, we made use of a generic adequacy criterion *adq* on (sets of) updates (I^-, I^+) to sort out irrelevant repairs. Here, we restrict to a subclass of such criteria which can determine the adequacy of a (single) repair (I^-, I^+) in polynomial time. We call according predicates $\mathcal{A}$ on repairs *admissibility predicates.* For instance, such a predicate $\mathcal{A}$ might test whether $I^- = M \cup S$, i.e., whether the entire input is to be deleted. We define the following decision problem.

Problem:	REPAIR
Instance:	An inconsistent traffic regulation problem $\mathcal{T}$ and a polynomial-time admissibility predicate $\mathcal{A}$ for repairs
Output:	Decide whether a repair (I^-, I^+) exists such that $\mathcal{A}(I^-, I^+)$ holds

Proposition 25 *Let $\mathcal{S} = \{\mathsf{EXP}, \mathsf{PSPACE}\}$. For* IMPL *classes $O \in \mathcal{S}$, deciding* REPAIR *is in O. For the other* IMPL *classes $O \in C_{\mathrm{IMPL}} \setminus \mathcal{S}$, deciding* REPAIR *is in NP^O.*

Proof (Sketch). We observe that, given an input I, each subset $J \subseteq I_G$ gives a unique update (I^-, I^+) by $I^- = I \setminus J$ and $I^+ = J \setminus I$. Thus, to solve REPAIR, we can first guess an update by some $J \subseteq I_G$ and then check in polynomial time whether $\mathcal{A}(I \setminus J, J \setminus I)$ holds. Further, no conflicts may be derivable by J. For IMPL classes $O \in \{\mathsf{EXP}, \mathsf{PSPACE}\}$, this test stays within O, as one can cycle through all exponentially many $J \subseteq I_G$.

For the other IMPL complexity classes $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \mathsf{P}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$, one can decide this with an oracle for CONS. This puts the problem in class NP^C, where C is the complexity of CONS. For the considered classes O, we have by Theorem 17 CONS complexity $C = \mathsf{P}_{\parallel}^{O}$. Thus, the problem is decidable in NP^O, as $\mathsf{NP}^{\mathsf{P}_{\parallel}^{O}} \subseteq \mathsf{NP}^{\mathsf{P}^O} \subseteq \mathsf{NP}^O$. □

Proposition 26 *Let $\mathcal{S} = \{\mathsf{EXP}, \mathsf{PSPACE}\}$. For* Impl *classes $O \in \mathcal{S}$, deciding* Repair *is hard for O. For the other* Impl *classes $O \in C_{\text{Impl}} \setminus \mathcal{S}$, deciding* Repair *is NP^O-hard.*

Proof (Sketch). For $O \in \{\mathsf{EXP}, \mathsf{PSPACE}\}$, the result is obtained by a reduction from Cons, which is O-complete. Hardness for Σ_i^p, where $i \in \{2, 3, 4\}$, for respective Cons complexities $O \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{P}_{\|}^{\Sigma_2^p}, \mathsf{P}_{\|}^{\Sigma_3^p}\}$ can be shown with extensions of the QBF-encodings in [13]. For the remaining Repair classes $O \in \{\mathsf{NP}^{\mathsf{NEXP}}, \mathsf{NP}^{\mathsf{NEXP}^{\mathsf{NP}}}\}$, hardness can be shown by encodings of respective machines as logic programs, using ASP$^{\neg}$ and ASP$^{\vee,\neg}$, respectively. For FOL+DCA, one can exploit a negation free encoding of an according co-NEXP complete problem in ASP$^{\neg}$, where minimal model inference and classical inference of a designated atom coincide.

Finally, we observe that the results for Repair remain unchanged if only strict repairs are considered since Corr has lower complexity. □

We present some more observations.

Lemma 27 *The following containment holds:* $\mathsf{P}^{\mathsf{NEXP}} \subseteq \mathsf{NP}_{\|}^{\mathsf{NEXP}}$

Proof (Sketch). Let $Q_1, \ldots, Q_{p(n)}$ be the polynomially many oracle queries the $\mathsf{P}^{\mathsf{NEXP}}$ machine M makes. Without loss of generality, we assume that $Q_{p(n)}$, i.e., the answer of the last query, is the result of the overall computation. The $\mathsf{NP}_{\|}^{\mathsf{NEXP}}$ machine M' works as follows. First, it guesses the computation path of M along with query answers $g = g_1, \ldots, g_{p(n)}$, where $g_i \in \{\text{yes}, \text{no}\}$. Let Q_i^g be the query that M will construct if the answers to the queries $Q_1, \ldots, Q_{i-1}$ are $g_1, \ldots, g_{i-1}$, respectively. Machine M' constructs queries Q_i^g, where $1 \leq i \leq p(n)$, and solves them in parallel. Finally, M' checks whether the answers to Q_i^g coincide with g_i and accepts if $Q_{p(n)}^g$ is 'yes'. □

Hemachandra [25] showed that the classes for deterministic and non-deterministic computations that use a NEXP oracle coincide, i.e., $\mathsf{P}^{\mathsf{NEXP}} = \mathsf{NP}^{\mathsf{NEXP}}$. Thus, by Lemma 27, we have the following equation:

$$\mathsf{P}^{\mathsf{NEXP}} = \mathsf{NP}^{\mathsf{NEXP}} = \mathsf{NP}_{\|}^{\mathsf{NEXP}} \tag{6.9}$$

With a census technique as applied in [25], it can be shown that $\mathsf{P}^{\mathsf{NEXP}^{\Sigma_i^p}} = \mathsf{NP}^{\mathsf{NEXP}^{\Sigma_i^p}}$ holds for $i \geq 0$. Furthermore, with a similar argument as for Lemma 27, one shows that $\mathsf{P}^{\mathsf{NEXP}^{\Sigma_i^p}} \subseteq \mathsf{NP}_{\|}^{\mathsf{NEXP}^{\Sigma_i^p}}$ also holds for $i \geq 0$. Consequently, we obtain the following generalization of equation (6.9).

Lemma 28 *For all $i \geq 0$, it holds that* $\mathsf{P}^{\mathsf{NEXP}^{\Sigma_i^p}} = \mathsf{NP}^{\mathsf{NEXP}^{\Sigma_i^p}} = \mathsf{NP}_{\|}^{\mathsf{NEXP}^{\Sigma_i^p}}$.

For the logics with Impl complexity classes $O \in \{\text{co-}\mathsf{NEXP}, \text{co-}\mathsf{NEXP}^{\mathsf{NP}}, \mathsf{P}^{\mathsf{NP}}, \Pi_2^p, \Pi_3^p\}$, we obtained completeness for NP^O for Repair. By Lemma 28, the obtained Repair complexity classes NP^C, where $C \in \{\mathsf{NEXP}, \mathsf{NEXP}^{\mathsf{NP}}\}$, for FOL+DCA, ASP$^{\neg}$ and ASP$^{\vee,\neg}$

Logic $\mathcal{L}$	general case	BPA
FOL+DCA	$\mathsf{NP}_{\|}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$	PSPACE
$\mathrm{ASP}^{\neg_s}$	EXP	$\mathsf{NP}^{\mathsf{P}^{\mathsf{NP}}} = \Sigma_2^p$
$\mathrm{ASP}^{\neg}$	$\mathsf{NP}_{\|}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$	$\mathsf{NP}^{\mathsf{P}_{\|}^{\Sigma_2^p}} = \Sigma_3^p$
$\mathrm{ASP}^{\vee,\neg}$	$\mathsf{NP}_{\|}^{\mathsf{NEXP}^{\mathsf{NP}}} = \mathsf{P}^{\mathsf{NEXP}^{\mathsf{NP}}}$	$\mathsf{NP}^{\mathsf{P}_{\|}^{\Sigma_3^p}} = \Sigma_4^p$

Table 6.5: Complexity results for Repair (completeness results)

	Logic $\mathcal{L}$	Impl	Cons	Corr	UMinDiag	Repair
general	FOL+DCA	co-NEXP	$\mathsf{P}_{\|}^{\mathsf{NEXP}}$			$\mathsf{P}^{\mathsf{NEXP}}$
	$\mathrm{ASP}^{\neg_s}$	EXP	EXP			EXP
	$\mathrm{ASP}^{\neg}$	co-NEXP	$\mathsf{P}_{\|}^{\mathsf{NEXP}}$			$\mathsf{P}^{\mathsf{NEXP}}$
	$\mathrm{ASP}^{\vee,\neg}$	co-$\mathsf{NEXP}^{\mathsf{NP}}$	$\mathsf{P}_{\|}^{\mathsf{NEXP}^{\mathsf{NP}}}$			$\mathsf{P}^{\mathsf{NEXP}^{\mathsf{NP}}}$
BPA	FOL+DCA	PSPACE	PSPACE			PSPACE
	$\mathrm{ASP}^{\neg_s}$	P^{NP}	P^{NP}		in $\mathsf{P}_{\|}^{\Sigma_2^p}$, Π_2^p-hard	Σ_2^p
	$\mathrm{ASP}^{\neg}$	Π_2^p	$\mathsf{P}_{\|}^{\Sigma_2^p}$		in $\mathsf{P}_{\|}^{\Sigma_3^p}$, Π_3^p-hard	Σ_3^p
	$\mathrm{ASP}^{\vee,\neg}$	Π_3^p	$\mathsf{P}_{\|}^{\Sigma_3^p}$		in $\mathsf{P}_{\|}^{\Sigma_4^p}$, Π_4^p-hard	Σ_4^p

Table 6.6: Complexity of reasoning tasks. Top: General case. Bottom: With bounded predicate arities (BPA). Unless stated otherwise, entries are completeness results.

(general case, respectively), thus coincide with P^C and $\mathsf{NP}_{\|}^C$. The latter emphasizes the additional guess step required before consistency evaluation, which can also be carried out in parallel given a non-deterministic machine. Interestingly, Repair can be decided in P^C as well for these C, however, not with the restriction of parallel oracle queries as in Cons, Corr and UMinDiag.

Theorem 29 *For* Repair*, the complexity results in Table 6.5 hold.*

6.6 Summary

In this chapter we have analyzed the complexity of decision problems associated with our major reasoning tasks: checking consistency for a traffic regulation problem, existence of a unique minimal diagnosis for a set of conflicts, correspondence of measures and signs, and existence of an admissible repair. Table 6.6 summarizes all complexity results.

Our examination shows the high computational cost of solving of the presented reasoning tasks. Even with stratified answer set programs, deciding CONS takes exponential time in the general case. On the other hand, if predicate arities are bounded, checking consistency of a traffic regulation problem is P^{NP}-complete with $\mathrm{ASP}^{\neg_s}$, and REPAIR is Σ_2^p-complete. Across all tasks, complexity increases for normal programs and disjunctive programs, respectively. This suggests that stratified programs should be used to model the traffic regulation, whenever possible. Towards inconsistency management tasks of lower complexity, it would be interesting to study similar reasoning tasks at a less generic level, where further properties may be exploited.

CHAPTER 7

Implementation

In this thesis, we deal with logic-oriented problems on traffic regulation data. To implement programs solving the reasoning tasks defined in Chapter 5, we may in principle use any programming language or paradigm. In this chapter, we will first recall why Answer Set Programming (ASP) is a suitable choice both as underlying logic for the formal model itself, as well as an actual implementation language. We then proceed by reviewing the reasoning tasks and identify their common pattern. Based on this, we present a uniform approach for the implementation of all reasoning tasks by means of ASP. We then show in detail an executable realization for these tasks, using the solver DLV [30], and demonstrate the implementation based on examples.

7.1 Answer Set Programming

In Chapter 2 we recalled the definition of Answer Set Programming [17]. We will now review ASP both as logic and as implementation language in light of previous chapters.

7.1.1 Answer Set Programming as Logic

In Chapter 4 (page 38) we defined $Cn_G(T, X, Y) = \{y \in Y \mid T \cup \overline{G} \cup \overline{X} \models y\}$, which served as core for the definitions of an effect mapping P and a conflict specification Sp. The underlying logic was left open. We defined a two-stage mapping approach from measures and signs to effects, and then from effects to conflicts and observed that, when using Answer Set Programming, we can compute both stages at once. Given a traffic regulation problem $\mathcal{T}$ with input I we obtain the conflicts of I in $\mathcal{T}$ by

$$\mathcal{C}_G^{P,Sp}(I) = Cn_G(Sp \cup P, I, C_G) = \mathcal{AS}^{\cap}_{C_G}(Sp \cup P \cup G \cup I) \qquad (7.1)$$

Recall that the latter expression abbreviates $\bigcap \mathcal{AS}(Sp \cup P \cup G \cup I) \cap C_G$, i.e., the intersection of according answer sets filtered for conflict atoms. The closed-world operator on G and X in the definition of $Cn_G(T, X, Y)$ is implicitly given with ASP's closed world assumption. That is, everything that is not derivable is assumed to be false (with respect to default negation). Following the intuition that the meaning of each configuration of measures and signs on a street graph should be unambiguous (but possibly illegal), we assumed that the traffic regulation $Sp \cup P$ is designed in a way such that for every (proper) input I the program $Sp \cup P \cup G \cup I$ has exactly one answer set. Thus, brave and cautious consequence (in terms of truth in any, respectively all answer sets) coincide and

$$Cn_G(Sp \cup P, I, C_G) = A \cap C_G, \tag{7.2}$$

where A is the answer set of $Sp \cup P \cup G \cup I$. As discussed in detail before, the restrictions of some traffic signs take effect *unless* certain conditions apply. Therefore, default negation allows for intuitive knowledge representation within our domain. Nonmonotonic reasoning, which comes with it, then enables defeasible inferences, i.e., the potential retraction of previous effects and conflicts in the presence of additional input.

Using ASP, we can directly instantiate our formal model with according rules and obtain a readable, executable specification, as we discuss next.

7.1.2 Answer Set Programming as Implementation Language

In Section 3.5.2 we highlighted the benefits of automated reasoning, modular programming and readability of a declarative program. With ASP, automation of reasoning is achieved since our sets of rules, i.e., effect mapping and conflict specification, immediately give an executable program when provided an according solver like DLV [30], the Potassco suite [21], SModels [45], Cmodels [32], GnT [26], or ASSAT [34]. Given these tools, no separate algorithms need to be developed to solve the specified reasoning tasks. With according knowledge representation, we obtain implementations for these tasks simply by specifying the properties of their solutions.

By separating logic and control of a program [28], any rule-based language improves the readability of the implementation, and in particular, the maintainability of the expressed domain semantics. Prolog is probably still the most prominent logic-oriented language. However, as discussed earlier, Prolog is not fully declarative, which imposes restrictions on modular composition of partial programs, e.g., different conflict specifications for different use cases.

Modularity plays a crucial role in our domain, since we must provide adaptability of rules without complicated redeployments. Changes in legal specifications and interpretations must be accounted for by the possibility to add and remove rules while leaving the rest of the implementation untouched. Indeed, we aim for a solution that allows for modifications of the rule set even while the overall system keeps running. Moreover, we will see below how we can solve all reasoning tasks by little additions to a core program. That is, given a fixed module to enumerate effects and conflicts for a range of inputs, we will obtain each reasoning task by the addition of respective modules, each containing only a small number of intuitive rules. A declaratively written, highly modular implementation of a program's logic significantly improves readability and maintainability, particularly in such sophisticated domains of knowledge representation and reasoning.

Beyond that, we are not interested in proof search (as in Prolog), but model building. With Answer Set Programming, we can easily encode our domain knowledge in such a way that solutions to reasoning tasks correspond with answer sets of according programs. An instance of a search problem is solved in ASP by (i) encoding it as logic program, (ii) compute the models of the program by an ASP solver and (iii) extract a solution for each model. In our case, the extraction will be a simple filtering for according target atoms that represent the defined solution set. Such atoms can then be processed further within typical software stacks towards user interaction and persistence, where other paradigms are more appropriate.

In this thesis, we focus on the logical core and present according command line scripts, i.e., logic programs to be interpreted by an ASP solver. On the mere engineering side, an integration of these scripts within the existing web application (or similar software) remains to be done.

7.1.3 The DLV System

Using ASP, we benefit from an easy transition from formal model to implementation. That is, little syntactic changes to the formal definitions immediately give executable programs. In this work we make use of the mature ASP solver DLV [30]. DLV replaces the symbol $\leftarrow$ by `:-` and $\neg$ (strong negation) by `-` (minus). Terms, where the first letter is in uppercase, are variables.

Example 45 Consider the following program.

$$\begin{aligned} T = \{ & a(X) \vee \neg a(X) \ \leftarrow \ b(c(X), Y), \ \text{not}\ d(X); \\ & b(c(x_1), x_2)\} \end{aligned}$$

We can encode T in a DLV file `T.lp` in a straightforward way. (The extension `.lp` stands for "logic program" but may be chosen arbitrarily.)

```
a(X) v -a(X) :- b(c(X),Y), not d(X).
b(c(x1),x2).
```

Since in the first rule the variable Y occurs only once, it is not of interest. Hence, we may replace it by an anonymous variable, written as _ (underscore), i.e., the expression `b(c(X),_)`. We compute the answer sets of T by a command line call to `dlv`.[1]

```
$ dlv T.lp
DLV [build BEN/Dec 17 2012 gcc 4.6.1]

{c(d(x1),x2), -a(x1)}

{c(d(x1),x2), a(x1)}
```

[1]The DLV system is available for download at `http://www.dlvsystem.com/`.

DLV offers a convenient output filter. By invoking the program with the additional parameter `-filter=a`, only atoms with predicate symbol `a` are reported, giving the filtered answer sets {`-a(x1)`} and {`a(x1)`}. Similarly, `-pfilter` limits the output to positive occurrences of according predicate symbols. Here, `-pfilter=a` yields {} and {`a(x1)`}. ■

Answer Set Programming can be seen as rule-based methodology for constraint satisfaction problems. The *guess and check* paradigm [14] suggests program composition of two parts. One subprogram *guesses* solution candidates by enumerating them as answer sets. This is achieved by employing disjunction in form of disjunctive rule heads or according default negated atoms in rule bodies. Then, another subprogram *checks* the properties of these solutions candidates and eliminates illegal ones by means of *constraints* (rules with empty heads).

In addition to this, DLV provides an optimization feature by means of so-called *weak constraints*, which are rules of form

```
:~ B. [W:L] ,
```

where `B` is any valid rule body and `W` and `L` are integers. Here, `W` is called the *weight* and `L` is called the *level* of the weak constraint. If the optional cost declaration `[W:L]` is omitted, it defaults to `[1:1]`. If such a body `B` of a weak constraint holds, the answer set is not discarded, but assigned a cost according to the values `W` and `L`. That is, each level is aggregated its own weight sum. Whenever a weak constraint with cost `[W:L]` fires, the number `W` is added to the sum on level `L`. All answer sets that do not have a minimal sum of weights on the highest level are discarded. Among the remaining ones, those are discarded that are not minimal on the next lower level, and so on. Only those answer sets that are indistinguishable after this evaluation are reported as *best models.* A formal definition of DLV's weak constraint semantics can be found in [30].

Example 46 (cont'd) Consider the DLV program `W.lp` containing the following two weak constraints:

```
:~  a(X).  [5:1]
:~ -a(X).  [1:2]
```

The first rule assigns a penalty of weight 5 on level 1 for every atom of form `a(X)` in an answer set. Similarly, the second rule adds a weight of 1 on level 2 for every atom `-a(X)`. Accordingly, in combination with program `T.lp` of Example 45, we obtain a single best model. (The `-silent` flag suppresses the version info.)

```
$ dlv -silent T.lp W.lp
Best model:  c(d(x1),x2), a(x1)
Cost ([Weight:Level]):  <[5:1],[0:2]>
```

The answer set {`c(d(x1),x2), a(x1)`} contains an atom of form `a(X)` which causes the first weak constraint in `W.lp` to fire. There are no further atoms with predicate symbol `a`. Therefore, the answer set is assigned a total weight of 5 on level 1. The body of the second weak constraint in `W.lp` does not hold. Thus, we have sum 0 at level 2.

Evaluate fixed input $I \subseteq I_G$	Enumeration of restricted $J \subseteq I_G$
Consistency Evaluation	Diagnoses
Correspondence	Repairs
	Strict Repairs
	Adjustment
	Generation

Table 7.1: Two groups of reasoning tasks

In contrast, the other answer set {`c(d(x1),x2), -a(x1)`} yields a weight sum of 1 at level 2. Hence, it is of higher cost and thus not reported as best model. ■

Weak constraints will allow us to elegantly encode the search for answer sets containing sets of atoms that are minimal with respect to cardinality as we show in detail below.

7.2 Uniform Approach for Reasoning Tasks

We discussed above that Consistency Evaluation, i.e., the computation of conflicts is solved by $Cn_G(Sp \cup P, I, C_G)$, implemented by the answer set program $Sp \cup P \cup G \cup I$, where we have to filter the resulting answer set by conflict atoms $c(t, v) \in C_G$. For Correspondence, we can compute $Cn_G(Sp \cup P, I', F_G)$ for $I' \in \{M, S\}$, and check whether the respective effects coincide. Alternatively, we can define an additional conflict specification Sp^{Cr} which entails a conflict whenever an effect is a consequence of a measure, but not any sign, or vice versa. Then, we get Correspondence by checking whether $Cn_G(Sp^{Cr} \cup P, I, C_G)$ is empty.

In Diagnoses, we want to find those $J \subseteq I$ such that all given conflicts are entailed. Dually, in Repairs, we want to list $J \subseteq I_G$ (close to I) such that none of the given conflicts are inferred. Hence, we can view these reasoning tasks as *enumeration* of $J \subseteq I_G$ such that certain conditions on J and their implied conflicts $Cn_G(Sp \cup P, J, C_G)$ hold. The remaining reasoning tasks, i.e., Strict Repairs, Adjustment and Generation are then additional restrictions on the Repairs task, as we will show in detail below. We observe that the reasoning tasks fall into two groups, as shown in Table 7.1. Towards a uniform encoding for all reasoning tasks (in answer set programming alone) we must therefore modify our traffic regulation $\Pi = Sp \cup P$ such that an enumeration of sets of measures and signs $J \subseteq I_G$ becomes possible.

Definition 26 (Reification) *Let p be a predicate symbol, T a program (respectively set of atoms) and let Q be a set of predicate symbols. We define the* p-reification *of Q in T as the program (resp. set) obtained by replacing in every rule (resp. atom) of T every occurrence of an atom x with predicate symbol in Q by an atom $p(x)$. We call p is the* reification symbol.

On a set of rules or atoms T, let $use(T)$ (resp. $input(T)$) denote the use-reification (resp. input-reification) of $\{m, s\}$ in T, i.e., all measure and sign atoms become terms of new atoms with predicate symbol use (resp. input).

Throughout, we assume that the reification symbols are fresh symbols. That is, when we employ $use(T)$ or $input(T)$, we assume the symbols *use* and *input* do not appear in T, respectively.

We observe that A is an answer set of $\Pi \cup G \cup I$ if and only if $use(A)$ is an answer set of $use(\Pi) \cup G \cup use(I)$.

Example 47 Let Π consist only of rule (4.27) (page 43) which defines the effect of a No Entry sign, i.e., $\Pi = \{f'(\ell_s, ban, X, Y) \leftarrow s(\textit{no-entry}, Y), e(T, X, Y)\}$. Moreover, let $G = \{e(\textit{straight}, a, b)\}$ and $I = \{s(\textit{no-entry}, b)\}$. We obtain the reified program $use(\Pi) \cup G \cup use(I)$ as

$$\begin{aligned}\{\ & f'(\ell_s, ban, X, Y) \leftarrow use(s(\textit{no-entry}, Y)), e(T, X, Y);\\ & e(\textit{straight}, a, b);\\ & use(s(\textit{no-entry}, b))\ \}\,.\end{aligned}$$

The set $A = \{s(\textit{no-entry}, b),\ e(\textit{straight}, a, b),\ f'(\ell_s, ban, a, b)\}$ is the unique answer set of program $\Pi \cup G \cup I$, and $use(A) = \{use(s(\textit{no-entry}, b)),\ e(\textit{straight}, a, b),\ f'(\ell_s, ban, a, b)\}$ is the single answer set of $use(\Pi) \cup G \cup use(I)$. ■

The predicate symbol *use* resembles the distinguished predicate symbol *ab* in the standard presentation of consistency-based diagnosis [11,44]. There, the normal behavior of a component *comp* is modelled with implications of form $A \supset B$, where B shall only be concluded if *comp* is not assumed to be faulty. Such an abnormality assumption for *comp* is represented as atom $ab(comp)$. To this end, the antecedent A is a conjunction including the atom $\neg ab(comp)$. Similarly, we will compute effects (and conflicts) of a measure or sign i only if $use(i)$ holds.

In order to find suitable $J \subseteq I_G$ in the reasoning tasks listed in the right column of Table 7.1, we will make use of the aforementioned *guess and check* paradigm. The following program *Pool* establishes the *guess*:

$$pool(I) \leftarrow input(I) \tag{7.3}$$

$$use(I) \vee \neg use(I) \leftarrow pool(I) \tag{7.4}$$

Observation 30 Let $input(I)$ be the *input*-reification of $\{m, s\}$ in input set I. Then, for any given $J \subseteq I$, if A is the answer set of $\Pi \cup G \cup J$, then $use(A)$ is an answer set of $use(\Pi) \cup G \cup input(I) \cup Pool$.

Based on this observation, we formalize the guess program for a traffic regulation problem, which will serve as core for the computation of all reasoning tasks.

Definition 27 (Guess) *Let $\mathcal{T} = (\Pi, Sc)$ be a traffic regulation problem with traffic regulation $\Pi = Sp \cup P$ and scenario $Sc = (G, M, S)$, $I = M \cup S$ and let Pool be a program containing rules* (7.3) *and* (7.4).[2] *The* guess (program) *for $\mathcal{T}$ is defined as*

$$guess(\mathcal{T}) \;=\; use(\Pi) \cup G \cup input(I) \cup Pool. \tag{7.5}$$

Consequently, $guess(\mathcal{T})$ evaluates the conflicts (and effects) of all subsets of $J \subseteq I$; one per answer set. To account for measures and signs $J \subseteq I_G \setminus I$, we will later add further elements to the "pool", i.e., atoms $pool(x)$ where $x \notin I$. To enable the distinction whether an element of the pool stems from the input I or not, we have rule (7.3) in *Pool* and employ in $guess(\mathcal{T})$ the set $input(I)$ instead of an according reification $pool(I)$.

Example 48 (cont'd) Let Π, G and I be as in Example 47. The guess program for $\mathcal{T}$ has two answer sets:

$$\begin{aligned} \mathcal{AS}(guess(\mathcal{T})) \;=\; & \big\{S \cup \{use(s(no\text{-}entry, b)),\ f'(\ell_s, ban, a, b)\}, \\ & \;S \cup \{\neg use(s(no\text{-}entry, b))\}\big\}, \end{aligned}$$

where $S = \{e(straight, a, b), input(s(no\text{-}entry, b)), pool(s(no\text{-}entry, b))\}$. The first answer set evaluates $J = \{s(no\text{-}entry, b)\}$, hence $use(J) = \{use(s(no\text{-}entry, b))\}$ is a subset. The second answer set is the evaluation of $J = \emptyset$. Since the No Entry sign on node b is not *used* in this case, the *ban* effect for edge (a, b) is not inferred. ∎

Given according sets of constraints X that establish the *check* of solution candidates, we obtain all reasoning tasks by programs of form $guess(\mathcal{T}) \cup X$, as we show next.

7.2.1 Consistency Evaluation

For Consistency Evaluation, we need to avoid a search of $J \subseteq I_G$ by fixing $J = I$, i.e., only the measures and signs from the input shall be evaluated. There are no further constraints. To this end, we use the following program *Eval*:

$$use(X) \;\leftarrow\; input(X) \tag{7.6}$$

$$\neg use(X) \;\leftarrow\; pool(X),\ \text{not}\ input(X) \tag{7.7}$$

The first rule forces every input element to be used for computation of effects and conflicts. The second rule accounts for potential additions to the *Pool* which we have not discussed so far. We get Consistency Evaluation as follows.

Proposition 31 *Let $\mathcal{T}$ be a traffic regulation problem. If A is the answer set of the program $guess(\mathcal{T}) \cup Eval$, then $\mathcal{C}(\mathcal{T}) = A \cap C_G$.*

[2]We assume that Π does no make use of any predicate symbol occurring in *Pool*.

Proof. Let $\mathcal{T}$ be a traffic regulation problem with traffic regulation Π, graph G and input I. Recall that $\Pi \cup G \cup I$ is assumed to have exactly one answer set. We have

$$\begin{aligned}
\mathcal{C}(\mathcal{T}) &= \mathcal{AS}^{\cap}_{C_G}(\Pi \cup G \cup I)\\
&= \bigcap \mathcal{AS}(\Pi \cup G \cup I) \cap C_G\\
&= \bigcap \{A \cap C_G \mid A \in \mathcal{AS}(\Pi \cup G \cup I)\}\\
&= \bigcap \{A \cap C_G \mid A \in \mathcal{AS}(\mathit{use}(\Pi) \cup G \cup \mathit{use}(I))\}\\
&= \bigcap \{A \cap C_G \mid A \in \mathcal{AS}(\mathit{use}(\Pi) \cup G \cup \mathit{input}(I) \cup \mathit{Pool} \cup \mathit{Eval})\}\\
&= \bigcap \{A \cap C_G \mid A \in \mathcal{AS}(\mathit{guess}(\mathcal{T}) \cup \mathit{Eval})\}\\
&= A \cap C_G \text{ where } A \text{ is the single answer set of } \mathit{guess}(\mathcal{T}) \cup \mathit{Eval}\,. \qquad \square
\end{aligned}$$

Thus, we obtain the conflicts $\mathcal{C}(\mathcal{T})$ of the traffic regulation problem $\mathcal{T}$ by the answer set of the program

$$\mathit{guess}(\mathcal{T}) \cup \mathit{Eval}\,, \tag{7.8}$$

filtered for conflict atoms C_G.

7.2.2 Correspondence

Consider the following conflict specification Sp^{Cr} that entails a conflict whenever there exists an unjustified traffic sign or an unannounced measure:

$$c(\mathit{unjustified}(F), X) \leftarrow f'(\ell_s, F, X, Y), \mathrm{not}\, f'(\ell_m, F, X, Y) \tag{7.9}$$

$$c(\mathit{unannounced}(F), X) \leftarrow f'(\ell_m, F, X, Y), \mathrm{not}\, f'(\ell_s, F, X, Y) \tag{7.10}$$

Rule (7.9) will infer a conflict whenever on some edge (X, Y) an effect of type F is concluded by means of a sign (language symbol ℓ_s), but not by means of any measure (ℓ_m). Dually, rule (7.10) captures the other case. We define the set of *correspondence conflicts* as $\mathcal{C}(\mathcal{T}^{Cr})$, where $\mathcal{T}^{Cr}$ is obtained by replacing in $\mathcal{T}$ the conflict specification Sp by Sp^{Cr}.

Corollary 32 *Let $\mathcal{T}^{Cr}$ be a traffic regulation with Sp^{Cr} as conflict specification. If A is the answer set of the program $\mathit{guess}(\mathcal{T}^{Cr}) \cup \mathit{Eval}$, then $\mathcal{C}(\mathcal{T}^{Cr}) = A \cap C_G$.*

Thus, we obtain the correspondence conflicts $\mathcal{C}(\mathcal{T}^{Cr})$ of the traffic regulation problem $\mathcal{T}$ by the answer set of the program

$$\mathit{guess}(\mathcal{T}^{Cr}) \cup \mathit{Eval}\,, \tag{7.11}$$

filtered for conflict atoms C_G. Consequently, Correspondence amounts to Consistency Evaluation, applied with the designated conflict specification Sp^{Cr}.

7.2.3 Diagnosis

Assume we are given a set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$. Then, $J \subseteq I_G$ is a diagnosis for C if (i) $J \subseteq I$ and (ii) J suffices to entail all given conflicts, i.e., every $\gamma \in C$ is contained in $\mathcal{AS}^{\cap}_{C_G}(\Pi \cup G \cup J)$. That is, we have to (i) disallow additions to the pool and (ii) eliminate those subsets of I where any of the given conflicts is *not* entailed. Towards a convenient representation, we extend *Pool* by the following two rules. Here, we assume that the symbols *add* and *del* do not appear in Π.

$$\begin{aligned} \mathit{del}(I) &\leftarrow \neg \mathit{use}(I),\ \mathit{input}(I) && (7.12)\\ \mathit{add}(I) &\leftarrow \quad \mathit{use}(I),\ \text{not}\ \mathit{input}(I) && (7.13) \end{aligned}$$

For a given set C of conflicts, we define the program

$$\mathit{Diag}_C \ = \ \{\leftarrow \mathit{add}(X)\} \ \cup \ \{\leftarrow \text{not}\, \gamma \mid \gamma \in C\}\,. \qquad (7.14)$$

Proposition 33 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts of a traffic regulation problem $\mathcal{T}$. Then, J is a diagnosis for C iff there is an answer set S of the program $\mathit{guess}(\mathcal{T}) \cup \mathit{Diag}_C$ such that $J = \{i \mid \mathit{use}(i) \in S\}$.*

Proof. Let $\mathcal{T}$ be a traffic regulation problem with traffic regulation Π, graph G and input I. Let $\mathit{use}^{-1}(S) = \{i \mid \mathit{use}(i) \in S\}$. We have

$$\begin{aligned} \textsc{Diagnoses}(\mathcal{T}, C) &= \{J \subseteq I \mid \Pi \cup G \cup J \models \gamma,\ \forall \gamma \in C\}\\ &= \{J \subseteq I \mid \mathit{use}(\Pi) \cup G \cup \mathit{use}(J) \models \gamma,\ \forall \gamma \in C\}\\ &= \{J \subseteq I_G \mid \mathit{use}(\Pi) \cup G \cup \mathit{use}(J) \cup \mathit{Diag}_C \not\models \bot\}\\ &= \{A \cap I_G \mid \mathit{use}(A) \in \mathcal{AS}(\mathit{use}(\Pi) \cup G \cup \mathit{input}(I) \cup \mathit{Pool} \cup \mathit{Diag}_C)\}\\ &= \{A \cap I_G \mid \mathit{use}(A) \in \mathcal{AS}(\mathit{guess}(\mathcal{T}) \cup \mathit{Diag}_C)\}\\ &= \{\mathit{use}^{-1}(S) \mid S \in \mathcal{AS}(\mathit{guess}(\mathcal{T}) \cup \mathit{Diag}_C)\}\,. \end{aligned} \qquad \square$$

Thus, we obtain the diagnoses for C by filtering the answer sets of the program

$$\mathit{guess}(\mathcal{T}) \cup \mathit{Diag}_C \qquad (7.15)$$

for atoms with predicate symbols *use*. Note that the subprogram $\{\leftarrow \mathit{add}(X)\}$ of Diag_C is only needed when the pool is extended beyond the input, i.e., when there are atoms of form $\mathit{pool}(x)$ where $x \notin I$. We will need such additions now for Repairs.

7.2.4 Repair

Given an input I, we can identify a set of measures or signs $J \subseteq I_G$ with an update (I^-, I^+) given by $I^- = I \setminus J$ and $I^+ = J \setminus I$. For an adequacy criterion *adq*, we may likewise write $\mathit{adq}(J)$ instead of $\mathit{adq}(I^-, I^+)$. We assume that this criterion can be expressed by an answer set program ADQ.

Practically, we are not interested in arbitrary repairs, but sets $J \subseteq I_G$ close to I. Based on domain knowledge, we will thus include further rules in *Pool* that allow additional *pool*-atoms to be considered for evaluation, increasing the search space beyond elements in I. We will create a flexible *pool* $IP \subseteq I_G$ of measures and signs based on which the guess program for $\mathcal{T}$ enumerates solution candidates. Then, a set $J \subseteq IP$ is a repair for C under *adq* if no conflict in C can be derived by J, and $adq(J)$ holds. Let C be a set of conflicts. We define $Rep_C^{adq} = Rep_C \cup ADQ$, where

$$Rep_C \;=\; \{\leftarrow \gamma \mid \gamma \in C\}\,. \tag{7.16}$$

For the following proposition, we assume that *Pool* generates all potential measures and signs on an input graph, i.e., the set $\{pool(x) \mid x \in I_G\}$. Practically, we will limit the search space to some set IP, $I \subseteq IP \subseteq I_G$, close to I.

Proposition 34 *Let $C \subseteq \mathcal{C}(\mathcal{T})$ be a set of conflicts of a traffic regulation problem $\mathcal{T}$, and let $J \subseteq I_G$. Then, $(I \setminus J, J \setminus I)$ is a repair for C in $\mathcal{T}$ under adq iff there is an answer set S of the program $guess(\mathcal{T}) \cup Rep_C^{adq}$ such that $J = \{i \mid use(i) \in S\}$.*

Proof. Let $\mathcal{T}$ be a traffic regulation problem with traffic regulation Π, graph G and input I. Let $use^{-1}(S) = \{i \mid use(i) \in S\}$. We have

$$\begin{aligned}
\textsc{Repairs}(\mathcal{T}, C, adq) &= \{J \subseteq I_G \mid \Pi \cup G \cup J \not\models \gamma, \forall \gamma \in C \text{ and } adq(J) \text{ holds}\}\\
&= \{J \subseteq I_G \mid use(\Pi) \cup G \cup use(J) \not\models \gamma, \forall \gamma \in C \text{ and } adq(J) \text{ holds}\}\\
&= \{J \subseteq I_G \mid use(\Pi) \cup G \cup use(J) \cup Rep_C^{adq} \not\models \bot\}\\
&= \{A \cap I_G \mid use(A) \in \mathcal{AS}(use(\Pi) \cup G \cup input(I) \cup Pool \cup Rep_C^{adq})\}\\
&= \{A \cap I_G \mid use(A) \in \mathcal{AS}(guess(\mathcal{T}) \cup Rep_C^{adq})\}\\
&= \{use^{-1}(S) \mid S \in \mathcal{AS}(guess(\mathcal{T}) \cup Rep_C^{adq})\}\,. \qquad \square
\end{aligned}$$

Consequently, we obtain the repairs for C in $\mathcal{T}$ under *adq* by the answer sets of the program

$$guess(\mathcal{T}) \cup Rep_C^{adq}\,, \tag{7.17}$$

filtered for atoms with predicate symbols *use*. However, by rules (7.12) and (7.13) in *Pool* we can extract repairs in a more intuitive way. Let S be an answer set of the program $guess(\mathcal{T}) \cup Rep_C^{adq}$, $I^- = \bigcup\{i \mid del(i) \in S\}$, and $I^+ = \bigcup\{i \mid add(i) \in S\}$. Then, the update (I^-, I^+) is a repair for C under *adq*.

Repairs for $\mathcal{T}$. In Chapter 5, we noted that, in general, the repairs for a traffic regulation problem $\mathcal{T}$ do not equal the repairs for $\mathcal{C}(\mathcal{T})$. While answer sets of the program $guess(\mathcal{T}) \cup Rep_C^{adq}$ are guaranteed not to contain any conflict $\gamma \in \mathcal{C}(\mathcal{T})$, they may contain new conflict atoms $\gamma \in C_G \setminus \mathcal{C}(\mathcal{T})$ due to side effects of the respective update.

Therefore, to obtain repairs for $\mathcal{T}$, we employ in addition to the guess program for $\mathcal{T}$ the program $Rep^{adq} = Rep \cup ADQ$, where

$$Rep = \{\leftarrow c(T, V)\}\,. \tag{7.18}$$

Proposition 35 *Let $\mathcal{T}$ be an inconsistent traffic regulation problem. The update (I^-, I^+) is a repair for $\mathcal{T}$ under adq if and only if there exists an answer set S of the program $guess(\mathcal{T}) \cup Rep^{adq}$ such that $I^- = \{i \mid del(i) \in S\}$ and $I^+ = \{i \mid add(i) \in S\}$.*

Thus, we obtain the repairs for $\mathcal{T}$ under *adq* by the answer sets of the program

$$guess(\mathcal{T}) \cup Rep^{adq}\,, \tag{7.19}$$

filtered for atoms with predicate symbols *del* and *add*.

7.2.5 Strict Repairs

For strict repairs, we require in addition to consistency that the effects of measures and signs correspond in the updated traffic regulation problem. We can elegantly define the set of strict repairs by adding Sp^{Cr} as second specification to the definition of Repairs. If for a set $J \subseteq I_G$, which we can view as update (I^-, I^+) for a traffic regulation problem $\mathcal{T}$, no conflict will be derived by $Sp \cup Sp^{Cr}$, then we have a consistent scenario where Correspondence holds. We assume that potential correspondence conflicts are included in C_G. Let $\mathcal{T}'$ be the traffic regulation problem obtained by $\mathcal{T}$, replacing its traffic regulation Π by $\Pi' = Sp \cup Sp^{Cr}$ and let Rep^{adq} be defined as before. Then, we obtain strict repairs for $\mathcal{T}$ under *adq* by the answer sets of the program

$$guess(\mathcal{T}') \cup Rep^{adq}\,, \tag{7.20}$$

filtered for atoms with predicate symbols *del* and *add*. With further restrictions on Strict Repairs we naturally get the remaining tasks.

7.2.6 Adjustment & Generation

In Section 5.6 we defined the adjustment and generation tasks for signs and measures as special cases of strict repair. The respective restrictions on the scenarios can be seen as preconditions, which we do not discuss here. We focus on the additional conditions on the updates. For instance, Sign Adjustment allows only for updates (I^-, I^+) where $I^- \cup I^+ \subseteq S_G$, i.e., the sets of deletions and additions can only contain traffic signs. This additional restriction is simply expressed by another two constraints in a program *SAdj*, stating that modifications of measures are not allowed.

$$\leftarrow del(m(T, V, W)) \tag{7.21}$$

$$\leftarrow add(m(T, V, W)) \tag{7.22}$$

Let $\mathcal{T}'$ and Rep^{adq} be as in program (7.20), and $SAdj^{adq} = Rep^{adq} \cup SAdj$. Then, we obtain the sign adjustments for $\mathcal{T}$ under *adq* by the answer sets of the program

$$guess(\mathcal{T}') \cup SAdj^{adq}\,, \tag{7.23}$$

filtered for atoms with predicate symbols *del* and *add*. We get measure adjustment, as well as sign/measure generation by adding similar sets of constraints (*MAdj*, *SGen*, and *MGen*, respectively) to Rep^{adq}.

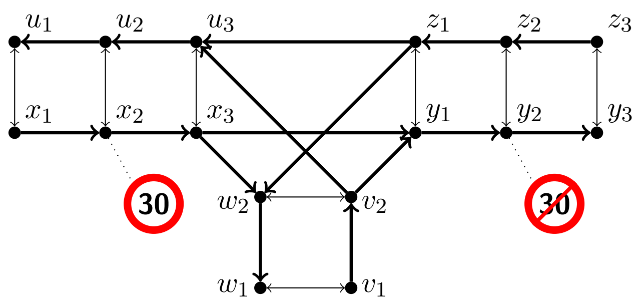


Figure 7.1: Running speed limit announcement example

7.3 Executable Realization

We will now show how to implement the reasoning tasks as executable answer set programs. Following the conceptual approach as developed in the previous section, we work out in detail a prototypical solution using DLV [30] as solver. We demonstrate how to invoke such programs from the command line but do not deal with engineering questions of how to integrate these ASP files within typical Java or C# software stacks. We will comment on the software engineering side of ASP in the final chapter.

7.3.1 Pool

First, we give the implementation that enables the guess step. We write a file `pool.lp` as follows.

```
pool(I) :- input(I).
use(I) v -use(I) :- pool(I).

del(I)  :- -use(I),     input(I).
add(I)  :-  use(I), not input(I).
keep(I) :-  use(I),     input(I).
```

In addition to program *Pool* above, we include the last rule for a more convenient representation of input measures and signs that are *kept* for evaluation, i.e., not deleted. We will later add further rules to `pool.lp`. As noted earlier, the DLV syntax closely resembles the usual mathematical notation of Answer Set Programs. The only difference here is the symbol `:-`, replacing ←, to separate a rule's head from its body.

7.3.2 Formal Model

We continue by illustrating a realization of the formal model. Based on a running example, we first show how a scenario is encoded. Then, we will turn to the implementation of the effect mapping and the conflict specification, as discussed in Chapter 4.

Example 49 Consider the scenario $Sc = (G, M, S)$ shown in Figure 7.1, which we use as running example. First, we create a file `G.lp` to represent the graph G.

```
e(lane,x1,x2). e(lane,x2,x3). e(straight,x3,y1). e(lane,y1,y2).
e(lane,y2,y3). e(lane,z3,z2). e(lane,z2,z1). e(straight,z1,u3).
e(lane,u3,u2). e(lane,u2,u1). e(lane,w2,w1). e(lane,v1,v2).

e(right,x3,w2). e(right,v2,y1). e(left,v2,u3). e(left,z1,w2).

e(uturn,x1,u1). e(uturn,u1,x1). e(uturn,x2,u2). e(uturn,u2,x2).
e(uturn,x3,u3). e(uturn,u3,x3). e(uturn,y1,z1). e(uturn,z1,y1).
e(uturn,y2,z2). e(uturn,z2,y2). e(uturn,y3,z3). e(uturn,z3,y3).
e(uturn,w2,v2). e(uturn,v2,w2). e(uturn,w1,v1). e(uturn,v1,w1).

in_node(x1).  in_node(v1). in_node(z3).
out_node(u1). out_node(w1). out_node(y3).
```

The input $I = \{s(start(spl(30)), x_2), s(end(spl(30)), y_2)\}$ of the depicted scenario, reflected as according input atoms $input(I) = \{input(i) \mid i \in I\}$, comprises no traffic measures, but two traffic signs. We save these facts as `I1.lp`.

```
input(s(start(spl(30)),x2)).
input(s(end(spl(30)),y2)).
```

By these atoms of `G.lp` and `I1.lp` we have fully represented the given scenario. ■

In Chapter 4 we discussed scenarios and assumptions on street graphs in detail. We also presented an effect mapping which we will adapt now to the new `use` atoms.

Effect Mapping

The meaning of a measure or sign `i` shall only be evaluated if an atom `use(i)` can be derived, i.e., if `use(i)` is guessed by the disjunctive rule in `pool.lp`. As explained above, we have to adapt the effect mapping P. That is, instead of testing for measures `m(t,v,w)` and signs `s(t,v)` (which are not modelled as atoms in the implementation), we must base rules on atoms of form `use(m(t,v,w))` and `use(s(t,v))`, respectively.

Example 50 (cont'd) We reify the effect mapping P of Chapter 4 and save it into a file `P.lp`. First, we collect all auxiliary facts and rules.

```
node(X) :- e(_,X,_).
node(Y) :- e(_,_,Y).

indir(X,Y) :- e(lane,X,Y).
indir(X,Y) :- e(straight,X,Y).
```

```
speed(5). speed(10). speed(20). speed(30).
speed(40). speed(50). speed(60). speed(70).
speed(80). speed(90). speed(100). speed(110).
speed(120). speed(130).

lang(ls). lang(lm).
```

Note that instead of the speed facts for the values 10 to 130 we could also write in DLV specific syntax the rule

```
speed(K) :- #int(K), #int(N), N>=1, N<=13, K=10*N.
```

DLV does not support floating numbers. Therefore, any model on continuous data must undergo a discretisation step. Here, the speed limit steps suffice for our purposes. There is no use in modelling potential driving speeds within these 10 km/h intervals.

The following are the essential rules, as explained in Chapter 4. The only difference is the replacement of every measure or sign i by `use(i)`. The predicate symbol f', reflecting language specific effects, is encoded as `fl`, the percentage sign `%` starts a single-line comment, and `!=` is DLV syntax for $\neq$, i.e., inequality.

```
f(F,X,Y) :- fl(L,F,X,Y).

% direct mapping for measures
fl(lm,F,X,Y) :- use(m(T,X,Y)), m2f(T,F).
m2f(spl(K),maxspeed(K)) :- speed(K).
m2f(traffic,nec).
m2f(no_traffic,ban).

% sign effect: start/end expression
expr_start(F,X) :- use(s(T,X)), start_of(T,F).
expr_end(F,X)   :- use(s(T,X)), end_of(T,F).
end_of(T,F) :- expl_end_of(T,F).
end_of(T,F) :- impl_end_of(T,F).
start_of(start(T),F)  :- m2f(T,F).
expl_end_of(end(T),F) :- m2f(T,F).
impl_end_of(start(spl(K)),maxspeed(J)) :- speed(K), speed(J), K!=J.

% base case and propagation
fl(ls,F,X,Y) :- expr_start(F,X), indir(X,Y).
fl(ls,F,Y,Z) :- fl(ls,F,X,Y), indir(X,Y), indir(Y,Z),
                not block_prop(F,Y).
block_prop(F,Y) :- fl(ls,F,X,Y), indir(Y,Z), expr_end(F,Y).
block_prop(F,Y) :- fl(ls,F,X,Y), e(straight,X,Y), has_perm_inc(ls,Y),
                   not expr_start(F,Y).
has_perm_inc(L,Y) :- e(left,X,Y),  lang(L), not fl(L,ban,X,Y).
has_perm_inc(L,Y) :- e(right,X,Y), lang(L), not fl(L,ban,X,Y).

% non-extending types
block_prop(ban,X) :- node(X).
block_prop(nec,X) :- node(X).

% no_turn
fl(ls,ban,X,Y) :- use(s(no_turn(T),X)), e(T,X,Y).
fl(ls,ban,X,Y) :- use(s(no_turn(left),X)), e(uturn,X,Y).

% mand_turn (omitted compound ones like left_or_straight)
fl(ls,nec,X,Y) :- use(s(mand_turn(T),X)), e(T,X,Y).
fl(ls,ban,X,Y) :- use(s(mand_turn(T),X)), e(ET,X,Y), T!=ET.

% no_entry
fl(ls,ban,X,Y) :- use(s(no_entry,Y)), e(_,X,Y).
```

This effect mapping expresses the meaning of all potential combinations of traffic signs for linear speed limits of 14 different values, 3 different prohibited driving directions (`left`, `right`, `uturn`), 4 different mandatory driving directions (additionally `straight`), and No Entry signs, and their corresponding measure types in only 27 rules and 18 facts (or 28 rules and 5 additional facts if the above rule for speed limit values was used instead). ■

In this effect mapping `P.lp` we assume that correct data is provided. Otherwise, non-existent traffic signs like `s(no_turn(straight),X)` or `s(mand_turn(lane),X)` would also be assigned effects. Such non-combinatorial data correctness checks can be seen as basic inconsistency management tasks which are flexibly implemented by a designated (or augmented) conflict specification.

In order to evaluate the input, we need to fix the considered pool of measures and signs to comprise exactly I. Therefore, we write a new program `eval.lp` containing the following two rules.

```
 use(I) :- input(I).
-use(I) :- pool(I), not input(I).
```

That is, everything that is given as input by the scenario is evaluated, and nothing else. So far, there is nothing in the pool that is not part of the input, but we will later add further measures and signs to the pool to allow additions in the repair tasks. By the rules presented so far we can compute the effects of a set of measures and signs.

Example 51 (cont'd) In Chapter 4, Example 20, we argued that the effects of an input I are given by the (single) answer set of $P \cup G \cup I$, i.e., $\mathcal{F}^P_G(I) = \mathcal{AS}^{\cap}_{F_G}(P \cup G \cup I)$. In Section 7.2 we explained that I needs to be replaced by $input(I) \cup Pool \cup Eval$, such that the same traffic regulation $\Pi = (P, Sp)$ can be used for the other reasoning tasks

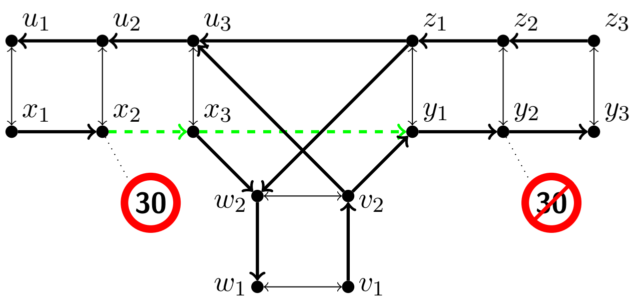


Figure 7.2: Expressed `maxspeed(30)` effect by start sign at x_2

as well. The solver DLV allows to filter the output of the computed answer sets for specified predicate symbols. Since F_G is the set of atoms with predicate symbol f, we get $\mathcal{F}_G^P(I)$ by the unique answer set of the programs developed so far, filtered by f.

```
$ dlv P.lp G.lp I1.lp pool.lp eval.lp -filter=f
DLV [build BEN/Dec 17 2012   gcc 4.6.1]

{f(maxspeed(30),x2,x3), f(maxspeed(30),x3,y1)}
```

These two atoms represent the dashed lines from x_2 to y_1 in Figure 7.2. As explained earlier, the effect propagation ends at y_1, since there is no repeated start sign, and y_1 is reachable from an incoming street via edge (v_2, y_1). ■

By an analogous adaption of the conflict specification, we will be able to determine whether a traffic regulation problem is inconsistent.

Conflict Specification

Now we are going to modify the conflict specification Sp developed throughout the previous chapters, employing the `use` atoms for the uniform approach. We discussed three classes of conflicts which we group into three blocks with an according comment. We write a file `Sp.lp` consisting of the following rules.

```
% bad announcement ends
c(bad_end(F),Y)  :- indir(X,Y), indir(Y,Z),
                    fl(ls,F,X,Y), not fl(ls,F,Y,Z),
                    needs_expr_end(F), not expr_end(F,Y).
c(cant_end(F),Y) :- use(s(T,Y)), expl_end_of(T,F), indir(X,Y),
                    not fl(ls,F,X,Y).
needs_expr_end(maxspeed(K)) :- speed(K).

% overlap
c(overlap(L,F1,F2),X) :- fl(L,F1,X,Y), fl(L,F2,X,Y), contr(F1,F2).
```

```
contr(motorway,residential_area).
contr(maxspeed(K),maxspeed(J)) :- speed(K), speed(J), K<J.
contr(ban,nec).

% no way out
c(no_way_out(lm),X) :- use(m(_,X,_)), not way_out(lm,X).
c(no_way_out(ls),X) :- use(s(_,X)), not way_out(ls,X).

way_out(L,X) :- lang(L), out_node(X).
way_out(L,X) :- lang(L), reachable(L,X,Y), way_out(L,Y).

reachable(L,X,Y) :- lang(L), e(T,X,Y), not fl(L,ban,X,Y), T!=uturn.
reachable(L,X,Y) :- e(uturn,X,Y), fl(L,nec,X,Y).
reachable(L,X,Z) :- reachable(L,X,Y), reachable(L,Y,Z).
```

We concluded Chapter 4 by the definition of a traffic regulation problem $\mathcal{T} = (Sc, \Pi)$, where $Sc = (G, M, S)$ is a scenario and $\Pi = (P, Sp)$ is a traffic regulation. We have presented an exemplary implementation for these concepts, allowing us to deal with inconsistency management tasks next.

7.3.3 Reasoning Tasks

We will now present a concrete realization for the presented reasoning tasks following the uniform approach developed in Section 7.2. Recall the guess program for a traffic regulation problem $\mathcal{T}$, which was given by

$$guess(\mathcal{T}) \;=\; use(\Pi) \cup G \cup input(I) \cup Pool\,.$$

With an according file `I.lp` formalizing the reification $input(I)$ for the input I in $\mathcal{T}$, we get an implementation for $guess(\mathcal{T})$ by the files

`Sp.lp P.lp G.lp I.lp pool.lp`.

We showed how to obtain all reasoning tasks essentially by (i) adding further constraints to the guess program and (ii) filtering the resulting answer sets for specific target atoms in the extraction step. This filtering is easily achieved by DLV's parameters `-filter`, respectively `-pfilter`.

Consistency Evaluation

We explained in Section 7.2.1 that $guess(\mathcal{T}) \cup Eval$ computes the conflicts for a traffic regulation problem $\mathcal{T}$. Thus, we have all ingredients for Consistency Evaluation.

Example 52 (cont'd) To compute the conflicts $\mathcal{C}(\mathcal{T})$ or our example problem $\mathcal{T}$, we invoke the implementation of the guess program for $\mathcal{T}$, extended by the constraints for the evaluation of the input, and filter for conflict atoms.

```
$ dlv -silent Sp.lp P.lp G.lp I1.lp pool.lp eval.lp -filter=c
{c(cant_end(maxspeed(30)),y2), c(bad_end(maxspeed(30)),y1)}
```

The option `-silent` prevents the display of DLV's version info. In the forthcoming examples, we will always use this flag but omit printing it for better readability. ∎

With the constraints of `eval.lp` we can evaluate the input, since every set $J \subseteq I$ where $J \neq I$ is discarded.

Diagnosis

Given a set $C \subseteq \mathcal{C}(\mathcal{T})$ of conflicts, we are interested in finding the diagnoses for C. In contrast to consistency evaluation, where we evaluate (only) the entire input, we are now interested in finding all subsets $J \subseteq I$ of the input, such that each conflict $c \in C$ can be derived. We observed above that we can compute the diagnoses for a set of conflicts C by extending the general guess program by constraints Diag_C that (i) disallow additions to the pool, and (ii) ensure entailment of every conflict in C. That is, we must provide a program containing a rule

```
:- not c(t,v).
```

for every $c(t, v) \in C$. By these constraints, those $J \subseteq I$ are discarded where some conflict in C cannot be derived. These constraints are problem-specific and thus not saved.

Example 53 (cont'd) In the speed limit scenario, let us consider only the conflict caused by the start sign, i.e., `c(bad_end(maxspeed(30)),y1)`. We can include additional rules and facts via the command line (after `--`), which we use to specify the conflict constraint. To see which subsets of the input suffice to entail all conflicts, we will filter for positive occurrences of `use` by `-pfilter=use`.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp -pfilter=use --
:- not c(bad_end(maxspeed(30)),y1).
{use(s(start(spl(30)),x2)), use(s(end(spl(30)),y2))}
{use(s(start(spl(30)),x2))}
```

We have two answer sets. If S is an answer set of this program, then a diagnosis for C is given by $\{i \mid \mathit{use}(i) \in S\}$. Note that we also could filter for *keep* instead of *use*. The first one is the trivial diagnosis, consisting of the entire input. The second one is the minimal diagnosis, i.e., the interesting one. That is, using only `s(start(spl(30)),x2)` is enough to satisfy the constraint. ∎

Later, input pool will contain more than the scenario's input, so we must prohibit guesses where measures or signs are added to I. Second, we are usually not interested in the list of all diagnoses, but only in minimal diagnoses.

To this end, we use DLV's weak constraints to compute minimal diagnoses with respect to cardinality. Therefore, we count the number of elements of the input that are *used* to derive the respective conflicts. By having a weak constraint that fires whenever an input atoms is used, i.e. *kept*, we count measures and signs needed to entail the conflicts and thus receive cardinality-minimal diagnoses by the answer sets. We write a new file `diagnosis.lp` comprising a constraint on additions and a weak constraint on elements from the input being kept for evaluation.

```
:- add(I).
:~ keep(I).
```

The weak constraint `:~ keep(I).` abbreviates `:~ keep(I). [1:1]`. We make use of these restrictions in the next example.

Example 54 (cont'd) Applying `diagnosis.lp` instead of `eval.lp`, and a constraint for every conflict to be diagnosed, only the minimal diagnoses with respect to cardinality are reported.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp diagnosis.lp -filter=keep --
:- not c(bad_end(maxspeed(30)),y1).
Best model: {keep(s(start(spl(30)),x2))}
Cost ([Weight:Level]): <[1:1]>
```

DLV reports only one model as the best model under the specified weak constraints. The last line says that on level 1 there was a total weight of 1 caused by weak constraints. This cost stems from the single firing of the rule `:~ keep(I)` where `I` matched with `s(start(spl(30)),x2)`. In general, the best model is not unique.

In this example, the other diagnosis, which is the entire input, would lead to two rule firings of the weak constraint, causing higher cost `<[2:1]>` and is therefore not being reported as best model. Consequently, $\{s(\mathit{start}(\mathit{spl}(30)), x_2)\}$ is the minimal diagnosis for $C = \{c(\mathit{bad\text{-}end}(\mathit{max\text{-}speed}(30)), y_1)\}$. Similarly, the other traffic sign is a minimal explanation for the other conflict, as illustrated by Figure 5.1 in the Chapter 5.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp diagnosis.lp -filter=keep --
:- not c(cant_end(maxspeed(30)),y2).
Best model: {keep(s(end(spl(30)),y2))}
Cost ([Weight:Level]): <[1:1]>
```

Since none of these signs cause both conflicts, the trivial diagnosis is also the minimal one for $\mathcal{C}(\mathcal{T})$. We compute this diagnosis by declaring a constraint for each conflict.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp diagnosis.lp -filter=keep --
:- not c(bad_end(maxspeed(30)),y1).
:- not c(cant_end(maxspeed(30)),y2).
Best model: {keep(s(start(spl(30)),x2)), keep(s(end(spl(30)),y2))}
Cost ([Weight:Level]): <[2:1]>
```

Now we have the aforementioned case of two firings of the weak constraint, resulting in a total weight of 2 on level 1. ■

We have seen how adding certain constraints to the common guess program $guess(\mathcal{T})$ leads to the implementation of two different reasoning tasks. We illustrate this principle now also for REPAIRS.

Repair

To repair a set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$, we are looking for an update (I^-, I^+) for $\mathcal{T}$ such that no conflict $c \in C$ can be derived in the updated traffic regulation problem $\mathcal{T}[I^-, I^+]$. Dually to diagnosis, we thus utilize a constraint

```
:- c(t,v).
```

for every $c(t, v) \in C$. To repair the entire traffic regulation problem $\mathcal{T}$ rather than any subset of its conflicts, we will constrain answer sets such that the entailment of *any* conflict is prohibited. We observe that the unit cost of an update (I^-, I^+) (as defined on page 60) is given by counting the number of changes to the input, i.e. $|I^- \cup I^+|$. That is, repairs under the adequacy criterion of minimal unit cost are reflected by answer sets S where the set

$$\{i \mid del(i) \in S\} \cup \{i \mid add(i) \in S\}$$

has minimal cardinality. Again, we take advantage of DLV's optimization feature. In a file `repair.lp` we specify a constraint on the entailment of any conflict (for repairs of traffic regulation problems) and the adequacy criterion of minimal unit cost by means of weak constraints on `del` and `add`.

```
:- c(_,_).
:~ del(I).
:~ add(I).
```

With these rules, only interpretations with $J \subseteq I$ that do not lead to any conflict can be answer sets. By the weak constraints, DLV will only report those answer sets with a minimal number of distinct atoms with predicate symbol `del` or `add`.

Example 55 (cont'd) We first consider *all* repairs for the *bad-end* conflict at node y_1. Since we have not stated any rules yet that may add further elements to the pool (beyond measures and signs given as input), we will not derive any atoms with predicate symbol `add`. That is, only repairs of form $(I^-, \emptyset)$ can be reported. Therefore, we will only filter for `del`.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp -filter=del --
:- c(bad_end(maxspeed(30)),y1).
{del(s(start(spl(30)),x2))}
{del(s(start(spl(30)),x2)), del(s(end(spl(30)),y2))}
```

Similarly as in the diagnosis task, we have a trivial and a minimal case. Here, the *bad-end* conflict ceases to exist if we delete at least the start sign at node x_2. We omit the similar repairs for the other conflict and turn to the repair for the traffic regulation problem by means of the constraints expressed in `repair.lp`.

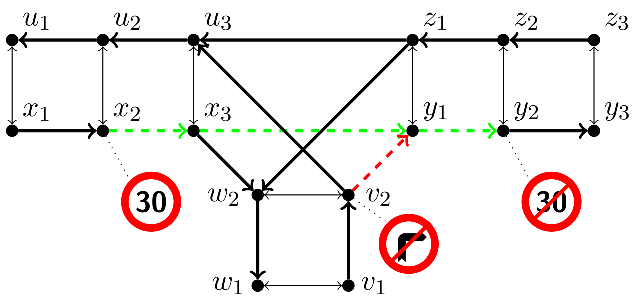


Figure 7.3: Questionable minimal repair

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp repair.lp -filter=del
Best model: {del(s(start(spl(30)),x2)), del(s(end(spl(30)),y2))}
Cost ([Weight:Level]): <[2:1]>
```

We obtain the repair $(\{s(\mathit{start}(\mathit{spl}(30)), x_2), s(\mathit{end}(\mathit{spl}(30)), y_2)\}, \emptyset)$ by reading from this answer set the atoms with predicate symbol `del`. ■

In order to enable additions in repairs, we must add new elements to the pool. To keep the search space small, we need a practical approach. The question is which measure types and which sign types to consider for which edges and nodes, respectively.

If we have a measure at some edge (X, Y), we certainly want to allow for a start sign at X and an explicit end sign at Y. Likewise, if we have such traffic signs, we want to consider the corresponding measure. In order to get some flexibility concerning the extent of represented measures, we associate the measure type with the remaining edges in direction of traffic. We extend `pool.lp` as follows.

```
pool(s(TS,X))   :- pool(m(TM,X,Y)), m2f(TM,F), start_of(TS,F).    % (1)
pool(s(TS,Y))   :- pool(m(TM,X,Y)), m2f(TM,F), expl_end_of(TS,F). % (2)
pool(m(T,X,Y)) :- pool(s(start(T),X)), indir(X,Y).              % (3)
pool(m(T,X,Y)) :- pool(s(end(T),Y)), indir(X,Y).                % (4)
pool(m(T,X,Y)) :- pool(m(T,Z,X)), indir(Z,X), indir(X,Y).       % (5)
```

Whenever we have a measure type associated with some edge (X, Y), rules `(1)` and `(2)` add according start and end signs to the pool. Thus, if these signs were missing in the input, they are now available. Similarly, if we have a start or end sign, rules `(3)` and `(4)` make sure that the corresponding measure is considered as well. Rule `(5)` stretches the potential extent of a measure in direction of traffic.

Example 56 (cont'd) DLV supports the evaluation of conjunctive queries, which we use to test what has become available in the pool in addition to the scenario's input. Queries are evaluated bravely or cautiously. Since the `pool` atoms are a deterministic consequence of the `input` atoms, there is no difference. In fact, towards our intention, the pool *must* be the same in each answer set.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp -cautious --
pool(X), not input(X)?
m(spl(30),x2,x3)
m(spl(30),x3,y1)
m(spl(30),y1,y2)
m(spl(30),y2,y3)
s(start(spl(30)),x3)
s(start(spl(30)),y1)
s(start(spl(30)),y2)
s(end(spl(30)),x3)
s(end(spl(30)),y1)
s(end(spl(30)),y3)
```

For better readability, DLV's original output was reordered here. By rule `(3)`, the start sign at node x_2 leads to the introduction of a speed limit measure on edge (x_2, x_3), which gets propagated until node y_3 due to rule `(5)`. Due to the rules `(1)` and `(2)`, we also introduce start and end signs for the involved nodes. This augmented pool allows us to derive the intended repair.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp repair.lp -filter=del,add
Best model: {add(s(start(spl(30)),y1))}
Cost ([Weight:Level]): <[1:1]>
```

The update $(\emptyset, s(\mathit{start}(\mathit{spl}(30)), y_1))$, i.e., the addition of the repeated start sign after the junction, is now the only minimal repair. In Chapter 5 we discussed other minimal repairs, such as the addition of a No Right Turn sign on node v_2, as shown again in Figure 7.3. Since the scenario does not contain any measure data, and in particular, no information about a traffic ban along edge (v_2, y_1), there is no reasonable argument why this repair should be proposed. However, suppose the user of the application explicitly requires this sign to be added. Then, the repair task reports that nothing else remains to be changed.

```
$ dlv Sp.lp P.lp G.lp I1.lp pool.lp repair.lp -filter=del,add --
use(s(no_turn(right),v2)).
Best model: {add(s(no_turn(right),v2))}
Cost ([Weight:Level]): <[1:1]>
```

According to our criterion of minimal unit cost, the addition of a No Right Turn sign at v_2 (instead of a start sign for a 30 km/h speed limit at y_1) is formally an equally good repair, albeit not a proposed one due to our limited pool. ■

The discussion of what should to be included in the pool brings us to the issue of correspondence between measures and signs.

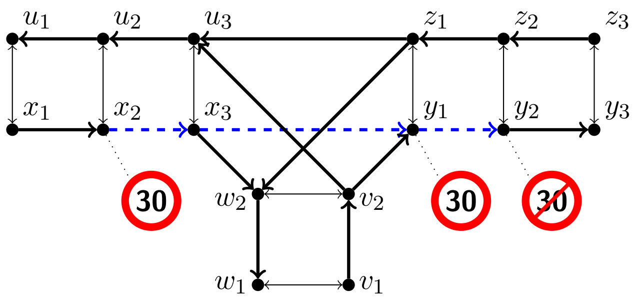


Figure 7.4: Correct sign posting for a 30 km/h speed limit measure from x_2 to y_2

Correspondence

In Section 7.2.2 we argued that Correspondence can be solved by applying Consistency Evaluation with a designated specification of correspondence conflicts. To this end, we write the following file `Corr.lp`:

```
c(unjustified(F),X) :- fl(ls,F,X,Y), not fl(lm,F,X,Y).
c(unannounced(F),X) :- fl(lm,F,X,Y), not fl(ls,F,X,Y).
```

Whenever a sign effect of type `F` is derived for an edge from `X` to `Y`, for which no according measure effect `F` can be derived, the first rule will lead to conflict of type `unjustified(F)` on node `X`. Similarly, by the second rule the label `unannounced(F)` will be associated with `X` in the reverse case.

With these rules, correspondence checking amounts to consistency evaluation, applying `Corr.lp` instead of `Sp.lp` as conflict specification.

Example 57 (cont'd) Since our scenario's input contains only signs, we must have unjustified effects.

```
$ dlv Corr.lp P.lp G.lp I1.lp pool.lp eval.lp -filter=c
{c(unjustified(maxspeed(30)),x2), c(unjustified(maxspeed(30)),x3)}
```

For both edges (x_2, x_3) and (x_3, y_1), there exists a sign effect of type `maxspeed(30)`, but no such measure effect can be derived. Hence, the effects are unjustified. ■

In case of inconsistency of either language, the question of correspondence is secondary. However, due to our approach we can evaluate and also establish consistency and correspondence in one step, which we demonstrate next.

Strict Repair

The repair task (for traffic regulation problems $\mathcal{T}$) ensures that no answer set contains a conflict. Consequently, if we include `Corr.lp` in the repair task, we automatically obtain strict repairs.

Example 58 (cont'd) A strict repair under minimal unit cost for our speed limit scenario leads to the deletion of both traffic signs.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I1.lp pool.lp repair.lp -filter=del,add
Best model: {del(s(start(spl(30)),x2)), del(s(end(spl(30)),y2))}
Cost ([Weight:Level]): <[2:1]>
```

In this setting, we can further explore the flexibility which comes with an ASP-based implementation. Suppose a user knows that all existing signs are correct, but some may be missing. She is surprised by the repair proposal and explicitly states via the user interface, that these two signs (or all signs) must not be deleted. Such flexible restrictions can always be added directly via additional constraints. For instance, a user may be provided the option to prevent all signs from being deleted, which can be instantly translated to a constraint $\leftarrow \mathit{del}(s(T, V))$. Likewise, she might click on single traffic sign symbols and manually decide whether it should be kept or deleted. Here, we present a variant which expresses the choice to use both given signs.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I1.lp pool.lp repair.lp -filter=del,add --
use(s(start(spl(30)),x2)).
use(s(end(spl(30)),y2)).
Best model: {add(m(spl(30),x2,x3)), add(m(spl(30),y1,y2)),
add(s(start(spl(30)),y1)), add(m(spl(30),x3,y1))}
Cost ([Weight:Level]): <[4:1]>
Best model: {add(m(spl(30),x2,x3)), add(m(spl(30),y1,y2)),
add(s(start(spl(30)),y1)), add(s(end(spl(30)),x3))}
Cost ([Weight:Level]): <[4:1]>
```

Now we have two optimal strict repairs which both require 4 changes to the input. Both repairs add the missing start sign at y_1. The first repair then adds the corresponding measure type to the three edges from x_2 to y_2. The update traffic regulation problem due to this update is shown in Figure 7.4. The second repair spares the measure on edge (x_3, y_1) and instead adds an end sign before the junction. ■

We will now review another example of the previous chapter which contained inconsistent information in both languages, measures and signs.

Example 59 In Example 42, we presented a scenario where two speed limit measures of 30 km/h and 40 km/h, respectively, overlap along the edge (y_1, y_2) of the T-junction graph. Additionally, there is a start sign at x_2 for the 30 km/h measure, and an end sign for the other measure at y_3. Figure 7.5 shows this scenario again. The according input `I2.lp` is given as follows.

```
input(m(spl(30),x2,x3)). input(m(spl(30),x3,y1)). input(m(spl(30),y1,y2)).
input(m(spl(40),y1,y2)). input(m(spl(40),y2,y3)).
input(s(start(spl(30)),x2)). input(s(end(spl(40)),y3)).
```

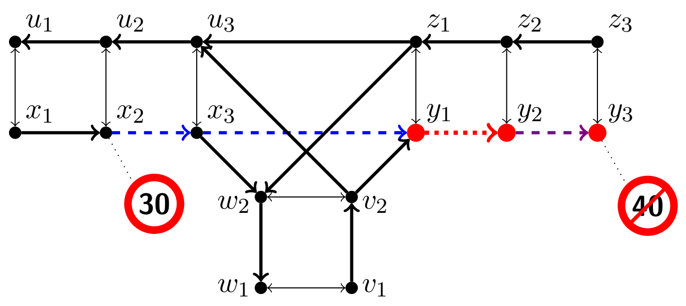


Figure 7.5: Scenario where both measures and signs are inconsistent. On edge (y_1, y_2), two speed limit measures of 30 km/h and 40 km/h overlap.

We apply a variant of consistency evaluation and correspondence check in one step in form of a query for all conflicts. We have results for nodes y_1, y_2 and y_3, as indicated in the figure.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I2.lp pool.lp eval.lp -cautious --
c(T,V)?
overlap(lm,maxspeed(30),maxspeed(40)), y1
cant_end(maxspeed(40)), y3
bad_end(maxspeed(30)), y1
unannounced(maxspeed(40)), y1
unannounced(maxspeed(40)), y2
unannounced(maxspeed(30)), y1
```

The strict repair under minimal unit cost suggests the following update.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I2.lp pool.lp repair.lp -filter=del,add
Best model: {del(m(spl(30),y1,y2)), add(s(start(spl(40)),y1))}
Cost ([Weight:Level]): <[2:1]>
```

The updated traffic regulation problem $\mathcal{T}[\{m(\mathit{spl}(30), y_1, y_2)\}, \{s(\mathit{start}(\mathit{spl}(40)), y_1)\}]$ due to this strict repair was shown in Figure 5.11 (page 69). It favours the 40 km/h measure on edge (y_1, y_2) based on the fact that retaining the label $\mathit{spl}(30)$ on that edge would require one more change:

```
$ dlv Sp.lp Corr.lp P.lp G.lp I2.lp pool.lp repair.lp -filter=del,add --
use(m(spl(30),y1,y2)).
Best model: {del(m(spl(40),y1,y2)), add(s(start(spl(30)),y1)),
add(s(start(spl(40)),y2))}
Cost ([Weight:Level]): <[3:1]>
```

That is, by forcing the use of the 30 km/h measure on edge (y_1, y_2) via the additional axiom `use(m(spl(30),y1,y2))`, we obtain the other (non-minimal) repair, as shown in Figure 5.12.

We illustrated that ad-hoc restrictions can be easily added with ASP. Apart from that, we will use predefined sets of restrictions to specify practically relevant subclasses of strict repairs. Two of these specific repair use cases have been introduced in Chapter 5. We will describe their realization next.

Adjustment

The implementation presented so far was based on a common set of rules from which we obtained different reasoning tasks by small extensions. Likewise, we get sign and measure adjustment by further constraining strict repairs. We write a new file `s-adj.lp` for sign adjustment, which can be used instead of (or in fact, also in addition to) `repair.lp`.

```
:- c(_,_).
:~ del(s(T,X)).
:~ add(s(T,X)).
:- del(m(T,X,Y)).
:- add(m(T,X,Y)).
```

The difference to `repair.lp` is that only modifications of signs are allowed (but penalized). Any answer set that suggests a deletion or an addition of a measure is discarded.

Example 60 In Example 43 we presented a scenario which is shown again in Figure 7.6. The following set of atoms reflects the input of this scenario, saved as `I3.lp`.

```
input(m(spl(30),x2,x3)). input(m(spl(30),x3,y1)).
input(m(spl(40),y1,y2)).
input(s(start(spl(40)),y1)). input(s(end(spl(40)),y2)).
```

Evaluating the scenario with a program including `Corr.lp`, we have no conflicts other than two correspondence conflicts at nodes x_2 and x_3, since the 30 km/h speed limit measure is unannounced.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I3.lp pool.lp eval.lp -filter=c
{c(unannounced(maxspeed(30)),x2), c(unannounced(maxspeed(30)),x3)}
```

The sign adjustment gives the same result as strict repair in a single best model proposing the addition of a 30 km/h start sign at x_2.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I3.lp pool.lp s-adj.lp -filter=del,add
Best model: {add(s(start(spl(30)),x2))}
Cost ([Weight:Level]): <[1:1]>
```

As we conclude by consistency evaluation, the traffic sign data (viewed in isolation) is consistent. Suppose an expert user knows that this traffic sign data is complete but that measure information may be flawed. Then the proper task is measure adjustment, which we similarly compute by prohibiting changes of signs and penalizing changes of measures (`m-adj.lp`).

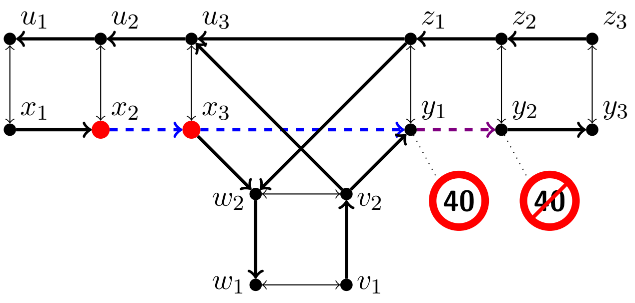


Figure 7.6: Consecutive 30 km/h and 40 km/h speed limit measures with correspondence conflicts at x_2 and x_3

```
$ dlv Sp.lp Corr.lp P.lp G.lp I3.lp pool.lp m-adj.lp -filter=del,add
Best model: {del(m(spl(30),x2,x3)), del(m(spl(30),x3,y1))}
Cost ([Weight:Level]): <[2:1]>
```

Applying measure adjustment to this scenario, the unannounced measures along the edges (x_2, x_3) and (x_3, y_1) are proposed to be deleted. This is not suggested by the strict repair, since it has higher unit cost than adding a single sign. ■

Next, we discuss the similar generation task, in which consistent input of only one language shall be used to generate information of the other language from scratch. This task is realized by another restriction on strict repairs.

Generation

In sign (respectively measure) generation, we deal with a scenario $(G, M, \emptyset)$ (respectively $(G, \emptyset, S)$), where M (respectively S) is consistent. We do not check these preconditions here. In fact, the following constraints for generations of either signs or measures may also be applied for scenarios which already contain elements of the respective language. If we apply sign (respectively measure) generation on scenarios where the set of measures (respectively signs) is non-empty, we get a variant of adjustment where only additions are allowed. These are the rules for measure generation, saved as `m-gen.lp`:

```
:- c(_,_).
:- del(s(T,X)).
:- add(s(T,X)).
:- del(m(T,X,Y)).
:~ add(m(T,X,Y)).
```

By this set of rules, only the addition of measures is allowed (and penalized), all other modifications are prohibited. Likewise, for sign generation (`s-gen.lp`), we prohibit all changes but the addition of signs.

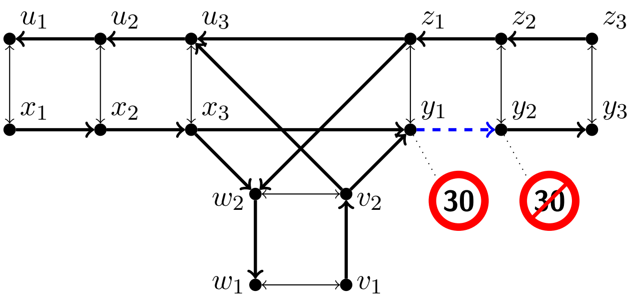


Figure 7.7: Result of an unintuitive strict repair given a correct sign posting for a missing speed limit measure from x_2 to y_2, which can be avoided by measure generation or measure adjustment

Example 61 Figure 7.4 depicts the scenario for a 30 km/h measure from x_2 to y_2 with a consistent, corresponding sign posting. Suppose we are given (in a file `I4.lp`) a the partial scenario which does not include the measure information, i.e. $(G, \emptyset, S)$. If we apply strict repair (under minimal unit cost), we obtain a unique best model, which suggests to add a 30 km/h measure along the edge (y_1, y_2) and the deletion of the start sign at x_2, leading to the scenario shown in Figure 7.7.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I4.lp pool.lp repair.lp -filter=del,add
Best model: {del(s(start(spl(30)),x2)), add(m(spl(30),y1,y2))}
Cost ([Weight:Level]): <[2:1]>
```

The expected generation of three measures has higher unit cost and is therefore not reported. However, we note that it is only due to our data model that the addition of a measure from x_2 to y_2 counts as modification of cost 3, rather than 1. In order to generate the atoms reflecting measure labels, we apply the designated reasoning task and get the desired result, regardless of the number of parts the measure is split into. This is done by replacing `repair.lp` with `m-gen.lp` in the previous call.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I4.lp pool.lp m-gen.lp -filter=del,add
Best model: {add(m(spl(30),x2,x3)), add(m(spl(30),y1,y2)),
add(m(spl(30),x3,y1))}
Cost ([Weight:Level]): <[3:1]>
```

In the dual scenario $(G, M, \emptyset)$, where only these three measures are given as input (`I5.lp`), we can likewise force the generation of the corresponding traffic signs.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I5.lp pool.lp s-gen.lp -filter=del,add
Best model: {add(s(start(spl(30)),x2)), add(s(start(spl(30)),y1)),
add(s(end(spl(30)),y2))}
Cost ([Weight:Level]): <[3:1]>
```

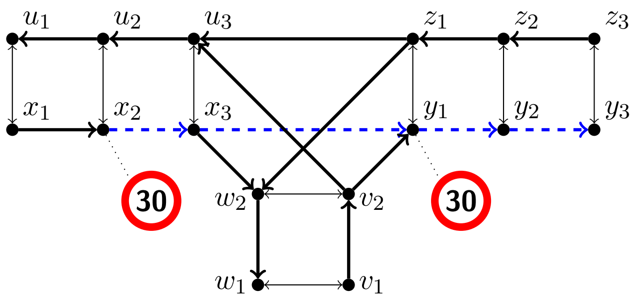


Figure 7.8: Result of an unintuitive strict repair for an unannounced 30 km/h measure from x_2 to y_2, which can be avoided by sign generation or sign adjustment

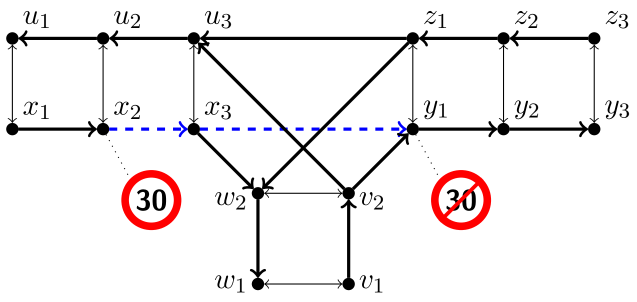


Figure 7.9: Result of another unintuitive strict repair for consistent measure data where sign adjustment is appropriate

Here, the strict repair would also list unintuitive updates which are also minimal according to unit cost. These variants include the deletion of the entire input (fourth answer set) and updates towards the scenarios depicted in Figures 7.8 (first answer set) and 7.9 (third answer set).

```
$ dlv Sp.lp Corr.lp P.lp G.lp I5.lp pool.lp repair.lp -filter=del,add
Best model: {add(s(start(spl(30)),x2)), add(s(start(spl(30)),y1)),
add(m(spl(30),y2,y3))}
Cost ([Weight:Level]): <[3:1]>
Best model: {add(s(start(spl(30)),x2)), add(s(start(spl(30)),y1)),
add(s(end(spl(30)),y2))}
Cost ([Weight:Level]): <[3:1]>
Best model: {del(m(spl(30),y1,y2)), add(s(start(spl(30)),x2)),
add(s(end(spl(30)),y1))}
Cost ([Weight:Level]): <[3:1]>
Best model: {del(m(spl(30),x2,x3)), del(m(spl(30),x3,y1)),
del(m(spl(30),y1,y2))}
Cost ([Weight:Level]): <[3:1]>
```

Hence, in a data adjustment setting, where only one kind of information is given, measure or sign generation is the appropriate variant of the strict repair task. ■

With Answer Set Programming we can easily provide even more flexible support for constraints and preferences. Here, the penalization control for DLV's weak constraints by weights and levels are especially helpful to encode different adequacy criteria, as we show next.

Flexible Preferences

In the sections on adjustment and generation, we discussed special settings in which restricted forms of strict repairs are generally preferable. Both cases illustrated the need for flexibility concerning constraint and preference handling on top of a common core, essentially realized by the effect mapping P (`P.lp`), the conflict specification Sp (`Sp.lp`), and the additional correspondence conflict specification Sp^{Cr} (`Corr.lp`).

From a conceptual point of view, it does not matter whether additional rules are predefined as files or generated on the fly following user interaction. As demonstrated, our framework allows for a modular choice concerning the addition of any new information, including constraints and optimization criteria. With this, we can overcome a natural mismatch between a very formal approach and complex systems like traffic regulations, which include many special cases and context-dependent situations.

There is a trade-off between closeness to reality and the simplicity of rules. If a formal treatment shall remain understandable, it necessarily has to hide subtle aspects implicit in common sense reasoning. With increasing experience, we can incorporate new knowledge by adding further conditions to rules, introducing new rules to the conflict specification, or applying a different adequacy criterion. For the latter option, DLV's weight/level control in weak constraints is particularly useful.

For instance, we may assume that in a given data source traffic measure information is more reliable than traffic sign data. If the measures are inconsistent, sign adjustment cannot be applied. However, we may prefer to change as few measures as possible. In terms of DLV's weak constraints, this amounts to putting measure changes on a *higher level* than sign changes. Among the equally good solutions concerning measures, we can then minimize the changes of traffic signs. We express this repair variant in a new file `m-high.lp`, which prohibits the entailment of any conflict atom and makes use of explicit cost control for weak constraints.

```
:- c(_,_).
:~ del(m(T,X,Y)). [1:2]
:~ add(m(T,X,Y)). [1:2]
:~ del(s(T,X)).   [1:1]
:~ add(s(T,X)).   [1:1]
```

Example 62 Consider the scenario shown in Figure 7.10, where both measures and signs are inconsistent. The input is given by the following atoms, written to `I6.lp`.

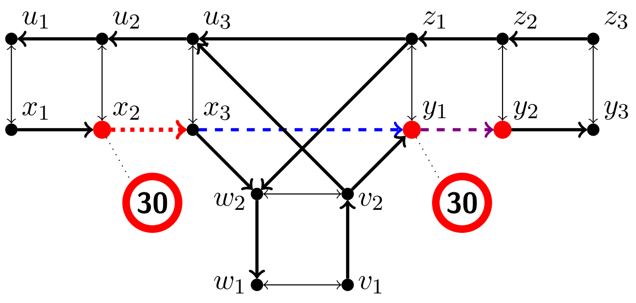


Figure 7.10: Ambiguous speed limit information. In addition to a 30 km/h measure from x_2 to y_1, a 40 km/h measure is given on edges (x_2, x_3) and (y_1, y_2)

```
input(m(spl(30),x2,x3)). input(m(spl(40),x2,x3)).
input(m(spl(30),x3,y1)). input(m(spl(40),y1,y2)).
input(s(start(spl(30)),x2)). input(s(start(spl(30)),y1)).
```

The two measures associated with edge (x_2, x_3) must not overlap, and the start signs do not correspond with the 40 km/h measures. Note that we also have a correspondence conflict at node y_2, since there is a *max-speed*(30) effect on edge (y_2, y_3) according to the start sign at note y_1.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I6.lp pool.lp eval.lp -cautious --
c(T,V)?
overlap(lm,maxspeed(30),maxspeed(40)), x2
unjustified(maxspeed(30)), y1
unjustified(maxspeed(30)), y2
unannounced(maxspeed(40)), x2
unannounced(maxspeed(40)), y1
```

Despite the small size of the scenario, five minimal strict repairs under unit cost are reported due to the ambiguous information.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I6.lp pool.lp repair.lp |
> grep -c "Best model"
5
```

All of these repairs suggest to resolve the conflict at x_2 by deleting the 40 km/h measure. This is reasonable, since the maximum speed effect for 30 km/h is also supported by a traffic sign.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I6.lp pool.lp repair.lp -cautious --
del(X)?
m(spl(40),x2,x3)
```

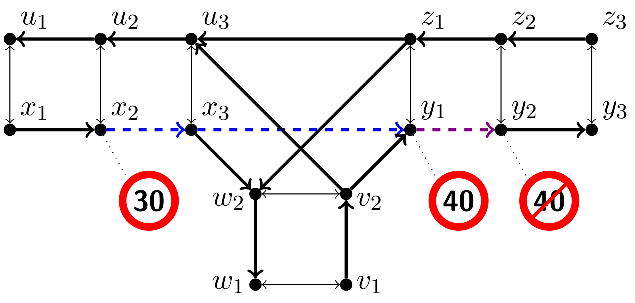


Figure 7.11: Scenario of Figure 7.10 after the unique strict repair, where measures have higher priority than signs

Suppose we consider the measure information to be more trustworthy. Using the constraint and the adequacy criterion defined by `m-high.lp` above, we have only one strict repair, which leads to the scenario shown in Figure 7.11.

```
$ dlv Sp.lp Corr.lp P.lp G.lp I6.lp pool.lp m-high.lp -filter=del,add
Best model: {del(m(spl(40),x2,x3)), del(s(start(spl(30)),y1)),
add(s(start(spl(40)),y1)), add(s(end(spl(40)),y2))}
Cost ([Weight:Level]): <[3:1],[1:2]>
```

Since in this update only one measure is deleted (on edge (x_2, x_3)), weight sum on level 2 is 1. On level 1, we have a weight of 3 since this repair replaces the end signs at y_2 and adds at y_1 the corresponding start sign for the 40 km/h measure. ■

Summary. We have demonstrated how Answer Set Programming allows for a purely declarative approach towards the realization of the advanced reasoning tasks defined in Chapter 5. By instantiating the formal model with according programs, we immediately get consistency evaluation, which is the central task. With little modification, we obtain a uniform, modular approach for all reasoning tasks, including correspondence checking, diagnosis, various predefined, practically relevant repair tasks, as well as a basis for flexible preference handling beyond a robust strict repair that can handle traffic measure and traffic sign data at the same time.

We illustrated some difficulties which arise due to the highly complex nature of traffic regulations and observed some consequences of design choices. In particular, we decided to reduce measures to single edge labels and base the minimality criterion in diagnosis and repairs on counting measure and sign labels. We note that different such criteria might be considered. For instance, we could disallow for the decrease or increase of measure extents by operating on identifiers, which group sets of labels that can only be used (and deleted) together. As another alternative, we might count the changes in effects, rather than measures and signs. Furthermore, we might discern different layers of priorities regarding measure, sign, or effect types and even define special treatment of frequently occurring regulations. For example, we would regard motorways and residential areas as least likely to be incorrect measure data. Contrary, end signs for non-open speed limits might be a special case of particularly untrustworthy input data.

We have demonstrated a conceptual framework into which such domain knowledge and expert judgement can be flexibly encoded. Integration of this logic-based core within a larger application, investigation of practical scalability issues, and evaluation on sets of real-world data remain to be done in future work.

CHAPTER 8

Conclusion

To the best of our knowledge, systematic inconsistency management for traffic regulations has been an entirely unexplored domain prior to this thesis. The presented work was done in cooperation with *PRISMA solutions*[1] to advance their road management web application SKAT, in which traffic measures and traffic signs can be stored and visualized on top of a digital street map.

Government officials in Lower Austria and Vienna are supported in their traffic regulation maintenance tasks by the existing tool. For instance, when a new measure is enacted, it is stored and visualized in the application and the variants for according traffic sign posting are computed. However, the rules for this simple announcement logic do not take into account the context of existing measures or signs. Moreover, there are no methods to evaluate the lawfulness and logical correctness of such data.

At this point the work on this thesis started with a vague goal to find inconsistencies in traffic measure data with respect to the Road Traffic Regulations. Traffic signs were thought of as secondary information to be derived more or less directly from their respective traffic measures. In a comprehensive domain analysis involving around 150 scenarios and 30 traffic measures we found that traffic signs need to be accounted for on the same level as traffic measures, for several reasons. First, combinations of traffic measures influence correct traffic sign posting options and traffic signs can serve as announcement of multiple measures. Moreover, some inconsistencies concern only traffic sign posting and cannot be detected on the level of their associated traffic measures. Finally, a practically important question is whether the current state of traffic sign posting corresponds with the intended restrictions as expressed by active traffic measures.

Based on these observations it became apparent that a mechanism is needed that allows for both separate and combined evaluations of measures and signs. We developed a formal model where these two kinds of inputs are viewed as different languages which both express road use restrictions. Accordingly, we captured the meaning of measures

[1]`http://www.prisma-solutions.at`

and signs in a uniform way by a mapping to so-called *effects* which allowed for their comparison on a semantic level.

Moreover, we introduced the notion of a *conflict* which represents an illegal or undesired situation as defined by the Road Traffic Regulations and supplementary documents, expert knowledge and common sense. The specifications to derive effects from measures and signs, and then conflicts from effects, are assumed to be defined by means of formulas in some form of predicate logic. Throughout this work, we presented examples using Answer Set Programming as underlying logic. We argued in detail why ASP is a suitable choice both theoretically and practically.

On top of the formal model, we defined relevant use cases in form of reasoning tasks. First, consistency evaluation was established by the two-step mapping from measures and signs to conflicts. Next, we examined diagnosis, i.e., finding the causes for derived conflicts. We studied the relation between conflicts and their related input and gave a characterization of according contexts by means of diagnoses. We then defined and investigated in detail the repair task to compute updates of inconsistent scenarios such that the result is free of conflicts. There, we also examined relations between diagnoses and repairs. Furthermore, by additionally requiring correspondence of the restrictions expressed by measures and signs, we introduced so-called strict repairs. Additionally, the practically important use cases of sign/measure adjustment and generation were obtained as special cases of strict repairs, which may be parameterized in a flexible way to derive further uses cases. In particular, the previously existing solution to generate announcement proposals for single measures is now given as special case of sign generation. Notably, all repair and strict repair mechanisms are inherently context-sensitive and can be easily extended.

For our major reasoning tasks we characterized the computational complexity of associated decision problems for different logics. Towards an implementation, we demonstrated how the formal model can be directly instantiated with Answer Set Programming, giving an executable specification. Finally, we developed a systematic approach in which all reasoning tasks are obtained by augmenting the encoded traffic regulation by few additional rules. As a result, we realized an elegant, easily adjustable solution for the logical core of envisaged traffic regulation administration tools that support advanced inconsistency management.

8.1 Future Work

We have developed methods to ensure the correctness of traffic regulation data on a digital street map with respect to flexible criteria. In conclusion, we point towards future work both at the practical and the theoretical side.

Deployment and Evaluation

As a first step, the presented ASP implementation needs to be integrated in an extended version of the existing web application. A mapping must be established that translates between the object-relational models of the actual street and traffic regulation data and the presented formal model. Traffic measures and signs are currently being captured by users of the existing web application. When sufficient data has become available, we can then start to develop an algorithm that generates a street graph due to our definitions based on the street map as stored in the existing database. This involves a discretisation step, since we do not deal with any metrics. We will have to deal with thresholds and approximations to reliably map quantitative and potentially noisy data to our discrete, qualitative model, which is essential for our purposes. For instance, we need to decide how many meters a traffic sign may be posted after a junction to still consider it as being posted *directly* after the junction. However, with according conflict specifications, we assume that the presented work will be of aid in evaluating such assumptions. Moreover, the flexibly adaptable variants of the strict repair task should be useful to automatically fill the gaps of missing data, possibly even in the presence of errors.

Based on this integration we will be able to evaluate our system with real-world data. Furthermore, the user interface must be adapted to the new use cases. Then, a first version of the application can be delivered.

Complex Street Graphs

To focus on the development of inconsistency management tasks, we assumed a minimalistic street graph with a single lane per direction, no roundabouts and no frontage roads. In real-world applications it is important to capture all physical aspects of streets that relate to traffic regulatory information. Since we did not deal with validity restrictions, we also had to omit a discussion of lanes for buses, taxis or bicycles.

Scaling and Contexts

We analyzed the computational complexity of decision problems for our reasoning tasks which are defined on a very abstract level. In real-world data, however, we will often encounter similar situations which may be tackled in a less generic way in order to improve performance.

One notable source of complexity is the search space in the repair task. Thus, it will be important to limit the possibilities which signs and measures are considered for addition. How this can be achieved with intuitive rules is practically challenging. We may try an alternative generation of repairs, however, where the search space is increased stepwise, if needed.

Furthermore, one could study restricted classes of traffic regulation problems or conflicts to investigate lower complexity variants of our reasoning tasks. For instance, the minimal repairs for a pairwise contradiction of two traffic signs are immediate. More generally, to repair overconstrained scenarios we know that additions of new traffic signs need not be considered. Along these lines, it seems appealing to further develop the connection between diagnosis and repair under additional assumptions stemming from practical examples. That is, we could isolate reoccurring conditions from frequent problems of real-world road management scenarios and examine whether it is possible to derive delete-only repairs directly from diagnoses given additional properties of the respective scenario.

This is related with contexts, respectively independent sets of conflicts, as introduced and investigated in Chapter 5. It would be valuable to work out conditions under which local repairs can be merged to obtain global repairs. In reality, geographically distant conflicts will always be independent and their local repairs will also be independent in the sense that their intersections are empty. Suppose there exist two such sets of repairs R_1 and R_2 for respective unrelated sets of conflicts C_1 and C_2. So far, if a user requests repairs for $C_1 \cup C_2$, we can only present the list obtained by the Cartesian product $R_1 \times R_2$. Both from a usability perspective and for improved performance of the repair task we should aim at identifying the unrelated sub-problems first. To this end, we defined the notion of a context as the subset of the input that is relevant for a set of conflicts. Along these lines, we may study under which conditions local repairs, operating on different contexts, can be merged without introducing new conflicts. We also characterized contexts by means of diagnoses. How to efficiently compute contexts, or similar such concepts that allow for parallel inconsistency management, are intriguing theoretical issues of practical relevance.

Validity Restrictions

Validity restrictions, or validities for short, are a practically important topic but technically challenging. By adding a validity restriction to a traffic sign, the set of conditions or road users the effects apply for is narrowed down. If we allow for different categories of validities as shown in Table 3.1 (page 24), we introduce a complex ontology. How to model the involved relations is not straightforward, in particular, if we want to detect implausible combinations of classes. For instance, a driving ban might be restricted to "trucks above 3.5 tons of weight, except school buses." This expression is formed along three independent dimensions: vehicle type, weight and a role in road participation. However, consider the similar restriction "school buses below 3.5 tons of weight, except trucks." It requires a very sophisticated domain model to reliably detect such implausible validity restrictions.

Moreover, all reasoning tasks increase in complexity along these new dimensions. First, we would need to redefine our building blocks of measures, signs, effects and conflicts, where the respective validity restriction is added as further component. Then, a conflict must be derived if an illegal situation occurs for *any* road user or condition. Taking into account validity restrictions thus requires to significantly revise the effect mapping and the conflict specification. As an example, consider two consecutive speed limit start signs along a lane, where the first one imposes a (general) 80 km/h restriction and the second one announces a 60 km/h limit for trucks. This is not a contradiction, since the new announcement overrules the former restriction only for trucks. On the other hand, if a general 60 km/h speed limit is followed by a 80 km/h limit for trucks, it practically has to be detected as conflict since trucks shall not be allowed to drive faster than, e.g., passenger cars. Besides the highly combinatorial nature of validity restrictions, such subtle issues make their modelling for automated reasoning even harder.

Advancement of Answer Set Programming Tools

As a paradigm, Answer Set Programming has been around for over two decades. In recent years, efficient solvers have become available, including DLV [30], Potassco [21], and Cmodels [32]. Yet, ASP has rarely been used outside academia. For wider adoption in industry it has been recognized that better tools, frameworks and libraries are in need to integrate ASP with mainstream programming paradigms and environments [12]. Promising steps in this direction include IDE Support [20, 37] and debugging techniques [22, 36], ASP with external computation sources as provided by dlvhex [16, 18], or including ASP rules within Java code [19].

Our application demonstrates the usefulness of ASP as a declarative paradigm for logic-oriented applications and constraint satisfaction problems. Towards an extension of SKAT, we will need to call an ASP solver from a Java program that will provide additional input atoms. Then, the answer sets (computing conflicts, diagnoses and repairs) must be translated back to Java objects for further processing. To this end, we would benefit from a Java library that (i) allows to target different ASP solvers and (ii) reduces the integration overhead to a minimum by automating the mapping between Java objects and logic predicates representing them.

The current version of SKAT is a successful showcase for this approach to externalize the logic into rule files. Here, the rule engine of the semantic web framework Apache Jena[2] was used for the aforementioned generation of traffic sign proposals for new measures. By annotations of involved Java classes the translation between objects and their RDF representation was defined. Based on these declarations, the jenabean[3] library automated the mapping of according input objects to triples which were then supplied to the engine in addition to a fixed set of custom rules. Finally, objects of designated target classes were likewise created automatically for associated inferred facts.

Given these frameworks, the rule-based encoding of a simple logic gave an advantage over previous Java implementations in terms of readability and automated reasoning. We also made use of a built-in mechanism which allows to call external Java computations. However, as a general rule engine, Jena has two major shortcomings. First, due to its design for semantic web applications, the rule syntax is restricted to triples. Second, the engine provides no mode of reasoning under purely declarative semantics. On the one hand, the order in which rules may fire cannot be controlled. On the other hand, when using the negation built-in, the order of rule firing matters in general. Thus, the meaning of a set of rules which include negation is undefined and the output of computations is unpredictable. This limits the applicability of Jena for advanced reasoning tasks as presented in this thesis.

Here, the answer set semantics provide clear benefits. Negation as failure for non-monotonic reasoning and disjunction to model choices are two attractive features of Answer Set Programming. With increasing availability of tool support and frameworks for software integration ASP may significantly gain traction in industrial applications.

[2]`http://jena.apache.org/`

[3]`https://code.google.com/p/jenabean/`

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[2] Harald Beck, Thomas Eiter, and Thomas Krennwallner. Inconsistency Management for Traffic Regulations. In Biplav Srivastava, Freddy Lécué, and Anupam Joshi, editors, *AAAI 2012 Workshop on Semantic Cities*, pages 2–8. AAAI Press, July 2012.

[3] Harald Beck, Thomas Eiter, and Thomas Krennwallner. Inconsistency Management for Traffic Regulations: Formalization and Complexity Results. In Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors, *13th European Conference on Logics in Artificial Intelligence (JELIA 2012), September 26-28, 2012, Toulouse, France*, volume 7519 of *LNCS*. Springer, September 2012.

[4] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994.

[5] Gerd Brewka, Thomas Eiter, and Miroslaw Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[6] Luca Console, Daniele Theseider Dupré, and Pietro Torasso. A theory of diagnosis for incomplete causal models. In *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 2*, IJCAI'89, pages 1311–1317, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[7] Luca Console and Pietro Torasso. Hypothetical reasoning in causal models. *International Journal of Intelligent Systems*, 5(1):83–124, 1990.

[8] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7:133–141, 1991.

[9] Luca Console and Pietro Torasso. Automated diagnosis. *Intelligenza Artificiale*, 3(1-2):42–48, 2006.

[10] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[11] Johan de Kleer and James Kurien. Fundamentals of model-based diagnosis. In *IFAC Symposium SAFEPROCESS 2003*, pages 25–36. Elsevier, 2003.

[12] Thomas Eiter. SMS and ASP: Hype or TST? In Maria Garcia de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 77–82. Springer, 2008.

[13] Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. Complexity Results for Answer Set Programming with Bounded Predicate Arities. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):123–165, 2007.

[14] Thomas Eiter, Wolfgang Faber, Christoph Koch, Nicola Leone, and Gerald Pfeifer. DLV - a system for declarative problem solving. *CoRR*, cs.AI/0003036, 2000.

[15] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12:12–1, 1999.

[16] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. Conflict-driven ASP solving with external sources. *TPLP*, 12(4-5):659–679, 2012.

[17] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009.

[18] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 90–96. Professional Book Center, 2005.

[19] Onofrio Febbraro, Nicola Leone, Giovanni Grasso, and Francesco Ricca. JASP: A framework for integrating answer set programming with java. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *KR*. AAAI Press, 2012.

[20] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. ASPIDE: Integrated development environment for answer set programming. In James P. Delgrande and Wolfgang Faber, editors, *LPNMR*, volume 6645 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2011.

[21] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[22] Martin Gebser, Jörg Pührer, Torsten Schaub, and Hans Tompits. A meta-programming technique for debugging answer-set programs. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 448–453. AAAI Press, 2008.

[23] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

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

[25] Lane A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989.

[26] Tomi Janhunen and Ilkka Niemelä. Gnt - a solver for disjunctive logic programs. In Lifschitz and Niemelä [33], pages 331–335.

[27] Kurt Konolige. Abduction versus closure in causal theories. *Artif. Intell.*, 53(2-3):255–272, 1992.

[28] Robert A. Kowalski. Algorithm = logic + control. *Commun. ACM*, 22(7):424–436, 1979.

[29] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[30] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7:499–562, 2002.

[31] Harry R. Lewis. Complexity results for classes of quantificational formulas. *J. Comput. Syst. Sci.*, 21(3):317–353, 1980.

[32] Yuliya Lierler and Marco Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In Lifschitz and Niemelä [33], pages 346–350.

[33] Vladimir Lifschitz and Ilkka Niemelä, editors. *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*. Springer, 2004.

[34] Fangzhen Lin and Yuting Zhao. Assat: Computing answer sets of a logic program by sat solvers. In Rina Dechter and Richard S. Sutton, editors, *AAAI/IAAI*, pages 112–118. AAAI Press / The MIT Press, 2002.

[35] Peter Lucas. Symbolic diagnosis and its formalisation. *The Knowledge Engineering Review*, 12:109–146, 1997.

[36] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Catching the ouroboros: On debugging non-ground answer-set programs. *TPLP*, 10(4-6):513–529, 2010.

[37] Johannes Oetsch, Jörg Pührer, and Hans Tompits. The sealion has landed: An IDE for answer-set programming—preliminary report. *CoRR*, abs/1109.3989, 2011.

[38] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[39] David Poole. A logical framework for default reasoning. *Artif. Intell.*, 36(1):27–47, 1988.

[40] David Poole. Representing knowledge for logic-based diagnosis. In *FGCS*, pages 1282–1290, 1988.

[41] David Poole. Normality and faults in logic-based diagnosis. In *IJCAI*, pages 1304–1310, 1989.

[42] Harry E. Pople. On the mechanization of abductive logic. In Nils J. Nilsson, editor, *IJCAI*, pages 147–152. William Kaufmann, 1973.

[43] Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[44] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[45] Tommi Syrjänen and Ilkka Niemelä. The Smodels system. In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczynski, editors, *LPNMR*, volume 2173 of *Lecture Notes in Computer Science*, pages 434–438. Springer, 2001.

[46] Klaus W. Wagner. Bounded query classes. *SIAM J. Comput.*, 19(5):833–846, 1990.

APPENDIX A

Appendix

A.1 Source Code

This section contains the entire DLV source code developed in Chapter 7.

Effect Mapping

```
%%%  P.lp (Section 7.3.2, page 99)  %%%

node(X) :- e(_,X,_).
node(Y) :- e(_,_,Y).

indir(X,Y) :- e(lane,X,Y).
indir(X,Y) :- e(straight,X,Y).

speed(5). speed(10). speed(20). speed(30).
speed(40). speed(50). speed(60). speed(70).
speed(80). speed(90). speed(100). speed(110).
speed(120). speed(130).

lang(ls). lang(lm).

f(F,X,Y) :- fl(L,F,X,Y).

% direct mapping for measure
fl(lm,F,X,Y) :- use(m(T,X,Y)), m2f(T,F).
m2f(spl(K),maxspeed(K)) :- speed(K).
m2f(traffic,nec).
m2f(no_traffic,ban).
```

```
% sign effect: start/end expression
expr_start(F,X) :- use(s(T,X)), start_of(T,F).
expr_end(F,X)   :- use(s(T,X)), end_of(T,F).
end_of(T,F) :- expl_end_of(T,F).
end_of(T,F) :- impl_end_of(T,F).
start_of(start(T),F)  :- m2f(T,F).
expl_end_of(end(T),F) :- m2f(T,F).
impl_end_of(start(spl(K)),maxspeed(J)) :- speed(K), speed(J), K!=J.

% base case and propagation
fl(ls,F,X,Y) :- expr_start(F,X), indir(X,Y).
fl(ls,F,Y,Z) :- fl(ls,F,X,Y), indir(X,Y), indir(Y,Z),
                not block_prop(F,Y).
block_prop(F,Y) :- fl(ls,F,X,Y), indir(Y,Z), expr_end(F,Y).
block_prop(F,Y) :- fl(ls,F,X,Y), e(straight,X,Y), has_perm_inc(ls,Y),
                   not expr_start(F,Y).
has_perm_inc(L,Y) :- e(left,X,Y),  lang(L), not fl(L,ban,X,Y).
has_perm_inc(L,Y) :- e(right,X,Y), lang(L), not fl(L,ban,X,Y).

% non-extending types
block_prop(ban,X) :- node(X).
block_prop(nec,X) :- node(X).

% no_turn
fl(ls,ban,X,Y) :- use(s(no_turn(T),X)), e(T,X,Y).
fl(ls,ban,X,Y) :- use(s(no_turn(left),X)), e(uturn,X,Y).

% mand_turn (omitted compound ones like left_straight)
fl(ls,nec,X,Y) :- use(s(mand_turn(T),X)), e(T,X,Y).
fl(ls,ban,X,Y) :- use(s(mand_turn(T),X)), e(ET,X,Y), T!=ET.

% no_entry
fl(ls,ban,X,Y) :- use(s(no_entry,Y)), e(_,X,Y).
```

Conflict Specification

```
%%%  Sp.lp (Section 7.3.2, page 102)  %%%

% bad announcement ends
c(bad_end(F),Y)  :- indir(X,Y), indir(Y,Z),
                    fl(ls,F,X,Y), not fl(ls,F,Y,Z),
                    needs_expr_end(F), not expr_end(F,Y).
c(cant_end(F),Y) :- use(s(T,Y)), expl_end_of(T,F), indir(X,Y),
                    not fl(ls,F,X,Y).
needs_expr_end(maxspeed(K)) :- speed(K).

% overlap
c(overlap(L,F1,F2),X) :- fl(L,F1,X,Y), fl(L,F2,X,Y), contr(F1,F2).
contr(motorway,residential_area).
contr(maxspeed(K),maxspeed(J)) :- speed(K), speed(J), K<J.
contr(ban,nec).

% no way out
c(no_way_out(lm),X) :- use(m(_,X,_)), not way_out(lm,X).
c(no_way_out(ls),X) :- use(s(_,X)), not way_out(ls,X).

way_out(L,X) :- lang(L), out_node(X).
way_out(L,X) :- lang(L), reachable(L,X,Y), way_out(L,Y).

reachable(L,X,Y) :- lang(L), e(T,X,Y), not fl(L,ban,X,Y), T!=uturn.
reachable(L,X,Y) :- e(uturn,X,Y), fl(L,nec,X,Y).
reachable(L,X,Z) :- reachable(L,X,Y), reachable(L,Y,Z).
```

Correspondence Conflicts

```
%%%  Corr.lp (Section 7.3.3, page 109)  %%%

c(unjustified(F),X) :- fl(ls,F,X,Y), not fl(lm,F,X,Y).
c(unannounced(F),X) :- fl(lm,F,X,Y), not fl(ls,F,X,Y).
```

Pool

```
%%%  pool.lp (Section 7.3.1, page 98, and Section 7.3.3, page 107)  %%%

pool(I) :- input(I).
use(I) v -use(I) :- pool(I).

del(I)  :- -use(I),     input(I).
add(I)  :-  use(I), not input(I).
keep(I) :-  use(I),     input(I).

% proposal of additional signs and measures
pool(s(TS,X))  :- pool(m(TM,X,Y)), m2f(TM,F), start_of(TS,F).
pool(s(TS,Y))  :- pool(m(TM,X,Y)), m2f(TM,F), expl_end_of(TS,F).
pool(m(T,X,Y)) :- pool(s(start(T),X)), indir(X,Y).
pool(m(T,X,Y)) :- pool(s(end(T),Y)), indir(X,Y).
pool(m(T,X,Y)) :- pool(m(T,Z,X)), indir(Z,X), indir(X,Y).
```

Eval

```
%%%  eval.lp (Section 7.3.2, page 101)  %%%

 use(I) :- input(I).
-use(I) :- pool(I), not input(I).
```

Diagnosis

```
%%%  diagnosis.lp (Section 7.3.3, page 105)  %%%

:- add(I).
:~ keep(I).
```

Repair

```
%%%  repair.lp (Section 7.3.3, page 106)  %%%

:- c(_,_).
:~ del(I).
:~ add(I).
```

Adjustment

```
%%%  s-adj.lp (Section 7.3.3, page 112)  %%%

:- c(_,_).

:~ del(s(T,X)).
:~ add(s(T,X)).
:- del(m(T,X,Y)).
:- add(m(T,X,Y)).
```

```
%%%  m-adj.lp (Section 7.3.3, page 112)  %%%

:- c(_,_).
:- del(s(T,X)).
:- add(s(T,X)).
:~ del(m(T,X,Y)).
:~ add(m(T,X,Y)).
```

Generation

```
%%%  s-gen.lp (Section 7.3.3, page 113)  %%%

:- c(_,_).
:- del(s(T,X)).
:~ add(s(T,X)).
:- del(m(T,X,Y)).
:- add(m(T,X,Y)).
```

```
%%%  m-gen.lp (Section 7.3.3, page 114)  %%%
```

```
:- c(_,_).

:- del(s(T,X)).
:- add(s(T,X)).
:- del(m(T,X,Y)).
:~ add(m(T,X,Y)).
```

Flexible Preferences

```
%%%  m-high.lp (Section 7.3.3, page 116)  %%%
```

```
:- c(_,_).
:~ del(m(T,X,Y)). [1:2]
:~ add(m(T,X,Y)). [1:2]
:~ del(s(T,X)).   [1:1]
:~ add(s(T,X)).   [1:1]
```

Graph

```
%%%  G.lp (Section 7.3.2, page 99)  %%%
```

```
e(lane,x1,x2). e(lane,x2,x3). e(straight,x3,y1). e(lane,y1,y2).
e(lane,y2,y3). e(lane,z3,z2). e(lane,z2,z1). e(straight,z1,u3).
e(lane,u3,u2). e(lane,u2,u1). e(lane,w2,w1). e(lane,v1,v2).

e(right,x3,w2). e(right,v2,y1). e(left,v2,u3). e(left,z1,w2).

e(uturn,x1,u1). e(uturn,u1,x1). e(uturn,x2,u2). e(uturn,u2,x2).
e(uturn,x3,u3). e(uturn,u3,x3). e(uturn,y1,z1). e(uturn,z1,y1).
e(uturn,y2,z2). e(uturn,z2,y2). e(uturn,y3,z3). e(uturn,z3,y3).
e(uturn,w2,v2). e(uturn,v2,w2). e(uturn,w1,v1). e(uturn,v1,w1).

in_node(x1).  in_node(v1). in_node(z3).
out_node(u1). out_node(w1). out_node(y3).
```

Input

```
%%%  I1.lp (Section 7.3.2, page 99)  %%%
```

```
input(s(start(spl(30)),x2)).
input(s(end(spl(30)),y2)).
```

```
%%%  I2.lp (Section 7.3.3, page 110)  %%%
```

```
input(m(spl(30),x2,x3)).
input(m(spl(30),x3,y1)).
input(m(spl(30),y1,y2)).
input(m(spl(40),y1,y2)).
input(m(spl(40),y2,y3)).
input(s(start(spl(30)),x2)).
input(s(end(spl(40)),y3)).
```

```
%%%  I3.lp (Section 7.3.3, page 112)  %%%
```

```
input(s(start(spl(40)),y1)).
input(s(end(spl(40)),y2)).
input(m(spl(30),x2,x3)).
input(m(spl(30),x3,y1)).
input(m(spl(40),y1,y2)).
```

```
%%%  I4.lp (Section 7.3.3, page 114)  %%%
```

```
input(s(start(spl(30)),x2)).
input(s(start(spl(30)),y1)).
input(s(end(spl(30)),y2)).
```

```
%%%  I5.lp (Section 7.3.3, page 114)  %%%
```

```
input(m(spl(30),x2,x3)).
input(m(spl(30),x3,y1)).
input(m(spl(30),y1,y2)).
```

```
%%%  I6.lp (Section 7.3.3, page 116)  %%%
```

```
input(m(spl(30),x2,x3)).
input(m(spl(40),x2,x3)).
input(m(spl(30),x3,y1)).
input(m(spl(40),y1,y2)).
input(s(start(spl(30)),x2)).
input(s(start(spl(30)),y1)).
```

A.2 Translation

Across different countries, similar traffic signs and traffic measures are applied. However, there are usually significant differences in the exact meaning of comparable concepts, which makes direct translations impossible. We made use of pragmatic, approximate translations as listed in Table A.1.

Translation	**Original Austrian term**
additional panel	Zusatztafel
design speed	Bauartgeschwindigkeit
frontage road	Nebenfahrbahn
Give Way	Vorrang geben
gross vehicle weight (GVW)	(höchst-) zulässiges Gesamtgewicht
halting ban	Halteverbot
highway	Autobahn
Road Traffic Regulations	Straßenverkehrsordnung (StVO)
home zone	Wohnstraße
lane	Fahrpur
(driving) direction, direction of travel	Fahrtrichtung
motorway	Autostraße
parking ban	Parkverbot
passenger car	Personenkraftwagen (PKW)
pedestrian zone	Fußgängerzone
one-way (street)	Einbahn
Mandatory Left Turn	Vorgeschriebe Fahrtrichtung nach links
Mandatory U-turn	Umkehrgebot
No Entry	Einfahrt verboten
No Left Turn	Abbiegeverbot nach links
No U-turn	Umkehrverbot
No Vehicles	Fahrverbot
residential area	Ortsgebiet
roundabout	Kreisverkehr
traffic measure	Verkehrliche Maßnahme
traffic regulation	Verkehrsvorschrift
traffic regulation order	Verkehrliche Verordnung
traffic sign	Verkehrszeichen
vehicle owner	Fahrzeughalter

Table A.1: Used translations of the original Autrian traffic regulation terms