Semantic Cities
AAAI Technical Report WS-12-13

Inconsistency Management for Traffic Regulations

Harald Beck and **Thomas Eiter** and **Thomas Krennwallner**
Institute of Information Systems, Vienna University of Technology
Favoritenstrasse 9–11, A-1040 Vienna, Austria
{hbeck,eiter,tkren}@kr.tuwien.ac.at

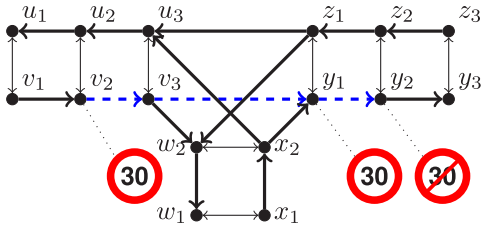
Abstract

Smart Cities is a vision driven by the availability of governmental data that fosters many challenging applications. One of them is the management of inconsistent traffic regulations, i.e., the handling of inconsistent traffic signs and measures in urban areas such as wrong sign posting, or errors in data acquisition in traffic sign administration software. We investigate such inconsistent traffic scenarios and formally model traffic regulations. Based on this, we consider relevant reasoning tasks including consistency testing, diagnosis, and repair, and present an implementation of the these tasks using answer set programming. The results of this research may improve existing governmental software maintaining traffic regulations.

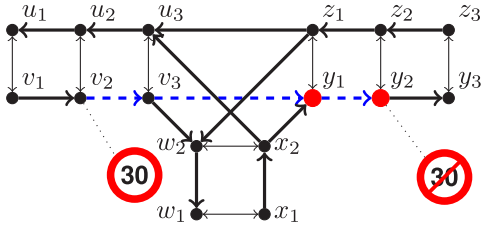
1 Introduction

The advent of the World Wide Web and distributed systems brought numerous new methods for intelligent management of data and knowledge. With initiatives such as Open Government Data,[1] the idea of Smart Cities has been gaining interest in research communities, with many innovative applications in ecological and city planning areas. Local governments manage their posted traffic signs and measures using software tools, i.e., authorities enact rules how traffic on urban streets and places should be regulated, and employees increasingly maintain this information with the help of specialized software. An important task is the management of inconsistent traffic regulations, as illustrated next.

Example 1 Consider the T-junction shown in Fig. 1a. It consists of three arms, each represented by two parallel lanes: u_3 to u_1 and v_1 to v_3, w_2 to w_1 and x_1 to x_2, and y_1 to y_3 and z_3 to z_1. We can turn from one arm to each other arm, and may reverse between nodes that are connected by edges with two arrows. The traffic signs at v_2, y_1, and y_2 symbolize a correct sign posting for a speed limit measure of 30 kmph, indicated by the dashed blue path from v_2 to y_2. The *effect* expressed by both the measure and the signs is that along the edges $(v_2, v_3), (v_3, y_1), (y_1, y_2)$, the maximal allowed speed for any road user is 30 kmph. The recurrent start sign at y_1 is necessary, since road users coming from x_2, turning into the lane starting at y_1, would otherwise be unaware of the speed limit. This situation is depicted in Fig. 1b. The effect of the start sign at v_2 can only be propagated to the arm starting at y_1, because y_1 can also be reached from the arm ending in x_2. We get an inconsistent traffic regulation due to two conflicts: the speed limit effect ends at y_1 without an end sign, and the end sign at y_2 has no associated effect.



(a) Correct sign posting for a speed limit measure



(b) Inconsistency: no recurrence of start sign at y_1

Figure 1: T-junction with 30 kmph speed limit measure

Such inconsistencies create problems in daily traffic. Officials are confronted with legal issues (e.g., challenging of speeding tickets) when two dissenting speed limits are announced. Even more delicate is the aspect of legal responsibility in case of accidents caused by wrong 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. Tools that detect, prohibit, and correct such errors are in need to help public administration with their traffic management tasks.

In order to gain new insights from available sign posting data, formal methods from knowledge representation and reasoning proved to be key to attack issues that arise when data is inconsistent (Poole 1994; Lucas 1997; de Kleer

[1] http://opengovernmentdata.org/

and Kurien 2003). Many issues arise in the context of traffic regulations. Traffic measures, i.e., intended constraints given as regulations on the traffic, may oppose the state of traffic sign posting, which can be seen as real-world constraints that announce what is allowed on the street. One natural question is how to find inconsistencies when combining traffic measures and street signs. Such questions become even more complex in dynamic environments, i.e., when so-called active traffic management comes into play. For instance, variable-message signs on motorways manage the traffic flow by varying speed limits based on events like traffic congestions, or weather conditions like fog or black ice. Contradicting speed limits may be posted by operators of such message signs, leading to aforementioned legal issues.

Finding such errors is not trivial in real life situations and many subtle inconsistencies may occur. When an inconsistency is found, one usually wants to diagnose and repair it. To the best of our knowledge, there is no automated support for inconsistency finding in complex traffic regulations, but the seemingly simple scenario in Example 1 shows the need for (semi-)automatic tool support in traffic regulation maintenance software. Different from the issues above is the problem of modeling transportation and traffic in a formal representation. Legal texts are ambiguous and often implicitly understood, and no single characterization has yet shown to be advantageous over others.

This motivates this work with the following contribution:

- We analyze the problem domain and identify main concepts and notions such as traffic signs, measures, effects, and inconsistencies in traffic regulation orders.
- Building upon well-known literature in abductive reasoning and model-based diagnosis (Poole 1994; Lucas 1997; de Kleer and Kurien 2003), we develop a formal model using predicate logic for traffic signs and measures, and introduce the notion of *traffic regulation problem*, the basis for reasoning tasks such as inconsistency detection, diagnosis, and repair.
- We show how these specifications can be implemented in an elegant way using answer set programming, which gives an executable specification.

This work is embedded in an industrial context dealing with specialized software, which is used by local government departments and allows for the visualization and administration of traffic regulations. The results of this research may assist to find inconsistencies and should give a clear advantage over simple traffic sign acquisition and storage tools.

2 Domain Analysis

In this section we briefly analyze the domain of traffic regulations, traffic measures and traffic signs.

A *traffic regulation* is a legal document that describes how road users can make use of the street and how these usages can be restricted by means of *traffic signs*. The legal act to introduce new traffic signs, or remove existing ones, is 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, that is, a restriction of road usage. This measure has to be *announced* by means of traffic signs and becomes legally effective as soon as the corresponding signs are posted on the street. We view road markings as special cases of traffic signs.

The restrictions described by measures and signs include speed limits, driving bans, parking or halting bans, prohibited or mandatory driving directions, information about zones like residential areas and pedestrian zones, motorways, and so on. We base our work on the Austrian traffic regulation and its potential measures and signs.[2] However, we focus on general aspects that are not bound to regional differences.

Inconsistencies. In general, a set of traffic regulation orders, resp. the resulting measures and signs, can lead to *conflicts* wrt. the traffic regulation. The aim of our work is to *detect* such inconsistencies, to *diagnose* and to *repair* them.

For instance, in Austria it is not allowed that a motorway overlaps with a residential area. In the perspective of traffic signs, it means that, when driving on a motorway, the end sign must precede the start sign of the residential area. In addition to such illustrative cases, complications quickly arise when many different kinds of restrictions are expressed.

What we understand by a conflict does not necessarily stem from the traffic regulation, but can also come from supplementary documents of expert knowledge such as traffic planning experience. It is thus our aim to provide a system that can detect different sorts of conflicts in a modular and easily extendable way. Whenever a conflict is detected, we want to provide the user with a diagnostic information, explaining which measures or signs caused it. Finally, we want to offer a repair mechanism that suggests by which modifications compliance with the specification can be established.

Data Model & Architecture. To achieve these goals, we first need a street model based on which we can express measures and signs, and the restrictions expressed by them. We will view streets 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. Any digital street map from which this view can be generated can be used as potential database.

By an *effect* of both measures and signs we understand the implicit restrictions they express. To reflect measures and signs (from a database or user input) in the street graph, we will use predefined labels on the edges (for measures) and nodes (for signs). Similarly, we will represent arising inconsistencies by associating nodes with specific conflict labels. Both the mapping from measures and signs to effects and from effects to conflicts will be established in a modular way by means of logic formulas. The conflict labeling can be used to visualize inconsistencies on a street map, followed by user interaction in connection with diagnosis and repair.

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

3 Formal Model

In this section we formalize our data model and formulate a traffic regulation probem based on it.

Definition 1 (Street graph) *By a* (street) graph *we understand a connected, labeled, directed graph $G = (V, E, \ell)$ of nodes V, edges $E \subseteq V \times V$, and a labeling function ℓ that assigns each edge $(v, w) \in E$ a unique label $\ell(v, w) \in \{left, straight, right, lane, uturn\}$.*

We identify G with the set of atoms $e(t, v, w)$, where t is the label of the edge (v, w). Several assumptions about the structure of these graphs are made. For instance, we intend to model junctions by means of edges with labels *left*, *right* and *straight*. Whenever (v, w) is such an edge, all incoming edges $(x, v) \in E$ to node v are labeled *uturn* or *lane*.

Example 2 (ctd.) Fig. 1a suggests how the edge labels ought to be used. For instance, the edge (v_3, y_1) models the direction straight ahead over a junction and thus gets the label *straight*. All other edges (v_i, v_{i+1}) and (y_i, y_{i+1}) are labeled with *lane*. The incoming street from below has a turn to the right starting at x_2 and ending at y_1, which will be modeled by an atom $e(right, x_2, y_1)$. Similarly, we use $e(left, x_2, u_3)$ for the left turn at x_2. The edges with arrows on both ends depict U-turns in both directions.

In the formulation of measures in traffic regulation orders concepts like street names, addresses and cardinal points are used to describe the intended topological dimensions. We assume that for the description at hand, a preprocessing (or specification) maps this scope to edges. We thus reduce measure descriptions to sets of such "atomic measures."

To describe measures, signs, their effects, as well as conflicts, we build upon disjoint sets of ground terms M, S, F and C called the *measure types, sign types, effect types* and *conflict types*, respectively. For instance, M may contain a set of terms $spl(k)$ for each speed limit value k that is needed, e.g., $spl(5)$, $spl(10)$, ..., $spl(130)$ in Austria.

Definition 2 (Measures, Signs, Effects, Conflicts) *Given a street graph G, 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 $I_G = M_G \cup S_G$;*
- *Effects $F_G = \{f(t, v, w) \mid t \in \mathsf{F}, (v, w) \in E\}$; and*
- *Conflicts $C_G = \{c(t, v) \mid t \in \mathsf{C}, v \in V\}$.*

For instance, to represent the prohibited case that a motorway overlaps with a residential area at a node v we might use $c(overlap(motorway, residential\text{–}area), v)$. Similarly, the fact that one is caught in a dead end or loop at u can be represented as $c(no\text{-}way\text{-}out, u)$.

Definition 3 (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, $Sc = (G, M, S)$ is called a* scenario.

Example 3 (ctd.) In Fig. 1a, the dashed blue path from v_2 to y_2 symbolizes a speed limit measure of 30 kmph. We formalize this as a set of atomic measures $\{m(spl(30), v_2, v_3), m(spl(30), v_3, y_1), m(spl(30), y_1, y_2)\}$. The depicted traffic signs are defined at nodes as the set $\{s(start(spl(30)), v_1), s(start(spl(30)), y_1), s(end(spl(30)), y_2)\}$.

The meaning of both measures and signs is captured by a mapping of the according languages to a common target language of effects. To assist modular composition, we define $\overline{X}_Y = X \cup \{\neg x \mid x \in Y \setminus X\}$ as the *closed world operator* applied to a set of ground atoms X relative to a *base set* $Y \supseteq X$. We always use the according base set of Definition 2, and thus leave out the subscript, e.g., $\overline{M}$ for a set measures M on G abbreviates $\overline{M}_{M_G}$.

Definition 4 (Effect mapping) *An effect mapping is a set P of formulas in predicate logic that associates with each input $I \subseteq I_G$ on a street graph G the set of atoms*

$$\mathcal{F}^P(G, I) = \{f(t, u, v) \mid P \cup \overline{G} \cup \overline{I} \models f(t, u, v)\} \ ,$$

called effects of I (on G).

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. Based on which effects (and conflicts) shall be defined, the exact logic must be fixed. We note that first-order logic in general will not suffice, as seen in Section 5. Logic programming offers a suitable, more expressive alternative discussed in Section 6.

Example 4 The first-order sentence

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

of an effect mapping P captures the meaning of speed limit (spl) measures. We informally describe when this effect label is obtained by signs: First, an edge (x, y) is labeled with $max\text{-}speed(k)$, if an according start sign $s(start(spl(k), x))$ is placed at x. From there, the effect is propagated in the direction of traffic, i.e., along the edges with label *lane*, until an end sign or a junction is reached. For the latter case, let $e(lane, u', u)$ be the last edge before the junction and $e(straight, u, v)$ be the next edge in the direction ahead. The effect continues after the crossroads on the (unique) edge $e(lane, v, w)$ only if another start sign is posted on v, or no edge (x, v) with label *left* or *right* permitted for traffic exists (and neither an end sign nor the start sign for a different speed limit is there).

The effect mapping uses measures and signs on a graph to derive effects. Likewise, these effect atoms will then be used to infer conflicts by means of a specification.

Definition 5 (Conflict specification) *A* conflict specification over an effect mapping P *is a set Sp of formulas in predicate logic that associates with each input $I \subseteq I_G$ on a street graph G the set of atoms*

$$\mathcal{C}^P_{Sp}(G, I) = \{c(t, v) \mid Sp \cup \overline{G} \cup \overline{\mathcal{F}^P(G, I)} \models c(t, v)\} \ ,$$

called conflicts of I (on G).

Example 5 Fig. 1b depicts the situation in which the intended speed limit is not sufficiently announced. Road users coming from node x_2, turning right into the lane starting at y_1 are not informed about the speed limit. Hence, according to the sign posting, the $max\text{-}speed(30)$ effect cannot be associated with edge (y_1, y_2). Since we have a $max\text{-}speed(30)$ effect until node y_1 but no end sign

mapped to it, we have a conflict which we may represent as $c(\textit{no-end}(\textit{max-speed}(30)), y_1)$. Note that another start sign for a different speed limit would be an implicit end sign for the former. The explicit end sign at y_2 would then lead to a second conflict, as there is no "open" effect anymore: $c(\textit{no-such-to-end}(\textit{max-speed}(30)), y_2)$.

Definition 6 (Traffic Regulation Problem) *Let Sp be a conflict specification over an effect mapping P, and Sc be a scenario. Then, the pair $\Pi = (Sp, P)$ is called a* traffic regulation *and the pair (Π, Sc) a* traffic regulation problem.

4 Reasoning Tasks

We now use the preceding definitions to specify some practically relevant use cases in form of reasoning tasks. We use the shorthand $sc(G, I)$ for the scenario (G, M, S), where $M = I \cap M_G$ and $S = I \cap S_G$. By an *update* of an input I on G we understand a pair (I^-, I^+), where $I^- \subseteq I$ and $I^+ \subseteq I_G \setminus I$. In the sequel, we let $\mathcal{T} = (\Pi, Sc)$ be a traffic regulation problem with a traffic regulation $\Pi = (Sp, P)$ and a scenario $Sc = (G, M, S)$, and $I = M \cup S$.

Definition 7 (Inconsistency) *The* conflicts of $\mathcal{T}$ *are given by $\mathcal{C}(\mathcal{T}) = \mathcal{C}^P_{Sp}(G, I)$. If $\mathcal{C}(\mathcal{T}) \neq \emptyset$, we call $\mathcal{T}$* inconsistent.

Additionally, we call every set of measures or signs $X \subseteq I_G$ on graph G *inconsistent*, if $\mathcal{C}^P_{Sp}(G, X)$ is non-empty.

Given an inconsistent $\mathcal{T}$, we are interested which part of the input, i.e., which hypotheses, explain the conflict observations. Hence we define an abductive diagnosis in line with (Poole 1989; Console and Torasso 2006).

Definition 8 (Diagnosis) *For inconsistent $\mathcal{T}$, a* diagnosis of a set of conflicts $C \subseteq \mathcal{C}(\mathcal{T})$ *is a set $J \subseteq I$, s.t. $C \subseteq \mathcal{C}^P_{Sp}(G, J)$.*

Since I is always a trivial (but non-informative) diagnosis for any set of conflicts, we are interested in (subset-)*minimal* diagnoses. We omit a formal definition of Π serving the forthcoming examples.

Example 6 (ctd.) The missing sign at y_1 leads to two conflicts. The minimal diagnosis for the missing sign $\{c(\textit{no-end}(\textit{max-speed}(30)), y_1)\}$ is $\{s(\textit{start}(\textit{spl}(30)), v_2)\}$. Independently, the other conflict $\{c(\textit{no-such-to-end}(\textit{max-speed}(30)), y_2)\}$ is minimally explained by $\{s(\textit{end}(\textit{spl}(30)), y_2)\}$.

Dually to diagnoses explaining the cause for inconsistency, we next define repairs, which establish consistency for $\mathcal{T}$.

Definition 9 (Repair) *A* repair *for an inconsistent $\mathcal{T}$ is an update (I^-, I^+) of I, if $sc(G, (I \setminus I^-) \cup I^+)$ is consistent.*

Example 7 (ctd.) A minimal repair for the scenario in Example 1 is $(\emptyset, \{s(\textit{start}(\textit{spl}(30), y_1)\})$.

For each of the definitions in this section, we immediately obtain a *reasoning task* which requires the computation of the respective concept. In Section 6 we will sketch how these tasks can be implemented using answer set programming.

Relation between Measures and Signs. We now add to our notion of consistency for traffic regulation problems a criterion requiring measures and signs in a scenario to express the *same* effects, i.e., $\mathcal{F}^P(G, M) = \mathcal{F}^P(G, S)$. If this condition holds, we say that measures and signs *correspond*.

Example 8 (ctd.) In the traffic regulation problem in Fig. 1b, the alternative addition of a no-right turn sign on x_2 is also a minimal repair, since in this case, the node y_1 can only be reached from v_3 (reversing at z_1 along the U-turn (z_1, y_1) is disregarded). Hence, another start sign at y_1 is not necessary and the effect propagation of the start sign at v_2 continues through y_1. However, the prohibition of traffic along the edge (x_2, y_1) as supported by the no-right turn sign is not supported by a corresponding measure. Therefore, the addition of this sign is not a repair in a strict sense.

Based on this observation, we define:

Definition 10 (Strict Repair) *A repair (I^-, I^+) for $\mathcal{T}$ is a* strict repair *for $\mathcal{T}$, if $(G, M', S') = sc(G, (I \setminus I^-) \cup I^+)$ and $\mathcal{F}^P(G, M') = \mathcal{F}^P(G, S')$.*

By additional restrictions on repairs we obtain further practically relevant use cases. We say a repair (X^-, X^+) is X-*based*, if $X^- \cup X^+ \subseteq X$.

Definition 11 (Adjustment) *A* sign adjustment for $\mathcal{T}$, *where M is consistent, is an S_G-based repair (S^-, S^+) for $\mathcal{T}$, such that $\mathcal{F}^P(G, M) = \mathcal{F}^P(G, (S \setminus S^-) \cup S^+)$.*

The definition of a *measure adjustment* similarly assumes S to be consistent and the repair to be M_G-based. Adjustment is useful if one kind of information shall serve as basis for the repair of the other.

Another relevant use case for data import is the generation of measures or signs, given consistent data of the other kind.

Definition 12 (Generation) *A* sign generation for $\mathcal{T}$, *where M is consistent and $S = \emptyset$, is an S_G-based repair $(\emptyset, S^+)$ for $\mathcal{T}$, such that $\mathcal{F}^P(G, M) = \mathcal{F}^P(G, S^+)$.*

A *measure generation* is defined analogously. The additional reasoning tasks which we get from the definition of adjustment and generation are special cases of the repair task. Regardless of whether or which constraints are used, we can rank repairs in different ways. The possibilities range from generic preferences, like favoring deletions over additions, to the encoding of very specific domain knowledge.

5 Case Study

In addition to our running example, we now examine a different problem: the creation of a loop like the one shown in Fig. 2, which is induced by four mandatory left turns.

We consider loops as special dead ends which we want to detect by deriving the conflict $c(\textit{no-way-out}, v)$ whenever a node v has a way in, but no way out. We say a node v has a *way in*, if it is predefined as *in-node* (by means of an according label), or if it is reachable from an in-node. Similarly, a node v has a *way out*, if it is a predefined *out-node*, or an out-node is reachable is from v.

A node w is *reachable* from v, if (i) (v, w) is an edge, where neither the node v is prohibited for traffic (e.g., through a no-entry sign), nor the edge itself (e.g., through a mandatory turn in a different direction), or (ii) if a node x is reachable from v, from which w is reachable.

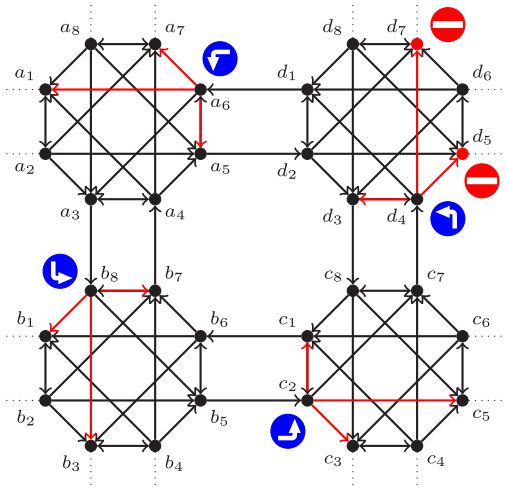


Figure 2: Loop caused by mandatory left turns

Example 9 The mandatory left turns in Fig. 2 induce a loop along the nodes $L = \{a_6, a_3, b_8, b_5, c_2, c_7, d_4, d_1\}$. Respective in-nodes and out-nodes are not depicted and assumed to be reachable from the nodes with dotted lines. For instance, from a_1, an out-node is reachable, and a_2 is reachable from an in-node. Each mandatory left turn prohibits the right turn, U-turn, and edge straight ahead over the junction. E.g., the mandatory left turn at a_6 prohibits moves along the edges (a_6, a_7), (a_6, a_5) and (a_6, a_1).

We try to keep the street model as simple as possible. Reversing along lanes is not a typical road usage. Therefore, we do not model any intermediate nodes along streets (and thus no U-turns within lanes), unless we need to represent a sign, or the start or end of a measure. In reality, we could in principle escape the loop in Example 9 by reversing somewhere along a lane. Since this is not supposed to be necessary, we still want to derive $c(\textit{no-way-out}, v)$ for all nodes $v \in L$.

Diagnosis & Repair. The unique minimal diagnosis for each of these conflicts—which can be seen as one conflict across many nodes—consists of all four mandatory left turns. Note that additional signs that do not restrict the reachability, like speed limits, would not change this diagnosis.

To repair the scenario, we may delete one of the mandatory left turns on nodes a_6, b_8, or c_2. Consider the case that we delete the mandatory left turn at d_4. The options to continue to drive straight over the junctions towards node d_7, as well as turning right towards node d_5, are not available due to the no-entry signs. According to the street model, there is a way out from node d_4 via the U-turn to node d_3, from which an out-node node is reachable via nodes c_8 and c_3. As argued before, it is reasonable to still classify the situation as loop, since paths should not use U-turns. A road user arriving at d_4 for the first time would not know that she will eventually come back to node d_4 (by taking the supposed path).

Challenges. Many conflicts will not be strictly illegal as defined by the traffic regulatory orders or additional legal documents, but arise from expert knowledge or common sense. Consequently, both the inclusion and the kind of definition of many conflicts will be a matter of preference, and shall in principle be configurable by domain experts.

Looking back at the example, one might have different categories of loop conflicts like "strong loop" and "weak loop," where the latter allows for escapes via U-turns. In this sense, Fig. 2 represents a strong loop, the repair where the sign at d_4 is removed represents a weak loop. Further, the question which repairs should be proposed and how to rank them, is very context-sensitive. For instance, we might wish to warn road users about weak loops through a no-through-road sign, but the ideal position to do so is obvious. Alternatively, we could add mandatory U-turn signs before junctions one has to eventually return to. While this might often be the desired solution, it is formally not optimal, since it restricts more than necessary.

6 Implementation

After we derive some desiderata from the aforementioned observations we will present how the introduced reasoning tasks can be implemented using answer set programming.

Since comprehensible specifications play a major role in this problem domain, we demand that the implementation should be *declarative*. An imperative way of programming such rules will quickly lead to deeply nested conditionals with intransparent dependencies. Further, we need a high degree of *modularity* to enable changes to specifications independent of the implementation of reasoning tasks. Since the domain comprises many patterns with exceptions and special cases, some sort of *default reasoning* would be desirable. For instance, traffic is permitted in a certain direction, *unless* it is explicitely prohibited.

Answer Set Programming (ASP) is a rule-based constraint programming methodology gaining increasing populariy (Brewka, Eiter, and Truszczyński 2011). Efficient solvers are available, such as DLV (Leone et al. 2006) and Potassco (Gebser et al. 2011). ASP is based on the answer set semantics (Gelfond and Lifschitz 1991).

We will now sketch how our reasoning tasks can be implemented using (disjunctive) answer set programs, which are sets of rules of the form

$$a_1 \vee \cdots \vee a_\ell \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$$

over first-order atoms. Intuitively, given an interpretation as a set of atoms, if all atoms b_i hold, and none of the atoms c_i provably hold, then at least one of the atoms a_i has to hold. Answer sets are models that satisfy special conditions to rule out self-supported situations in cyclic programs.

First, we show how the reachability relation of Section 5 can be naturally translated into ASP rules. Recall that we wanted to label a node v with $\textit{no-way-out}$, if it has a way in, but *not* a way out. The rule

$$c(\textit{no-way-out}, V) \leftarrow \textit{way-in}(V), \text{not } \textit{way-out}(V).$$

says that if for some node V, $\textit{way-in}(V)$ is derivable but $\textit{way-out}(V)$ is *not*, then we conclude $c(\textit{no-way-out}, V)$. Note that ASP adopts the closed world assumption, enabling such *negation as failure*.

We specify $way\text{-}in$ and $way\text{-}out$ recursively using $reach$, which holds the reachability relation.

$$way\text{-}in(W) \leftarrow way\text{-}in(V), reach(V, W).$$
$$way\text{-}out(V) \leftarrow reach(V, W), way\text{-}out(W).$$

Reachability in turn is defined as follows:

$$reach(V, W) \leftarrow e(T, V, W), \text{not}\, prohib\text{-}n(V), \text{not}\, prohib\text{-}e(V, W).$$
$$reach(V, W) \leftarrow reach(V, X), reach(X, W).$$

Assuming that effect definitions provide the predicates $prohib\text{-}n$ and $prohib\text{-}e$, these five rules in principle suffice to classify dead ends of a graph given as a set of atoms that represent its edges, as in the loop of Fig. 2. Note, however, that this is just a naive implementation for illustration, computing $reach(V, W)$ for all possible pairs of nodes.

We now briefly illustrate the use of ASP to find, diagnose and repair conflicts in our running example of Section 3.

Evaluation. Assume we are given the scenario (G, M, S) as described in Fig. 1b with according edges of form $e(T, V, W)$. The measures are represented by facts of form $input_m(spl(30), X, Y)$, the signs are given by $input_s(start(30), v_2)$ and $input_s(end(30), y_2))$. For brevity, we use predicates $\mathtt{y}$ and $input_{\mathtt{y}}$ to distinguish the input from program-generated measures ($\mathtt{y} = m$) and signs ($\mathtt{y} = s$); the resp. arity is clear from context. The input creates an initial pool of available measures and signs with rules of form

$$\mathtt{y}(\vec{X}) \leftarrow input_{\mathtt{y}}(\vec{X}).$$

Whenever we have an atomic measure $m(T, V, W)$, resp. a sign $s(T, V)$, we may use it or omit it:

$$use_{\mathtt{y}}(\vec{X}) \vee \neg use_{\mathtt{y}}(\vec{X}) \leftarrow \mathtt{y}(\vec{X}).$$

With additional rules, effects of form $f(T, V, W)$ must be derived by measures and signs, in case they are used.

For every effect atom $f(T, V, W)$ derived by a measure, the same effect must be derived by signs. To detect potential correspondence conflicts, we use $supp(BY, f(T, V, W))$ to track the source BY of each effect, i.e., whether it is *supported* by a measure or a sign. One of two similar rules is:

$$c(no\text{-}s(T), V) \leftarrow f(T, V, W), \text{not}\, supp(by\text{-}s, f(T, V, W)).$$

In order to evaluate the input, we use it with rules of form

$$use_{\mathtt{y}}(\vec{X}) \leftarrow input_{\mathtt{y}}(\vec{X}). \quad (1)$$

In summary, we get a single answer set with the conflict $c(no\text{-}s(maxspeed(30)), y_1)$, as $f(maxspeed(30), y_1, y_2)$ is not supported by a sign.

Diagnosis & Repair. Integrity constraints (written with $\bot$ in the head) disallow answer sets that satisfy their bodies, i.e., they express s. By dropping rules (1), we allow input to be unused. We can diagnose a conflict by requiring that it is derived, i.e., by prohibiting that it is *not* derived:

$$\bot \leftarrow \text{not}\, c(no\text{-}s(maxspeed(30)), y_1).$$

We now get multiple answer sets that all contain the atom $c(no\text{-}s(maxspeed(30)), y_1, y_2)$. The minimal one with respect to the positive occurrences of the symbol use contains only one such, namely $use(m(spl(30), y_1, y_2))$.

ASP solvers come with useful optimization features that allow to express preferences. DLV for instance minimizes the number of violated *weak constraints* in a program (distinguishable by $:\sim$ from other rules), resulting in preferred answer sets w.r.t. an optimization criterion. With the additions of the following weak constraints, the aforementioned diagnosis is now the single one, as we require that explanations of a certain conflict are minimal.

$$:\sim input(m(T, V, W)), use(m(T, V, W)).$$
$$:\sim input(s(T, V)), use(s(T, V)).$$

Implementing the repair task works similarly. We now constrain the usage of the input such that *no* conflict is derived, by using the constraint $\bot \leftarrow c(T, V)$. The optimization to obtain a minimal number of *deletions* uses the two previous rules above, where use is negated. We can also consider repairs including the addition of new measures and signs which then have to be introduced during the repair process. This can be done by encoding domain knowledge about how measures have to be announced.

Generation, Adjustment. If we replace the weak constraints for the penalization of measure modifications by according constraints, we get sign adjustments by using the same repair mechanism. If we additionally do not load the given signs, we get sign generation, and so on.

Preference Handling. More sophisticated preferences can be encoded in DLV, by using levels for penalties, which are minimized by priority. This way we can, for instance, prefer modifications of signs over modifications of measures, and in further refinement, deletions over additions.

7 Conclusion and Outlook

To date, tools for advanced inconsistency management of traffic regulations are lacking. We presented a logic-based approach to this problem, which is especially relevant in envisaged future dynamic regulation settings. We formalized the notion of traffic regulation problem and considered major reasoning tasks on it. We implemented them using answer set programming, which serves as an executable specification.

Current and future work includes refining and enhancing the prototype in cooperation with domain experts, as well as integrating street data from relevant real-world scenarios which are currently collected in an ongoing industrial project. Based on the refined implementation, we will conduct an experimental evaluation to assess the viability of our approach in practice. Initial experiments on a number of synthetic scenarios are encouraging.

Acknowledgements. Supported by PRISMA solutions EDV-Dienstleistungen GmbH, and the Austrian Science Fund (FWF) projects P20841 and P24090.

References

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Console, L., and Torasso, P. 2006. Automated diagnosis. *Intelligenza Artificiale* 3(1-2):42–48.

de Kleer, J., and Kurien, J. 2003. Fundamentals of model-based diagnosis. In *SAFEPROCESS'03*, 25–36. Elsevier.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. T. 2011. Potassco: The Potsdam answer set solving collection. *AI Comm.* 24(2):107–124.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *Next Generat. Comput.* 9(3–4):365–386.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL* 7(3):499–562.

Lucas, P. 1997. Symbolic diagnosis and its formalisation. *Knowl. Eng. Rev.* 12:109–146.

Poole, D. 1989. Normality and faults in logic-based diagnosis. In *IJCAI'89*, 1304–1310.

Poole, D. 1994. Representing diagnosis knowledge. *Ann. Math. Artif. Intell.* 11:33–50.