Resolving Conflicts in Action Descriptions

Thomas Eiter and **Esra Erdem** and **Michael Fink** and **Ján Senko**
Institut für Informationssysteme 184/3,
Technische Universität Wien
Favoritenstraße 9-11, 1040 Wien, Austria
email: {eiter, esra, michael, jan}@kr.tuwien.ac.at

Abstract

We study the problem of resolving conflicts between an action description and a set of conditions (possibly obtained from observations), in the context of action languages. In this formal framework, the meaning of an action description can be represented by a transition diagram—a directed graph whose nodes correspond to states and whose edges correspond to transitions describing action occurrences. This allows us to characterize conflicts by means of states and transitions of the given action description that violate some given conditions. We introduce a basic method to resolve such conflicts by modifying the action description, and discuss how the user can be supported in obtaining more preferred solutions. For that, we identify helpful questions the user may ask (e.g., which specific parts of the action description cause a conflict with some given condition), and we provide answers to them using properties of action descriptions and transition diagrams. Finally, we discuss the computational complexity of these questions in terms of related decision problems.

Introduction

Action languages (Gelfond & Lifschitz 1998) are a formal tool for reasoning about actions, where an agent's knowledge about a domain in question is represented by a declarative action description that consists of logical formulas. Consider for instance a light bulb with a switch. When the light is off, then toggling the switch turns the light on; this can be expressed in the action description language $\mathcal{C}$ (Giunchiglia & Lifschitz 1998) by the formula

$$\textbf{caused } Light \textbf{ after } Toggle \wedge \neg Light. \quad (1)$$

On the other hand, at every state, if the light bulb is broken then the light is off. This can be expressed by the formula

$$\textbf{caused } \neg Light \textbf{ if } Broken. \quad (2)$$

Other pieces of knowledge, like laws of inertia, may be also included. The meaning of such an action description can be represented by a transition diagram—a directed graph whose nodes correspond to the states of the world and the edges to the transitions describing action occurrences. For instance, the transition diagram of the action description above (consisting of (1), (2), and the inertia laws) is presented in Figure 1.

Note that the action description above is "buggy", since the effects of toggling the switch are not completely specified. Our goal is to "repair" such descriptions taking into account some additional information, such as observations or axioms about the action domain, which can be represented in an action query language (Gelfond & Lifschitz 1998). For example, when the light bulb is broken, toggling the switch may lead to a state where the light is off; this information is possibly obtained from some observations of the agent, and can be expressed in an action query language, e.g., by the statement

$$\textbf{possibly } \neg Light \textbf{ after } Toggle \textbf{ if } Broken. \quad (3)$$

Some of the additional information may conflict with the action description. For instance, condition (3) does not hold relative to the action description above, since at the state where the light bulb is broken and the light is off, toggling the light switch is not possible. Thus, there is a conflict between the action description and this condition.

In this paper, we consider such conflicts, and how the agent's action description can be modified to resolve them. This may be accomplished in many different ways, and there is no canonical method which works satisfactorily in all cases. According to (Eiter *et al.* 2005), one might aim at dropping a smallest set of candidate formulas to resolve the conflict. In our example, dropping (1) would work. However, under further conditions, like

$$\textbf{necessarily } \neg Light \textbf{ after } Toggle \textbf{ if } Light, \quad (4)$$

the conflict cannot be resolved just by dropping formulas: removing any of (1), (2) and inertia laws will not lead to an edge from a state where the light is on to a state where the light is off. A refined approach is needed which, semantically, modifies the transition diagram by suitable changes of the formulas to "repair" the action description such that the given queries (i.e., conditions) hold.

This paper makes two main contributions in this direction:

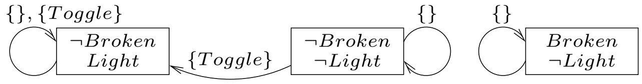

Figure 1: Transition diagram of the action description { (1), (2), (7) }.

1. It provides a precise *notion of conflict* between an action description and a set of queries, and presents a basic algorithm to resolve such conflicts. The idea is to modify the transition diagram of the action description by adding or deleting transitions so that all given conditions are satisfied; such a modification of the transition diagram is possible by adding, deleting or modifying some formulas in the action description. Based on this idea, our algorithm calculates a repair whenever it is possible.
2. Intuitive repair preferences might be difficult to formalize (e.g., both syntactic and semantic aspects might play a role) and thus to achieve with the basic algorithm above. In such cases, the designer might want to *ask questions* about the action description, the transition diagram, and the extra information, whose answers could guide her to come up with an appealing repair in an iterative repair process. For that, we explore several kinds of such questions and determine properties of action descriptions, transition diagrams, and extra information which are helpful in answering them. We also analyze the computational complexity of related problems.

Preliminaries

Transition diagrams. We start with a *propositional action signature* $\mathcal{L} = \langle \mathbf{F}, \mathbf{A} \rangle$ that consists of a set $\mathbf{F}$ of fluent names, and a set $\mathbf{A}$ of action names. Satisfaction of a propositional formula G over atoms $At \subseteq \mathbf{F} \cup \mathbf{A}$ by an interpretation $P \mapsto I(P) \in \{t, f\}$ for all $P \in At$ as usual, is denoted by $I \models G$. An *action* is an interpretation of $\mathbf{A}$, denoted by the set of action names that are mapped to t.

A *transition diagram* of $\mathcal{L}$ consists of a set S of *states*, a function $V : \mathbf{F} \times S \to \{f, t\}$ such that each state s in S is uniquely identified by the interpretation $P \mapsto V(P, s)$, for all $P \in \mathbf{F}$, and a subset $R \subseteq S \times 2^{\mathbf{A}} \times S$ of *transitions*. We say that $V(P, s)$ is the *value* of P in s. The states s' such that $\langle s, A, s' \rangle \in R$ are the possible *results of the execution* of the action A in the state s. We can think of a transition diagram as a labeled directed graph. Every state s is represented by a vertex labeled with the function $P \mapsto V(P, s)$ from fluent names to truth values; we denote by s the set of fluent literals satisfied by this function. Each triple $\langle s, A, s' \rangle \in R$ is represented by an edge from s to s' and labeled A. See Figure 1 for an example.

Action descriptions. We consider a subset of the action description language $\mathcal{C}$ (Giunchiglia & Lifschitz 1998) that consists of two kinds of expressions (called *causal laws*): *static laws*

$$\textbf{caused } L \textbf{ if } G, \tag{5}$$

where L is a fluent literal or *False*, and G is a propositional combination of fluent names; and *dynamic laws* of the form

$$\textbf{caused } L \textbf{ if } G \textbf{ after } U, \tag{6}$$

where L and G are as above, and U is a propositional combination of fluent names and action names. In (5) and (6) the part **if** G can be dropped if G is $True$.[1] An *action description* is a set of causal laws. For instance, one formalization of the light domain described in the introduction can be expressed in this language by the causal laws (1), (2), and the inertia laws

$$\begin{array}{l}\textbf{inertial } Light, \neg Light \\ \textbf{inertial } Broken, \neg Broken.\end{array} \tag{7}$$

Here an expression of the form **inertial** $L_1, ..., L_k$ stands for the causal laws **caused** L_i **if** L_i **after** L_i for $i \in \{1, ..., k\}$.

The meaning of an action description can be represented by a transition diagram as follows. We say that a causal law l is *applicable* to a transition $\langle s, A, s' \rangle$ in a transition diagram, if

- l is a static law (5), such that $s' \models G$; or
- l is a dynamic law (6), such that $s' \models G$ and $s \cup A \models U$.[2]

We denote by $D(tr)$ the set of all causal laws in an action description D that are applicable to a transition tr; by $H_D(tr)$ the set of the heads of all causal laws in $D(tr)$; and by $sat(H_D(tr))$ the set of interpretations of $\mathbf{F}$ that satisfy $H_D(tr)$.

Let D be an action description with an action signature $\mathcal{L} = \langle \mathbf{F}, \mathbf{A} \rangle$. Then the transition diagram $\langle S, V, R \rangle$ *described* by D, denoted $T(D)$, is defined as follows:

- S is the set of all interpretations s of $\mathbf{F}$ such that, for every static law (5) in D, $s \models G \supset L$,
- $V(P, s) = s(P)$,
- R is the set consisting of all transitions $\langle s, A, s' \rangle$ such that $sat(H_D(\langle s, A, s' \rangle)) = \{s'\}$.

We denote by $S(D)$ (resp. $R(D)$) the set of states (resp. transitions) of $T(D)$. For instance the transition diagram described by the action description consisting of (1), (2), (7) is shown in Figure 1.

Conditions. For expressing extra conditions, we consider here a language with two kinds of statements ("queries") about an action description: *possibility queries* and *necessity queries* of the respective forms

$$\textbf{possibly } \psi \textbf{ after } A \textbf{ if } \phi \tag{8}$$

[1] $True$ (resp. $False$) is the empty conjunction (resp. disjunction).

[2] We identify states s with the interpretations $P \mapsto V(P, s)$.

$$\textbf{necessarily } \psi \textbf{ after } A \textbf{ if } \phi \qquad (9)$$

where ϕ and ψ are propositional combinations of fluent names, and A is an action. These queries are syntactically different from the ones presented in (Gelfond & Lifschitz 1998) and (Eiter *et al.* 2005); on the other hand, semantically they constitute a fragment of an extension of the action query language $\mathcal{P}$ (Gelfond & Lifschitz 1998) (from which we draw the term "query") and the condition language in (Eiter *et al.* 2005) (see Related and Further Work for a discussion).

A query q of form (8) (resp., (9)) *is satisfied at state s in a transition diagram T*, denoted $T, s \models q$, if either $s \not\models \phi$, or for some (resp., every) transition $\langle s, A, s' \rangle$ of T $s' \models \psi$ holds. We say that T *entails* a set Q of queries (denoted $T \models Q$), if $T, s \models q$ for every $q \in Q$ and for every state s in T. Accordingly, an action description D *entails* Q (denoted $D \models Q$) if $T(D) \models Q$.

Example 1 Let us consider the light domain described in the introduction as our running example. Let D be the action description consisting of (2), (1), and (7); and Q be the set of two queries: possibility query (3) and the necessity query (4), denoted by q_p and q_n respectively. Figure 1 shows $T(D)$ (i.e., the transition diagram of D). Then it can be easily verified that, at state $\{\neg Light, Broken\}$, since there is no transition from this state with action $Toggle$, the query q_n is trivially satisfied while q_p is not satisfied.

What a query describes is different from what a causal law does: action descriptions allow us to describe a transition diagram, based on causal explanations (what "is caused"), whereas queries allow us to state assertions (what "holds") about transition diagrams. These assertions may, e.g., be observations or axioms about the action domain. (See (Gelfond & Lifschitz 1998) for a discussion on action query languages.)

Conflicts in Action Descriptions

Given an action description D and a set Q of queries, we say that there is a *conflict* between D and Q, if $D \not\models Q$. Our goal is to resolve these conflicts by modifying the action description.

Conflicts can be characterized, from a semantic point of view, in terms of states and transitions "violating" some queries. We assume that the states of the world are correctly described by the given action description. Thus conflicts are existing transitions (for the violation of a necessity query) and non-existing transitions (for the violation of a possibility query) that cause such conflicts. The idea is then to "repair" an action description by a syntactic modification, such as adding, deleting, or modifying some of its causal laws, so that the detected conflicts are resolved by adding and/or deleting some transitions in the transition diagram.

For an action description D and a set Q of queries, the states and transitions violating possibility and necessity queries in Q, respectively, are as follows.

- A state s of $T(D)$ *violates* a possibility query q of form (8) in Q, if $T(D), s \not\models q$.
- A transition $tr = \langle s, A, s' \rangle$ of $T(D)$ *violates* a necessity query q of form (9) in Q (denoted $tr \not\models q$), if $s \models \phi$ and $s' \not\models \psi$.

Example 2 (cont'd) From $T(D)$ we can identify the following conflicts: the single state violating the possibility query q_p is $\{\neg Light, Broken\}$, and the single transition violating the necessity query q_n is $\langle \{Light, \neg Broken\}, \{Toggle\}, \{Light, \neg Broken\} \rangle$.

Since we suppose that states of the world are correctly described by D, we do not need to modify the static laws in D for a repair.

A Method for Resolving Conflicts

Under the assumption above, we can resolve conflicts between an action description D and a set Q of queries by the algorithm presented in Figure 2. Before we explain how this algorithm works, let us describe the notation used in it.

For a set Q of queries, we denote by Q_p (resp. Q_n) the set of possibility (resp. necessity) queries in Q. Then

$$conf_p(D,Q) = \{(q,s) \mid q \in Q_p, s \in S(D), T(D), s \not\models q\}$$
$$conf_n(D,Q) = \{(q,tr) \mid q \in Q_n, tr \in R(D), tr \not\models q\}.$$

For any triple $tr = \langle s, A, s' \rangle$, where s and s' are states and A is an action, and a dynamic causal law of form $l = \textbf{caused } L \textbf{ if } U \textbf{ after } G$, $\langle s, A, s' \rangle \models l$ if either l is not applicable to tr, or $s' \models L$.

A *repair item* is an expression of form $(modify, l, l')$, or (add, l), where l and l' are dynamic causal laws. A *repair* is a set of repair items. For an action description D and a repair M, we denote by $M(D)$ the action description obtained from D by applying the modifications specified by the repair items in a repair M: (add, l) modifies D by adding l; $(modify, l, l')$ modifies D by replacing l with l'; all repair items are executed in parallel, i.e., if M comprises several *modify* items for the same law l, all corresponding modifications l' are generated and eventually replace l. The repairs used by the algorithm RESOLVE(D,Q) are as follows (in causal laws, a state s stands for $\bigwedge_{L \in s} L$, and an action A for $\bigwedge_{X \in A} X \wedge \bigwedge_{X \in \mathbf{A} \setminus A} \neg X$):

$$Delete(\langle s, A, s' \rangle) = \{(add, \textbf{caused } False \textbf{ if } s' \textbf{ after } A \wedge s)\}$$

$$\begin{aligned} Insert(Tr, D) = & \{(add, \textbf{caused } L \textbf{ if } s' \textbf{ after } A \wedge s) \mid \langle s,A,s' \rangle \in Tr, L \in s'\} \\ \cup & \{(modify, l, \textbf{caused } L \textbf{ if } G \textbf{ after } U \wedge \alpha(Tr,l)), \\ & (modify, l, \textbf{caused } L \textbf{ if } G \wedge L \textbf{ after } U \wedge A \wedge s) \mid \\ & l = \textbf{caused } L \textbf{ if } G \textbf{ after } U, l \in D, \\ & \langle s,A,s' \rangle \in Tr, \langle s,A,s' \rangle \not\models l\} \end{aligned}$$

Algorithm RESOLVE(D,Q) : *Mod*, *Incon*

Input: An action description, D, and a set of queries, Q.
Output: A repair, *Mod*, and a set of queries, *Incon*.

$Mod := \emptyset$; $\quad Incon := \emptyset$;
for all $(q, tr) \in conf_n(D,Q)$ **do**
  $Mod := Mod \cup Delete(tr)$;
$D' := Mod(D)$; $\quad Ins := \emptyset$;
for all $(q,s) \in conf_p(D',Q)$ **do**
  ($q =$ **possibly** ψ **after** A **if** ϕ)
  $Cands := \{\langle s,A,s'\rangle \mid s' \in S(D), s' \models \psi, \langle s,A,s'\rangle \models q', \forall q' \in Q_n\}$;
  if ($Cands \neq \emptyset$) **then**
    select $tr \in Cands$;
    $Ins := Ins \cup \{tr\}$;
  else
    $Incon := Incon \cup \{q\}$;
return $Mod \cup Insert(Ins, D'), Incon$;

Figure 2: An algorithm to resolve conflicts.

where $\alpha(Tr,l) = \bigwedge_{\langle s,A,s'\rangle \in Tr, \langle s,A,s'\rangle \not\models l} \neg A \vee \neg s$.

In the algorithm above, first every transition tr violating the necessity queries in Q is removed, by adding to D the causal laws $Delete(tr)$. The new action description, D', entails Q_n. Then, for each state s violating a possibility query $q =$ **possibly** ψ **after** A **if** ϕ in Q relative to D', a set *Cands* of transition candidates tr (triples of form $\langle s,A,s'\rangle$ where $s' \in S(D)$) that, when added to $T(D')$, would satisfy q at s (i.e., $s' \models \psi$) but not violate any necessity queries in Q (i.e., $tr \models q', \forall q' \in Q_n$), is computed. If such transition candidates exist (i.e., $Cands \neq \emptyset$), by introducing only one of these candidates into $T(D')$, the violation of q at s is prevented; otherwise no repair of D exists for Q (i.e., *Incon* is not empty, and it contains the possibility queries that conflict with some necessity query in Q). The set *Ins* denotes all the transition candidates to be introduced into $T(D')$ so that no possibility query is violated in any state. Adding *Ins* to $T(D')$ can be achieved by adding to D' the causal laws $Insert(Ins, D')$.

Theorem 1 *For any repair Mod and set Incon of queries output by* RESOLVE(D,Q)*, the following hold:*

1. *$D \models Q$ iff $Mod = \emptyset$ and $Incon = \emptyset$;*
2. *$Incon = \emptyset$ iff $\exists D'$ such that $S(D) = S(D')$ and $D' \models Q$;*
3. *if $Incon = \emptyset$, then $Mod(D) \models Q$.*

The selection of a transition candidate $tr \in Cands$ for repairing a possibility query constitutes a choice point of the algorithm, where further heuristics can be employed to prune the set of repairs. We could, e.g., prefer transition candidates that *respect inertia conditions* or compute *minimal modifications*, i.e., repairs such that the modifications to $T(D)$ are minimal w.r.t. addition or deletion of transitions.

Example 3 (cont'd) Stipulating preference of transition candidates that respect inertia, the basic method resolves the conflicts as follows. First, the only transition violating q_n (i.e., $\langle s_1, \{Toggle\}, s_1\rangle$, where $s_1 = \{Light, \neg Broken\}$) is deleted from $T(D)$ by adding the law:

caused $False$ **if** $Light \wedge \neg Broken$ **after** $Toggle \wedge Light \wedge \neg Broken$.

Then, to resolve conflicts with q_p, the only transition candidate respecting inertia (i.e., $\{\langle s_2, \{Toggle\}, s_2\rangle\}$, where $s_2 = \{\neg Light, Broken\}$) is introduced into $T(D')$ by replacing (1) with the laws:

caused $\neg Light$ **if** $\neg Light \wedge Broken$ **after** $Toggle \wedge \neg Light \wedge Broken$,
caused $Broken$ **if** $\neg Light \wedge Broken$ **after** $Toggle \wedge \neg Light \wedge Broken$,
caused $Light$ **if** $Light$ **after** $Toggle \wedge \neg Light \wedge Broken$,
caused $Light$ **after** $Toggle \wedge \neg Light \wedge \neg Broken$.

We remark that algorithm RESOLVE can be implemented to use polynomial work space, producing its output, which is exponential in general, as a stream. After computing RESOLVE(D,Q), to get a more concise description, one may drop redundant causal laws that might have been introduced (e.g., (6) where $U \equiv False$), and apply some equivalence preserving transformations (e.g., replacing two laws **caused** L **after** $A \wedge U$ and **caused** L **after** $A \wedge \neg U$ with **caused** L **after** A.) Note also, that if there exists a repair for D, then there always also exists a repair D' of polynomial size. Informally speaking, D' can be obtained by expressing all necessity queries as dynamic laws and dispensing causality for all actions occurring in queries (**caused** L **if** L **after** A, for every literal L). Such a repair is independent of D apart from static laws and semantically it amounts to a complete transition graph w.r.t. actions occurring in queries modulo transitions violating necessity queries. Thus, it is even less appealing than solutions computed by RESOLVE(D,Q), which aim at making modifications as local as possible on single transitions (in order to retain the original semantics of D as much as possible even in case of further modifications). In most cases, however, neither of these basic repairs will be satisfactory. This motivates the utilization of additional knowledge of certain properties for repair.

Towards User-Assisted Repairs

With the method described above we can automatically repair an action description D with respect to a set Q of queries, under the assumption that the states of the world are described correctly by D. However, we may end up with an action description with many causal laws, some possibly redundant or implausible. To get a more appealing description most often requires respecting additional knowledge or intuitions of the *designer* about the action description.

Usually, this knowledge cannot be easily formalized, as the following example illustrates:

Example 4 The designer of D might use her knowledge about the domain, i.e., light bulbs and switches, to infer from the conflict with the observation expressed in q_n that the duality of the toggle action has not been modeled correctly, and that the conflict with q_p is due to neglecting the effects of toggling when the bulb is broken. Hence, instead of D, she might consider D' consisting of (2), (7), and:

$$\begin{array}{l}\textbf{caused } \mathit{Light} \textbf{ after } \mathit{Toggle} \wedge \neg \mathit{Light} \wedge \neg \mathit{Broken} \\ \textbf{caused } \neg \mathit{Light} \textbf{ after } \mathit{Toggle} \wedge \mathit{Light} \wedge \neg \mathit{Broken}.\end{array} \quad (10)$$

Note that this description is more concise and plausible than the one generated by the basic method (see Example 3).

For (interactively) providing support to a designer repairing an action description, we present some questions that she may ask about Q, D, and $T(D)$. Answers to these questions are obtained from useful properties of queries, action descriptions, and transition diagrams.

Questions about queries and causal laws. To better understand the reasons for conflicts, the designer may want to check the given queries Q make sense with each other. Then the question is:

> **D1:** If Q is contradictory relative to D, which queries in Q are contradictory?

We understand contradiction in a set Q as follows:

Definition 1 *A set Q of queries is* contradictory *relative to an action description D, if there is no action description D' such that $S(D) = S(D')$ and $D' \models Q$.*

With an answer to **D1**, the designer may drop contradictory queries from Q. Here are some sufficient conditions to find these queries.

Proposition 1 *A set Q of queries is contradictory relative to D, if Q includes some query (8) such that some $s \in S(D)$ satisfies ϕ, but no $s \in S(D)$ satisfies ψ.*

Proposition 2 *A set Q of queries is contradictory relative to D, if Q includes a necessity query* **necessarily** ψ' **after** A **if** ϕ' *and a possibility query (8) such that some state in $S(D)$ satisfies $\phi \wedge \phi'$, but no state in $S(D)$ satisfies $\psi \wedge \psi'$.*

Example 5 In our running example (i.e., Example 1), if Q had contained the query **possibly** $\mathit{Light} \wedge \mathit{Broken}$ **after** Toggle **if** True then, due to Proposition 1, Q would be contradictory relative to D.

If the given set of queries is not contradictory, then she may ask:

> **D2:** If D does not satisfy a particular necessity query q in Q, which dynamic causal laws in D violate q?

We understand violation of a query as follows:

Definition 2 *A dynamic causal law $l \in D$* violates *a given necessity query q, if there is a transition $tc = \langle s, A, s' \rangle$ in $T(D)$ such that tc violates q, l is applicable to tc, and s' satisfies the head of l.*

Once the designer finds out which causal laws violate which queries, she may want to repair the action description in a way that some causal laws (e.g., the inertia laws) are not modified at all:

> **D3:** Can we resolve a conflict between D and Q, without modifying a set D_0 of causal laws in D?

To answer **D3** the following definition and proposition are helpful.

Definition 3 *A transition diagram T* satisfies *a set D of causal laws (denoted $T \models D$), if, for each transition $tc = \langle s, A, s' \rangle$ in T, for each causal law $l \in D$, l is not applicable to tc or s' satisfies the head of l.*

Proposition 3 *Let D be an action description, and Q be a set of queries. If there exists a transition diagram T such that $T \models D$ and $T \models Q$, then there exists an action description D', such that $S(D) = S(D')$, $D \subseteq D'$ and $T = T(D')$.*

With this proposition, we can answer **D3** by checking if any transition diagram, having states $S(D)$, that satisfies D_0 also entails Q.

Example 6 In our running example it is possible to repair D without modifying the inertia laws: there exists an action description containing the inertia laws and satisfying the given queries (cf. Example 4).

In another scenario, the designer may suspect that the definition of a particular fluent causes problems, so she may want to know whether some particular laws have to be modified in order to obtain a repair:

> **D4:** Do we have to modify a set D_0 of dynamic causal laws in D to resolve a conflict between D and Q?

For this, due to the proposition below, we can check whether none of the transition diagrams, with the same states as D (and thus as D_0), that satisfy D_0, entails Q.

Proposition 4 *Let D_0 be an action description, and Q be a set of queries. If there exists an action description D, such that $S(D_0) = S(D)$, $D_0 \subseteq D$ and $D \models Q$, then there exists a transition diagram T such that $T \models D_0$ and $T \models Q$.*

Questions about states and transitions. Alternatively, the designer may want to extract some information from $T(D)$. For instance, an answer to the following question gives information about states violating a query q in Q:

> **T1:** Which states of $T(D)$ that satisfy a given formula ϕ', violate q?

Example 7 In Example 1, if we just consider states where the light is on (i.e., $\phi' = Light$), then the only state at which a query of Q is violated is $\{Light, \neg Broken\}$.

An answer to the following question gives information about transitions violating a necessity query q in Q:

> **T2:** Given formulas ψ' and ϕ', which transitions $\langle s, A, s' \rangle$ of $T(D)$ such that s satisfies ϕ' and s' satisfies ψ', violate q?

With such information extracted from the transition diagram, the designer might decide *how* to modify the action description D.

Suppose that D does not satisfy a possibility query (8) in Q. The designer may want to learn about possible transition candidates that, when added to $T(D)$ by modifying the definition of some literal L in D, might lead to a repair:

> **T3:** Given a literal L, for every state s of $T(D)$ such that s satisfies ϕ, is there some under-specified transition candidate $tc = \langle s, A, s' \rangle$ for D such that s' satisfies $\psi \wedge L$ and L is under-specified relative to tc? If there is, then what are they?

Here under-specification is understood as follows:

Definition 4 *A transition candidate $tc = \langle s, A, s' \rangle$ for D is* under-specified*, if $\{s'\} \subset sat(H_D(tc))$. A literal L is* under-specified *relative to a transition candidate tc, if $\{L, \overline{L}\} \cap H_D(tc) = \emptyset$.*

With a positive answer to **T3**, the designer may try to modify the description D, for instance, by adding the law **caused** L **if** ψ **after** $A \wedge \phi$.

Complexity Results

In this section, we consider computational aspects of the problems in the previous section and report complexity results for associated decision problems, respectively existence problems.

First let us remind the following result from (Eiter *et al.* 2005): Given an action description D and a set Q of queries, deciding $D \models Q$ is Π_2^p-complete in general. Note that, when Q contains the single query **possibly** $True$ **after** A **if** $True$, which expresses the executability of an action A at every state, this result conforms with the ones reported in (Turner 2002; Lang, Lin, & Marquis 2003).

In the following, we formally state two central results and, informally discuss how to obtain further results. The first main result is about the existence of a conflict resolution between an action description D and Q without modifying a subset D_0 of D.

Table 1: Complexity results (completeness) for problems **D1–D4**, **T1–T3**.

Problem	**D1**	**D2**	**D3**	**D4**	**T1**	**T2**	**T3**
	Σ_2^p	NP	Π_2^p	Σ_2^p	Σ_2^p	NP	Π_2^p
$Q_n = \emptyset$	$\mathrm{P}_{\parallel}^{\mathrm{NP}}$	$O(1)$	Π_2^p	Σ_2^p	Σ_2^p	$O(1)$	Π_2^p
$Q_p = \emptyset$	$O(1)$	NP	$O(1)$	$O(1)$	NP	NP	Π_2^p

Theorem 2 *Given D, Q, and $D_0 \subseteq D$, deciding if there exists some D', such that $S(D){=}S(D')$, $D_0{\subseteq}D'$, and $D' \models Q$, is Π_2^p-complete.*

We can show Π_2^p-hardness even for $D_0 = \emptyset$; for such D_0, complexity drops only if in addition Q is restricted to queries of form (8) (to $\mathrm{P}_{\parallel}^{\mathrm{NP}}$-completeness, i.e., polynomial time with parallel queries to an NP-oracle, see, e.g., (Johnson 1990)).

The second main result is about the existence of a conflict resolution between an action description D and Q without modifying the transition diagram described by D.

Theorem 3 *Given D and Q, deciding if there exists some D', such that $S(D){=}S(D')$, $D' \models Q$, and $R(D){\subseteq}R(D')$, is Π_2^p-complete.*

We remark that if some repair of D for Q is known to exist, then deciding the above problem is coNP-complete.

Table 1 shows complexity results for the decision problems resp. existence problems related to the questions above (denoted **D1–D4**, resp. **T1–T3**) for the general case, and when $Q_n{=}\emptyset$, or $Q_p{=}\emptyset$.

Deciding whether Q is contradictory w.r.t. D (**D1**) is Σ_2^p-complete in general. Intuitively, this is because deciding the violation of a possibility query q is Σ_2^p-complete. We have to guess a violating state and verify, by means of an NP-oracle, for corresponding transition candidates that they do not satisfy q. Since we can express necessity queries by dynamic causal laws, this source of complexity carries over to deciding whether a set of (mixed) queries Q is contradictory. From these observations, Σ_2^p-completeness of the existence version of **T1** (i.e., whether such a state exists) is straightforward. However, if $Q_n{=}\emptyset$, to show that Q is contradictory w.r.t. D, it is sufficient to test whether, for some query (8) in Q_p, some state satisfies ϕ but no state satisfies ψ. This amounts to a Boolean combination of SAT instances, whose evaluation is in $\mathrm{P}_{\parallel}^{\mathrm{NP}}$. For $Q_p{=}\emptyset$, note that a set Q_n of necessity queries cannot be contradictory.

On the other hand, deciding whether a necessity query q is violated is in NP: Guess and verify in polynomial time a transition violating q. Thus, e.g., deciding whether a causal law $l \in D$ violates $q \in Q_n$ (i.e., **D2**) is NP-complete, as well as the existence version of **T2**.

D3 is the problem considered in Theorem 2, **D4** is the complementary problem, and corresponding results have been discussed above. Finally, the property of **T3** fails if there exists a state s satisfying ϕ, such that no underspecified transition candidate $tc = \langle s, A, s' \rangle$ for D exists, such that $s' \models \psi \wedge L$. Since for a given s, this can be checked with an NP-oracle, failure of the property is in Σ_2^p.

Related and Further Work

In (Eiter *et al.* 2005), the authors describe a method to minimally modify an action description, when new causal laws are added, by deleting some causal laws, so that given queries are satisfied. In the method above, we obtain an action description by adding or modifying some causal laws, motivated by some reasons for conflicts. For some problems, as discussed in the introduction, just dropping causal laws as in (Eiter *et al.* 2005) does not lead to a solution, whereas our method above does.

Similar to (Eiter *et al.* 2005), (Sakama & Inoue 2003) discusses how to minimally update a logic program syntactically so that given observations are satisfied. A semantical approach to updating a logic program by changes to Kripke structures (which are related to transition diagrams) is given in (Sefránek 2000), but no conditions are considered. In (Zhang, Foo, & Wang 2005) the authors describe how to resolve conflicts between a logic program and a set of constraints by "forgetting" some atoms in the program; in (Zhang & Foo 2005), they describe how logic programs can be updated following this approach.

That an action description can be transformed into a logic program (resp. a propositional theory) (Lifschitz & Turner 1999) might suggest applying update approaches for logic programming mentioned above (resp. for propositional logic (Winslett 1990; Katsuno & Mendelzon 1991)), when applicable (there is no given condition, the action description is inconsistent, etc.), and then obtaining an action description from the respective output. However, such transformations (to and from action descriptions) may lose information about the causal structure of the action domain and yield large and unintuitive action descriptions. In our work, we aim at preserving the causal structure, and keeping the action description intuitive and concise.

In (Balduccini & Gelfond 2003), the authors extend an action description, encoded as a logic program, with "consistency restoring" rules, so that when the action description and given observations are incompatible, these rules can be "applied" to get some consistent answer set. This, however, is more geared towards handling exceptions. Lifschitz describes in (Lifschitz 2000) an action domain in language $\mathcal{C}$ such that every causal law is defeasible (by means of an abnormality predicate). Then, to formulate some other variations of the domain (e.g., to satisfy some observations), the agent can just add new causal laws. Some of these laws are to "disable" some existing causal laws. In (Balduccini & Gelfond 2003) and (Lifschitz 2000), the causal laws of the original domain description are not modified.

Ongoing and future work includes an implementation of the method described above for resolving conflicts, and the investigation of the use of a SAT solver or an answer set solver to answer the questions discussed above (as suggested by the computational complexity results of the corresponding decision problems, presented in Table 1). Furthermore, (Eiter *et al.* 2005) employs a richer language for conditions, in which like in an extension of action query language $\mathcal{P}$ (Gelfond & Lifschitz 1998), e.g., conditions on sequences of action occurrences can be expressed. However, "repair" of such conditions is not immediate (e.g., many possibilities exist to eliminate "bad" trajectories from the transition diagram in general). This remains for future study.

Acknowledgments

We thank anonymous referees for comments and suggestions on a draft of this paper. This work is supported by the Austrian Science Fund (FWF) grant P16536-N04.

References

Balduccini, M., and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *Working notes of AAAI Spring Symposium*, 9–18.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2005. Updating action domain descriptions. In *Proc. of IJCAI-05*, 418–423.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *ETAI* 3:195–210.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proc. AAAI*, 623–630.

Johnson, D. S. 1990. A catalog of complexity classes. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science*, volume A. MIT Press, Cambridge, MA. chapter 2, 67–161.

Katsuno, H., and Mendelzon, A. O. 1991. On the difference between updating a knowledge base and revising it. In *Proc. KR*, 387–394.

Lang, J.; Lin, F.; and Marquis, P. 2003. Causal theories of action: A computational core. In *Proc. of IJCAI*, 1073–1078.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Proc. LPNMR*, 92–106.

Lifschitz, V. 2000. Missionaries and cannibals in the causal calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, 85–96.

Sakama, C., and Inoue, K. 2003. An abductive framework for computing knowledge base updates. *TPLP* 3(6):671–713.

Sefránek, J. 2000. A Kripkean semantics for dynamic logic programming. In Parigot, M., and Voronkov, A., eds., *Proc. 7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000), Reunion Island, France, November 11-12, 2000*, volume 1955 of *Lecture Notes in Computer Science*, 469–486. Springer.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02)*, 111–124.

Winslett, M. 1990. *Updating Logical Databases*. Cambridge University Press.

Zhang, Y., and Foo, N. Y. 2005. A unified framework for representing logic program updates. In Veloso, M. M., and Kambhampati, S., eds., *Proceedings Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, 707–713. AAAI Press AAAI Press / The MIT Press.

Zhang, Y.; Foo, N.; and Wang, K. 2005. Solving logic program conflicts through strong and weak forgettings. In *Proc. IJCAI-05*, 627–632.