# A Tool for Answering Queries on Action Descriptions ⋆

Thomas Eiter, Michael Fink, and Ján Senko

Institute of Information Systems, Vienna University of Technology, Vienna, Austria, Email: (eiter | michael | jan)@kr.tuwien.ac.at

## 1 Introduction

Action languages [1] 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. For instance, consider 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 $\mathcal{C}$ [2] by the dynamic causal law:

$$\textbf{caused } Light \textbf{ after } Toggle \wedge \neg Light. \tag{1}$$

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

$$\textbf{caused } \neg Light \textbf{ if } Broken. \tag{2}$$

Other pieces of knowledge, like laws of inertia, may be also included:

$$\textbf{inertial } Light, \neg Light, Broken, \neg Broken.\ ^{1} \tag{3}$$

The meaning of such an action description, $D$, can be represented by a transition diagram, $T(D)$—a directed graph whose nodes correspond to the states of the world, $S(D)$, and the edges to the transitions, $R(D)$, describing action occurrences. For instance, the transition diagram of the above action description is shown in Figure 1. [2]

We consider the problem of revising action descriptions in the presence of conflicts between the action description and a set of conditions (axioms or observations) represented in an action query language [1]. For example, when the light bulb is broken, toggling the switch may lead to a state where the light is off; this is expressed by:

$$\textbf{possibly } \neg Light \textbf{ after } Toggle \textbf{ if } Broken. \tag{4}$$

Since at the state where the light bulb is broken and the light is off, toggling the light switch is not possible, There is a conflict between the action description and this condition. Moreover, under further conditions, like the following query:

$$\textbf{necessarily } \neg Light \textbf{ after } Toggle \textbf{ if } Light, \tag{5}$$

the conflict cannot be resolved just by dropping laws. In general, it is difficult to formalize the process of arriving at appealing "repairs", which often depend on additional knowledge or intuitions of the designer. We aim at supporting a designer in conflict and modification analysis and developed a tool that allows a user to issue a number of

⋆ Work supported by the Austrian Science Fund (FWF) under grant P16536-N04.

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

[2] The action description is "buggy" (the effects of toggling the switch are improperly described).

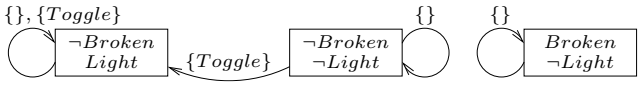


**Fig. 1.** Transition diagram of the action description { (1), (2), (3) }.

relevant tests on an action description in $\mathcal{C}$ and its associated transition diagram in the presence of conditions. The tool computes answers to these tests by answer-set programming, revealing possible causes of conflicts or effects of certain modifications.

## 2 Tests

The tests a designer can issue by our tool, resemble the questions about a set of queries (conditions), $Q$, and $D$, respectively $T(D)$, as identified in [3] (see also below). Although the formal statement of these questions served as the basis for implementing a corresponding test library for the system, we confine here to an informal treatment and refer to [3] for details. As there, focusing on dynamic aspects, $S(D)$ is assumed to be correct and hence static laws need not be modified.

*Tests on queries and causal laws.* To better understand the reasons for conflicts, the designer may want to check whether the given queries $Q$ make sense with respect to each other, find out which causal laws violate certain queries, or whether repairing an action description can be done without modifying some causal laws, resp. whether certain causal laws need to be modified:

- **D1:** Is $Q$ contradictory relative to $D$?
- **D2:** If $D$ does not satisfy a particular **necessarily**-query $q$ in $Q$, which dynamic causal laws in $D$ violate $q$?
- **D3:** Can we resolve a conflict between $D$ and $Q$, without modifying a set $D_0$ of causal laws in $D$?
- **D4:** Do we have to modify a set $D_0$ of dynamic causal laws in $D$ to resolve a conflict between $D$ and $Q$?

*Example 1.* In our running example, if $Q$ consisted of the query **possibly** $Light \wedge Broken$ **after** $Toggle$ **if** $True$ then, $Q$ would be contradictory relative to $D$ (no state in $S(D)$ satisfies $Light$ and $Broken$), while $Q = \{(4)(5)\}$, is not contradictory (**D1**).

*Tests on states and transitions.* Alternatively, the designer may want to extract information from $T(D)$. For instance, information about states, respectively transitions, violating a query $q$ in $Q$, or information about candidates for transitions, that do not constitute transitions due to *under-specification* (i.e., not every fluent is causally explained):

- **T1:** Which states of $T(D)$ that satisfy a given formula $\phi$, violate $q$?
- **T2:** Given formulas $\psi$ and $\phi$, which transitions $\langle s, A, s' \rangle$ of $T(D)$ such that $s$ satisfies $\phi$ and $s'$ satisfies $\psi$, violate $q$?
- **T3:** Given a literal $L$, for every state $s$ of $T(D)$ such that $s$ satisfies $\phi$, 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 $tc$?

**T4:** Which transition candidates $tc = \langle s, A, s' \rangle$ for $D$ such that $s$ satisfies $\phi$ and $s'$ satisfies $\psi$ are under-specified?

*Example 2.* In Ex. 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 $\{Light, \neg Broken\}$ (**T1**).

## 3 Implementation

To compute test answers, we use disjunctive logic programming (DLP) – disjunction is actually needed due to $\Sigma_2^p$-completeness of most of the tests [3]. We translate action description, queries, and test into a DL program, and call the DLP solver DLV [3] to compute the models of this program, which encode the answer of the test performed.

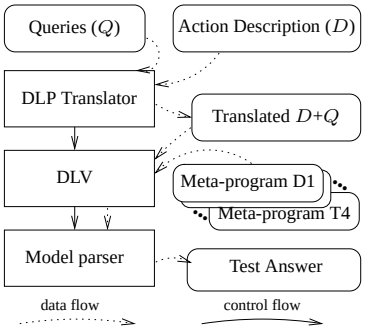


**Fig. 2.** Tool Architecture.

The translation of an action description and queries into a logic program is uniform for all tests, and each test has been encoded in a 'meta-program' which operates on these translations, i.e., input programs. Figure 2 depicts the architecture of our tool. It is a command-line oriented Perl script consisting of two main parts: the *DLP Translator* and a *Model parser*. After pre-processing the input and translation to a DLP, calls to DLV are executed and their output is post-processed into human-readable form by the Model parser. The tool operation is controlled by the first command-line parameter that specifies the type of test to perform (e.g. `-T1`). The remaining parameters are supposed to be input files, i.e., text files, where each line either starts with one of the following keywords:

`Action/Fluent:` declares a new action or fluent literal;
`Inertial/Caused:` describes an inertia, static or dynamic law;
`Possibly/Necessarily:` describes a respective type of query;
`Initial/Successor:` describes a condition on a state ($\phi$ and $\psi$ in tests);

or, otherwise, is directly copied to the output (e.g., to add background knowledge).

The *DLP Translator* compiles the input $D$ and $Q$ into a DLP representation on a file, which is combined with the fixed meta-program for the issued test to a single program on which DLV is invoked. The output of DLV (i.e., the answer sets) is then processed by the Model parser.

Model parsing is specific for each test: some tests yield yes/no answers by means of inconsistency (no model). E.g., no model for test **D1** means that the queries are not contradictory, whereas for some tests models encode test results such as violating states (**T1**,**T3**), violating transitions (**T2**), dynamic causal laws that violate a query (**D2**), etc.

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

While this information is encoded in DLP atoms, the Model parser prints the essential information in a human-readable format.

For user convenience, our implementation allows for generic statements as shortcuts in an action description using fluents and rules with parameters (i.e., variables). E.g., to extend our example to multiple light bulbs, one may re-write (1) as:

`caused light(X) after toggle, -light(X) requires bulb(X).`

where the keyword `requires` marks type information for variable $X$, provided by the background, e.g.: `bulb(green). bulb(yellow). bulb(red).`

## 4 Usage of the System

We now demonstrate a possible session of a designer using our tool. We assume that the action description consisting of (1), (2), and (3) is provided in a file `example.in`, and that the queries (4) and (5) are in files `example.pos` and `example.nec`, respectively.

First, the designer wants to check whether a query is violated at all (**T1** and **T2**):

```
./ad-query -T1 example.in example.pos
  VIOLATING STATE: broken. −light.
./ad-query -T2 example.in example.nec
  VIOLATING TRANSITION: (−broken. light.), (−broken. light.)
```

Since both queries are violated, she wonders whether they are contradictory (**D1**):

```
./ad-query -D1 example.in example.nec example.pos
  The set of queries is not contradictory.
```

Thus, the action description can be repaired such that both queries are satisfied. However, is it inevitable to modify the existing causal laws (issue **D4** with $D_0 = D$)?

Because the answer is 'yes' ((4) is violated at state $(broken, -light)$), she might ask whether at least the inertia laws can be kept by running test **D3** with $D_0 = D - \{(1)\}$. From the answer, 'yes', she eventually knows that the dynamic causal law (1) has to be modified (indeed, this law does not properly reflect the effects when the bulb is broken).

## 5 Conclusion

Our tool ad-query, which is available at `www.kr.tuwien.ac.at/research/ad-query/`, is to our knowledge the first tool to answer queries on action descriptions in $\mathcal{C}$ in the context of revision and design as described. The current version implements a common fragment of $\mathcal{C}$ and queries (heads of laws are literals and other formulas are conjunctions of literals). Ongoing work will extend the language and consider additional tests, as well as a methodology for using the tool.

## References


1. Gelfond, M., Lifschitz, V.: Action languages. Electronic Transactions on Artificial Intelligence **3** (1998) 195–210
2. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proc. AAAI '98, AAAI Press (1998) 623–630
3. Eiter, T., Erdem, E., Fink, M., Senko, J.: Resolving conflicts in action descriptions. In: Proc. ECAI 2006. See `http://www.kr.tuwien.ac.at/research/ecai06.pdf`.