Ann Math Artif Intell (2007) 50:273–304
DOI 10.1007/s10472-007-9077-y

Comparing action descriptions based on semantic preferences

Thomas Eiter · Esra Erdem · Michael Fink · Ján Senko

Published online: 8 August 2007

Abstract The focus of this paper is on action domain descriptions whose meaning can be represented by transition diagrams. We introduce several semantic measures to compare such action descriptions, based on preferences over possible states of the world and preferences over some given conditions (observations, assertions, etc.) about the domain, as well as the probabilities of possible transitions. This preference information is used to assemble a weight which is assigned to an action description. As applications of this approach, we study updating action descriptions and identifying elaboration tolerant action descriptions, with respect to some given conditions. With a semantic approach based on preferences, not only, for some problems, we get more plausible solutions, but also, for some problems without any solutions due to too strong conditions, we can identify which conditions to relax to obtain a solution. We further study computational issues, and give a characterization of the computational complexity of computing the semantic measures.

Keywords Action domain descriptions · Semantic preferences · Transition diagrams

Mathematics Subject Classifications (2000) 68T30 · 68T27

T. Eiter (✉) · M. Fink · J. Senko
Institute of Information Systems, Vienna University of Technology, Vienna, Austria
e-mail: eiter@kr.tuwien.ac.at

M. Fink
e-mail: michael@kr.tuwien.ac.at

J. Senko
e-mail: jan@kr.tuwien.ac.at

E. Erdem
Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul, Turkey
e-mail: esraerdem@sabanciuniv.edu

1 Introduction

Action languages [1] have been introduced for representing knowledge about actions (including their effects) by means of action descriptions, which consist of sentences in a logic whose meaning can be represented by transition diagrams – directed graphs whose nodes correspond to states and whose edges correspond to transitions caused by occurrences and non-occurrences of actions. They provide a domain-independent tool for expressing this knowledge in a highly declarative manner, on which various reasoning tasks (including planning) can be carried out. However, a particular action description is always a (more or less) accurate model of a domain, in the sense that it is a correct abstraction of the modeled world, but as such never "complete," i.e., different such descriptions are possible.

This raises the issue of how to compare different action descriptions. This task is important for applications where certain descriptions are more preferred than others. One such application is the action description update problem [2–4]: when updating an action description with respect to some given information, usually several possibilities exist and one has to choose one of these action descriptions. Another application is related to representing an action domain in a more elaboration tolerant way [4–6]. Recall that, according to McCarthy [5], a "formalism is elaboration tolerant to the extent that it is convenient to modify [...] the formalism to take into account new phenomena or changed circumstances." Applied to action descriptions, this problem can be expressed as the task of identifying among several action descriptions representing the same action domain, which one is the most elaboration tolerant one, with respect to some given conditions describing possible elaborations?

The preference of an action description may be based on a syntactic measure, such as the number of formulas: the less the number of formulas contained in an action description, the more preferred it is. A syntactic measure can be defined also in terms of set containment with respect to a given action description D: an action description is more preferred if it is a maximal set, among others, that is contained in D. For instance, according to the syntactic measure used in [2] for updating an action description D with some new knowledge Q, an action description D' is more preferred if D' is a maximal set, among others, containing D and contained in $D \cup Q$.

In this paper, we describe the preference of an action description on a semantic basis, with respect to some given conditions. The idea is to describe a semantic measure by assigning weights (i.e., real numbers) to action descriptions, with respect to their transition diagrams and some given conditions; then, once the weights of action descriptions are computed, to compare two descriptions by comparing their weights.

We consider action descriptions in a fragment of the action language $\mathcal{C}$ [7], which consists of "causal laws." For instance, consider a (simplified) TV system with the possible actions of pushing the power button on the TV, or pushing the power button of a remote control. In this domain, the causal law

$$\textbf{caused } \mathit{PowerON} \textbf{ after } \mathit{PushPB}_{TV} \wedge \neg \mathit{PowerON}, \tag{1}$$

expresses that the action PushPB_{TV} causes the value of the fluent $\mathit{PowerON}$ to change from f (false) to t (true); such causal laws describe direct effects of actions. The causal law

$$\textbf{caused } \mathit{TvON} \textbf{ if } \mathit{PowerON}, \tag{2}$$

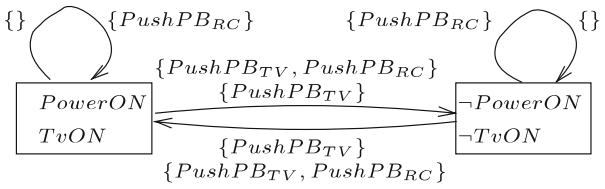

Fig. 1 A transition diagram

expresses that if the fluent *PowerON* is caused to be true, then the fluent *TvON* is caused to be true as well; such causal laws describe state constraints. The meaning of an action description D can be represented by a transition diagram, like in Fig. 1. In this transition diagram, the nodes of the graph (shown by boxes) denote the states of the world: (s) one where both the power and the TV is on, and (s') the other where both the power and the TV is off. The edges denote action occurrences. For instance, the edge from s to s' labeled by the action of pushing the power button on the TV describes that executing this action at s leads to s'. The edges labeled by the empty set are due to the law of inertia.

Suppose that we are given another action description D' describing the domain above; and that the transition diagram of D' is almost the same as that of D, except that there is no outgoing edge from the state {*PowerON*, *TvON*} with the label {$PushPB_{RC}$}. Which action description should be preferred? To answer this question, we assign weights to these two action descriptions, based on their transition diagrams, and given conditions (observations, assertions, etc.).

We describe conditions in an action query language, like in [1], by means of "queries." For instance,

$$\textbf{ALWAYS} \bigvee\nolimits_{A \in 2^{\mathbf{A}}} \textbf{executable}\ A, \tag{3}$$

where $2^{\mathbf{A}}$ denotes the set of all actions, expresses that, at every state, there is some action executable. The query

$$\textbf{SOMETIMES evolves}\ PowerON; \{PushPB_{RC}\}; PowerON \tag{4}$$

expresses that, at some state when the power is on, pushing the power button on the remote control does not turn the power off. Then we can define the weight of an action description as the number of queries it entails. For instance, according to the transition diagram of D, (3) and (4) are entailed, so the weight of D is 2; according to the transition diagram of D', only (3) is entailed, so the weight of D' is 1. Therefore, D is preferred over D'.

The main question we study is the following: *Given a set $\mathcal{D}$ of action descriptions and a set C of queries, which action description in $\mathcal{D}$ is a most preferred one with respect to C?* We address this question taking a semantically-oriented stance, that is by assigning weights to action descriptions in $\mathcal{D}$, based on their transition diagrams and the given conditions. Providing different means for expressing preference by means of weights over the essential elements, i.e., states, possible transitions, and conditions, we obtain four different approaches which are convenient in different scenarios. Mirroring the respective focus of preference assignment, they are intuitively suited for situations where the dynamics of a system is disregarded and static requirements dominate, for situations where dynamic aspects come to the fore and are assessed by the probability of transitions and sequences thereof (i.e., trajectories), or for

situations where the given conditions are central, serving the purpose of a system specification, respectively. We apply these approaches to the problem of updating an action description, as well as to the problem of identifying more elaboration tolerant action descriptions, and observe the following two benefits. First, if a problem has many solutions with the syntactic approach of [2], a semantic approach can be used to further prune the set of admissible solutions and may, eventually, allow to pick one. Second, if a problem does not have any solution with any of the approaches, due to too strong conditions, a semantic approach can be used to identify which conditions could be relaxed in order to find a solution.

As for the organization of the remainder of this article, in the next two sections we briefly introduce the languages – syntax and semantics – we use to encode an action domain by means of an action description, respectively to express conditions on such an action description by means of action queries. In Section 4 we introduce weights for encoding semantic preference of action descriptions, which thus give us a means for semantic comparison. The subsequent Section 5 addresses applications of this approach, while in Section 6 formal results on computational aspects are collected. Section 7 concludes the paper.

2 Transition diagrams and action descriptions

We consider a (*propositional*) *action signature* that consists of a set $\mathbf{F}$ of fluent names, and a set $\mathbf{A}$ of action names. Moreover, let t and f denote the truth values *True* and *False*, respectively. An *action* is a truth-valued function on $\mathbf{A}$, denoted by the set of action names that are mapped to t. A (*propositional*) *transition diagram* of an action signature $\mathcal{L} = \langle \mathbf{F}, \mathbf{A} \rangle$ consists of a set S of *states*, a function $V : \mathbf{F} \times S \to \{f, t\}$, and a subset $R \subseteq S \times 2^{\mathbf{A}} \times S$ of *transitions*. We say that $V(P, s)$ is the *value* of fluent 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 say that A is *executable* in s, if at least one such state s' exists.

A transition diagram can be thought of 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. Every triple $\langle s, A, s' \rangle \in R$ is represented by an edge leading from s to s' and labeled A. An example of a transition diagram is shown in Fig. 1.

We consider a subset of the action description language $\mathcal{C}$[7] that consists of two kinds of expressions (called *causal laws*): *static laws* of the form

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

where L is a fluent literal (an expression of the form P or $\neg P$, where P is a fluent name) and G is a fluent formula, (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 formula (a propositional combination of fluent names and action names). In both cases the part **if** G can be dropped if G is *True*. An *action description* is a set of causal laws. For instance, the action description consisting of the causal laws given in Fig. 2 encodes how a TV system operates; **inertial** $L_1, \ldots, L_k$ stands for **caused** L_i **if** L_i **after** L_i $(1 \leq i \leq k)$.

Fig. 2 An action description encoding a TV domain

$$\begin{aligned}
&\textbf{caused } PowerON \textbf{ after } PushPB_{TV} \wedge \neg PowerON\\
&\textbf{caused } \neg PowerON \textbf{ after } PushPB_{TV} \wedge PowerON\\
&\textbf{caused } TvON \textbf{ if } PowerON\\
&\textbf{caused } \neg TvON \textbf{ if } \neg PowerON\\
&\textbf{inertial } PowerON, \neg PowerON, TvON, \neg TvON.
\end{aligned}$$

The fragment of $\mathcal{C}$ we consider, where L is restricted to a fluent literal in causal laws of the form (5) and (6) rather than allowing fluent formulas, allows to express *definite* causal theories [7]. This means, one can state elementary causal effects, i.e., causation wrt. a particular fluent, but nondeterminism in the attribution of causation is prohibited. For example, in our TV setting we could not express the following nondeterministic causal effect of replacing a broken fuse: **caused** $TvON \vee \neg PowerON$ **after** *ReplaceFuse*. Note that we could rewrite the law into definite laws that are equivalent on our example TV domain. In general, however, such a rewriting would require a semantic reconstruction and thus cannot be done efficiently (i.e., cause an exponential blow-up).

The meaning of an action description can be represented by a transition diagram. Let the *interpretation of a state* be given by the truth assignment to fluent names as induced by the respective fluent values under V. Note that the general definition of a transition diagram as introduced above, allows for different, i.e. distinguishable, states in our domain to yield equal interpretations. However, since such states are indistinguishable by the properties, i.e. fluents, chosen to model the domain, we further on do not distinguish between them. Therefore, let D be an action description with a signature $\mathcal{L} = \langle \mathbf{F}, \mathbf{A} \rangle$. The transition diagram $\langle S, V, R \rangle$[1] *described* by 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 satisfies $G \supset L$; $V(P, s) = s(P)$; and R is the set of all triples $\langle s, A, s' \rangle$ such that s' is the only interpretation of $\mathbf{F}$ which satisfies the heads L of all static laws (5) in D for which s' satisfies G, and dynamic laws (6) in D for which s' satisfies G and $s \cup A$ satisfies U. For example the transition diagram described by the action description of Fig. 2 is shown in Fig. 1. Note that every action description D describes a unique transition diagram, which we denote by $T(D)$. We call an action description D *consistent*, if $T(D)$ has a nonempty state set, and finally remark that checking for consistency is already intractable (see also Section 6). Thus, computing $T(D)$ given D is also not polynomial.

3 Action queries

To talk about observations of the world, or assertions about the effects of the execution of actions, we use an action query language consisting of queries described as follows.

Definition 1 A *basic query* is (a) a *static query* of the form

$$\textbf{holds } F, \tag{7}$$

[1] Note that, by this definition V actually is redundant (but kept for conformity).

where F is a fluent formula; (b) a *dynamic query* of the form

$$\textbf{necessarily } Q \textbf{ after } A_1; \ldots; A_n, \tag{8}$$

where Q is a basic query and each A_i is an action; or (c) a propositional combination of basic queries.

An *existential query* is an expression of the form

$$\textbf{SOMETIMES } Q, \tag{9}$$

where Q is a basic query; a *universal query* is of the form

$$\textbf{ALWAYS } Q, \tag{10}$$

where Q is a basic query.

A *query* is a propositional combination of existential and universal queries.

As for the semantics, let $T = \langle S, V, R\rangle$ be a transition diagram, with a set S of states, a value function V mapping, at each state s, every fluent P to a truth value, and a set R of transitions. A *history* of T of length n is a sequence

$$s_0, A_1, s_1, \ldots, s_{n-1}, A_n, s_n \tag{11}$$

where each $\langle s_i, A_{i+1}, s_{i+i}\rangle$ $(0 \leq i < n)$ is in R.

Definition 2 A state $s \in S$ *satisfies* a basic query q relative to a transition diagram $T = \langle S, V, R\rangle$, denoted $T, s \models q$, if

- q is of the form (7) and the interpretation $P \mapsto V(P, s)$ satisfies F;
- q is of the form (8) and, for every history $s = s_0, A_1, s_1, \ldots, s_{n-1}, A_n, s_n$ of T of length n, basic query Q is satisfied at state s_n;
- q is of the form $\neg Q$ and $T, s \not\models Q$; q is of the form $Q \wedge Q'$, and $T, s \models Q$ and $T, s \models Q'$; or q is of the form $Q \vee Q'$, and $T, s \models Q$ or $T, s \models Q'$.

Note that, for every state s and for every fluent formula F, $T, s \models$ **holds** F iff $T, s \models \neg$**holds** $\neg F$. And since satisfaction of basic queries is defined only wrt. particular states, for every state s, every fluent formula F, and every action sequence $A_1, \ldots, A_n$ (where $n \geq 1$), if it holds that $T, s \models$ **necessarily** (**holds** F) **after** $A_1; \ldots; A_n$ then symmetrically it holds that $T, s \models \neg$**necessarily** ($\neg$**holds** F) **after** $A_1; \ldots; A_n$.

Definition 3 A transition diagram $T = \langle S, V, R\rangle$ *entails* a query q, denoted $T \models q$, if one of the following holds:

- q is an existential query (9) and $\exists s \in S\ (T, s \models Q)$;
- q is a universal query (10) and $\forall s \in S\ (T, s \models Q)$;
- q is of the form $\neg Q$ and $T \not\models Q$; q is of the form $Q \wedge Q'$, and $T \models Q$ and $T \models Q'$; or q is of the form $Q \vee Q'$, and $T \models Q$ or $T \models Q'$.

Note that, $T \models$ **SOMETIMES** Q iff $T \models \neg$**ALWAYS** $\neg Q$, for any basic query Q. We say that T *entails* a set of queries C, denoted $T \models C$, if $T \models q$ for every $q \in C$.

Satisfaction and entailment of queries by action descriptions is now naturally defined as follows.

Definition 4 An action description D satisfies a basic query q at a state s, denoted $D, s \models q$, if $T(D), s \models q$; it entails a query q, denoted $D \models q$, if $T(D) \models q$; and, it entails a set of queries C, denoted $D \models C$, if $T(D) \models C$.

Example 1 Consider the action description in Fig. 2 encoding how a TV system operates; **inertial** $L_1, \ldots, L_k$ stands for **caused** L_i **if** L_i **after** L_i $(1 \leq i \leq k)$. This action description does not entail any set of queries containing

$$\textbf{ALWAYS necessarily (holds } \neg TvON\textbf{) after } \{PushPB_{RC}\}$$

because this query is not satisfied at the state $\{TvON, PowerON\}$; but, it entails the queries:

$$\begin{aligned} &\textbf{ALWAYS holds } PowerON \equiv TvON, \\ &\textbf{ALWAYS holds } PowerON \wedge TvON \supset \\ &\quad \neg\textbf{necessarily (holds } TvON\textbf{) after } \{PushPB_{TV}\}. \end{aligned} \tag{12}$$

In the rest of the paper, an expression of the form

$$\textbf{possibly } Q \textbf{ after } A_1; \ldots; A_n,$$

where Q is a basic query and each A_i is an action, stands for the dynamic query $\neg$**necessarily** $\neg Q$ **after** $A_1; \ldots; A_n$; an expression of the form

$$\textbf{evolves } F_0; A_1; F_1; \ldots; F_{n-1}; A_n; F_n, \tag{13}$$

where each F_i is a fluent formula, and each A_i is an action, stands for **holds** $F_0 \wedge$ **possibly** (**holds** $F_1 \wedge$ **possibly** (**holds** $F_2 \wedge$ ...) **after** A_2) **after** A_1; and an expression of the form

$$\textbf{executable } A_1; \ldots; A_n,$$

stands for the dynamic query **possibly** *True* **after** $A_1; \ldots; A_n$. We sometimes drop **holds** from static queries appearing in dynamic queries.

Intuitively, satisfaction of a query by an action description reduces to satisfaction of respective basic queries at certain states of the transition diagram. Thereby, static queries require a (static) condition to hold in these states, while dynamic queries require particular (sequences of) transitions to emanate from these states. Existential and universal quantification naturally express whether existence of such a state is sufficient for satisfaction or whether the respective basic queries have to hold at all states.

Queries allow us to express various pieces of knowledge about the domain. For instance, we can express the existence of states where a formula F holds by **SOMETIMES holds** F. Similarly, we can express the existence of a transition from some state where a formula F holds to another state where a formula F' holds, by the execution of an action A:

$$\textbf{SOMETIMES holds } F \wedge \textbf{possibly } F' \textbf{ after } A.$$

In general, the existence of a history (11) such that, for each s_i, the interpretation $P \mapsto V(P, s_i)$ satisfies some formula F_i can be expressed by the query:

$$\textbf{SOMETIMES evolves } F_0; A_1; F_1; \ldots; F_{n-1}; A_n; F_n. \tag{14}$$

For instance, query

$$\textbf{SOMETIMES evolves } PowerON;\{PushPB_{TV}\};\ \neg PowerON;\{PushPB_{TV}\};\ PowerON. \tag{15}$$

describes the presence of the following history in Fig. 1:

$$\{PowerON, TvON\},\{PushPB_{TV}\},\{\neg PowerON,\ \neg TvON\},\{PushPB_{TV}\},\{PowerON, TvON\}. \tag{16}$$

Also we can express that there is no transition from any state where a formula F holds:

$$\textbf{ALWAYS holds } F \supset \bigwedge\nolimits_{A\in 2^{\mathbf{A}}} \textbf{necessarily } False \textbf{ after } A.$$

Like in [2], we can express, given an action sequence $A_1, \ldots, A_n$ ($n \geq 1$), executability of it at every state by

$$\textbf{ALWAYS executable } A_1; \ldots; A_n;$$

mandatory effects of it in a given context by

$$\textbf{ALWAYS holds } G \supset \textbf{necessarily } F \textbf{ after } A_1; \ldots; A_n;$$

and possible effects of it in a context by

$$\textbf{ALWAYS holds } G \supset \textbf{possibly } F \textbf{ after } A_1; \ldots; A_n.$$

In the last two queries, F describes the effects and G the context.

4 Weight assignments for action descriptions

There are many possibilities for comparing action descriptions. A generic method is to use a preference relation, i.e., a preorder $\preceq$, which describes for each pair of actions descriptions D and D' whether D is preferable to D', in symbols $D \preceq D'$. The preorder $\preceq$ can be defined in various ways, and may depend on syntactic and/or semantic properties of the action descriptions D and D'.

One particular way is to compare action descriptions on the basis of a numeric *weight* assigned to them, such that preference of an action description D over D' is determined by comparing $weight(D)$ and $weight(D')$, where $weight(D)$ and $weight(D')$ are the weights assigned to D and D', respectively. To this end we present in this section four weight assignments, which assign to each action description D a numeric weight, based on the transition diagram of D and a given set of conditions. Each of the weight assignments aims at valuing some particular aspect of the action description, but without an a priori epistemic meaning. Briefly, the four weight assignments and their aims are as follows:

– *Weighted states*: to value preference over states, e.g., in situations where the dynamics of a system is disregarded and static requirements dominate.

- *Weighted queries*: to value preference over conditions in a generic way, i.e., without resorting to the transition diagram in detail. This applies, e.g., in situations where the conditions serve the purpose of a specification for some behavior of a dynamic system and an action description amounts to a syntactic realization attempt.
- *Weighted histories*: to value preference over trajectories, i.e., sequences of transitions, by taking into account the probability at which they occur. This is needed in situations where dynamic aspects are central.
- *Weighted queries relative to weighted states*: to value a combination of the approaches above, e.g., in situations where static requirements also need to be taken into account when assessing dynamic requirements of a system.

These four assignments are by no means exhaustive, and many others are conceivable. However, they allow to specify preferences over the main semantic constituents of an action description with respect to an action domain – states, transitions, and queries (which allow to express desirable properties and axioms), as well as a combination thereof. Furthermore, the corresponding preference orders are total and, unlike arbitrary preference orders, beneficial with respect to discrimination of choices or component-wise comparability; this will be discussed in more detail in Section 7. After presenting the assignments in detail, we will consider possible usage scenarios for them in Section 4.5.

4.1 Weighted states

Consider two descriptions of an action domain that involves consumption of some resource, such as money. Although some reasoning tasks, such as finding a plan to do shopping, can be performed well with respect to either description, one might prefer finding/executing the plan with respect to the description according to which the number of possible world states where resource consumption is less than some given value. Such a preference over states of a transition diagram $T = \langle S, V, R\rangle$ can be modelled by assigning a weight to each state in S, by means of a function g. Such a function assigning real numbers to states of the world can be considered as a *utility function*, as in decision theory. If one state of the world is preferred to another state of the world then it has higher utility for the agent; here "utility" is understood as "the quality of being useful" as in [8]. Alternatively, the function g can be viewed as a *reward function*: being at a state s will give a reward of $g(s)$ to the agent. For instance, in the example above, $g(s)$ could be the amount of money the agent has at state s. In this case, $g(s)$ can be understood as a utility/reward function.

Given a utility function for a set S of states, we can identify the highly preferred states relative to a given number l: a state with the weight greater than l is highly preferred.

Definition 5 The weight of an action description D relative to g and l is defined as:

$$weight_s(D) = |\{s : s \in S,\ T(D) = \langle S, V, R\rangle,\ g(s) > l\}|.$$

With respect to this definition, the more the number of states that are highly preferred by the agent, the more preferred the action description is.

Example 2 Consider the transition diagram in Fig. 1 described by D. Take, for each $s \in S$,

$$g(s) = \begin{cases} 2 \text{ if } PowerON \in s \\ 1 \text{ otherwise.} \end{cases} \tag{17}$$

Take $l = 1$. Then $weight_s(D) = 1$.

4.2 Weighted queries

Using weighted states we can express static preferences over action descriptions. Most often, however, action descriptions serve to model dynamic systems, where certain specifications have to be met, that do not just depend on the different system states. One possibility to account for the dynamic behaviour in comparing action descriptions is by checking whether they satisfy some given conditions (e.g., some "soft" constraints), possibly of different importance. For instance, one condition might be concurrence of at most three actions, and another condition might be the executability of some particular action; and the first condition might be more important for one sort of planning tasks because the execution of more than three actions is often problematic. Such a preference over conditions can be specified by assigning weights to the queries describing these conditions. Based on such weighted queries, we can define the weight of an action description D as follows.

Definition 6 Let C be a set of queries, along with a weight function f mapping each condition in C to a real number. The weight of D (relative to C and f) is

$$weight_q(D) = \sum\nolimits_{c \in C,\, D \models c} f(c).$$

Intuitively, the weight of an action description defined relative to the weights of queries shows how much the set C of given preferable queries are satisfied. With this definition, the more the highly preferred queries are satisfied, the more preferred the action description is.

Example 3 Suppose that C consists of (15) and

$$\textbf{ALWAYS executable } \{PushPB_{RC}\}, \tag{18}$$

with weights 1 and 2 respectively. For the description D with the transition diagram in Fig. 1, $weight_q(D) = 3$.

4.3 Weighted histories

A major field of application for action descriptions is in representing planning problems. In this context the focus is on how a state evolves by the execution of a sequence of actions. This motivates the comparison of different action descriptions by assigning weights to such "histories," rather than just states/conditions, and to accumulate them. Informally, this can be achieved as follows.

For a query c of form (14), we value any history w which satisfies c, as a "desired" witness of this query, in terms of the utility $u(w)$ of w, which is recursively

determined by a state reward function $g(s)$ and probability distribution $m(\langle s, A, s\rangle)$ on the transitions. We then accumulate all these values. Intuitively, the more and the higher desired the histories which satisfy the query are, the more will the action description be preferred. For handling several queries and discriminating between them, weighted sums of history utilities using query weights $f(c)$ are calculated and accumulated. So intuitively, a high weight (and thus preference) is assigned to action descriptions in which 'important" queries are (often) satisfied by desired histories.

More formally, the weight assignment is defined as follows. In a transition diagram $T = \langle S, V, R\rangle$, we say that a history (11) of length n is *desired* with respect to a given query (14), if, for each i, the interpretation $P \mapsto V(P, s_i)$ satisfies F_i.

Let D be an action description, and $T(D) = \langle S, V, R\rangle$. Let C be a set of queries, along with a weight function f mapping each condition in C to a number. Let H_C be the set of pairs (w, c) such that w is a desired history in $T(D)$ with respect to the query c of form (14) in C. Let us denote by $st(w)$ the starting state s_0 of a history w of form (11). We define a function h mapping each desired history w appearing in H_C to a real number, in terms of the utility $u(w)$ of state $st(w)$ with respect to w:

$$h(w) = u(w) \times \sum\nolimits_{(w,c)\in H_C} f(c).$$

The function u mapping a history w of form (11) to a real number can be defined in terms of a sequence of functions u_i. Given a utility function (or a reward function) g mapping each state in S to a real number, and a *transition model* m mapping each transition $\langle s, A, s'\rangle$ in R to a probability (i.e., the probability of reaching s' from s after execution of A):

$$\begin{aligned} u_n(w) &= g(s_n) \\ u_i(w) &= g(s_i) + m(\langle s_i, A_{i+1}, s_{i+1}\rangle) \times u_{i+1}(w)\ (0 \le i < n) \\ u(w) &= u_0(w). \end{aligned}$$

These equations are essentially obtained from the equations used for value determination in the policy-iteration algorithm described in [8, Chapter 17]: take $\{s_0, \ldots, s_n\}$ as the set of states, $\langle s_i, A_{i+1}, s_{i+1}\rangle$ as the possible transitions, the mapping $s_i \mapsto A_{i+1}$ as the fixed policy, U as u, U_i as u_i, R as g, and M as m.

Definition 7 The weight of D in terms of the weights of desired histories $w_1, \ldots, w_z$ appearing in H_C is defined as

$$weight_h(D) = \sum\nolimits_{i=1}^{z} h(w_i).$$

The more the utilities of desired histories (or trajectories) satisfied by the action description, the more preferred the action description is.

Example 4 Suppose that C consists of query (15), with weight 3. Consider the transition diagram $T = \langle S, V, R\rangle$ in Fig. 1. Let us denote history (16) by w, and

query (15) by c. Then H_C contains (w, c). Take $g(s)$ as in (17). Take $l = 1$. Suppose that, for each transition $\langle s, A, s' \rangle$ in R,

$$m(\langle s, A, s' \rangle) = \begin{cases} 0.5 \text{ if } s = \{PowerON, TvON\} \ \wedge \ |A| = 1 \\ 1 \quad \text{otherwise.} \end{cases} \tag{19}$$

Then $u(w) = 3.5$ and $h(w) = u(w) \times \sum_{(w,c) \in H_C} f(c) = 3.5 \times 3 = 10.5$. Hence $weight_h(D) = 10.5$.

4.4 Weighted queries relative to weighted states

The three approaches above can be united by also considering to what extent each universal query in C is entailed by the action description. The idea is to take, while computing the weight of a description relative to weighted queries, also the states into account at which these queries are satisfied.

To this end, the reward $g(s)$ of a state s is redefined if it is the starting point of a history which satisfies some existential query of form (14), to its accumulated utility in this respect. The given weight $f(q)$ of such a query q is then scaled with the average reward of the states in the transition diagram which witness the query. Thus intuitively, few strong (highly desired) witness states are valued higher than many weak (less desired) ones. We may then similarly measure to what extent a universal query of form **ALWAYS** Q dual to form (14) is satisfied, and can extended the measure to disjunctions of universal and existential queries.

More formally, the weight assignment is as follows. Let D be an action description. Let $T(D) = \langle S, V, R \rangle$, along with a weight function g mapping each state in $T(D)$ to a real number. Let C be a set of queries such that every query q in C is an existential query, a universal query, or a disjunction of both.

First, for each state s in S, we compute its new weight $g'(s)$, taking into account utilities of the desired histories starting with s. Let H_C be the set of pairs (w, c) such that w is a desired history in $T(D)$ with respect to the query c of form (14) in C. Let W be the set of histories that appear in H_C. Let u be a function mapping a history w to a real number, describing the utility of state s with respect to w. Then the new weight function g' is defined as follows:

$$g'(s) = \begin{cases} g(s) & \text{if } \nexists w (w \in W \wedge st(w) = s) \\ \sum_{w \in W, st(w) = s} u(w) & \text{otherwise.} \end{cases}$$

Next, for each query c in C, we compute its new weight $f'(c)$. Let f be a function mapping each condition in C to a real number. We will denote by $S_D(B)$ the set of states s such that $D, s \models B$. Then we define f' as follows:

$$f'(q) = \begin{cases} f'(q') + f'(q'') & \text{if } q = q' \vee q'' \\ \beta & \text{if } q = \textbf{ALWAYS } B \\ \gamma & \text{if } q = \textbf{SOMETIMES } B \wedge |S_D(B)| > 0 \\ 0 & \text{if } q = \textbf{SOMETIMES } B \wedge |S_D(B)| = 0, \end{cases}$$

where $\beta = f(q) \times \sum_{s \in S_D(B)} g'(s)$; $\gamma = f(q) \times [(\sum_{s \in S_D(B)} g'(s))/|S_D(B)|]$. Intuitively, f' describes to what extent each preferable query q is satisfied.

Definition 8 The weight of D (relative to C and f') is defined by the sum

$$weight_{qs}(D) = \sum\nolimits_{q \in C} f'(q).$$

Intuitively, it describes how much and to what extent the given preferable queries are satisfied.

Example 5 Suppose that C consists of three queries:

$$\textbf{ALWAYS executable } \{PushPB_{TV}\}, \tag{20}$$

$$\textbf{SOMETIMES } \neg\textbf{executable } \{PushPB_{RC}, PushPB_{TV}\}, \tag{21}$$

and query (15), denoted by c_1, c_2 and c_3 respectively. Consider an action description D, with the transition diagram in Fig. 1. Let us denote history (16) by w; then $H_C = \{(w, c_3)\}$. Take the utility function g as (17), and the transition model m as (19). Take $f(c_1) = 1$, $f(c_2) = 2$, $f(c_3) = 3$. Then, we obtain g' ($\{PowerON, TvON\}$)$=3.5$, $g'(\{\neg PowerON, \neg TvON\})=1$, as well as $f'(c_1)=4$, $f'(c_2)=4$, $f'(c_2)=$ 10.5. Therefore, $weight_{qs}(D) = 18.5$.

4.5 Usage

As pinpointed at the beginning of this section, there are other possibilities of defining a weight function for action descriptions; and as hinted above by examples, deciding for a weight function depends on particular reasoning task and/or what the user is concerned about the most. Let us briefly compare distinguishing qualities of the weights introduced above and deflect some usage scenarios upon this.

One dimension along which the weights can be differentiated is to what extent they respond to details of the model, that is to the transition diagram. Weighted states, for example, just aggregate over preferred states, neglecting any transitions. Thus, they provide an abstract 'static view' on the model. If one is more interested in prediction and has concerns about the values of some fluents, then one would probably assign weights to action descriptions with respect to the weights of states. This may serve to identify points of failure or sources of error, however, without revealing information on the structure of the error.

By means of weighted queries, not only static but also dynamic requirements that are expected to hold on the transition diagram of an action description can be expressed in a temporal logic like language. Nevertheless, it is also an extensional view on the model that does not account for the particular structure, i.e. the number of violations. It therefore provides a 'specification view' on the model, similar as in model checking: One is interested in a model that satisfies the specification but not in how it does. So, if the major concern is that an action description satisfies a formal system specification, then one may resort to weighted queries to enforce it.

The intention of weighted histories is to allow for a qualitative assessment of certain actions, respectively sequences of actions (trajectories, runs). They thus allow for an 'agent view' on the model building on utilities of states and probabilities for transitions. For instance, if one is interested in planning, and has some preferences over executions of sequences of actions, then one would assign weights to a description with respect to these histories.

Eventually, the idea of weighted queries relative to weighted states is to extend the qualitative assessment to queries in general, that is, taking state utilities into account when weighing queries. This weight thus allows to combine the above approaches, yielding a more generic approach: one can express different, and more sophisticated views on the model – sometimes perhaps at the cost of clarity and ease of use.

Further examples of the weight measures are given in Appendices.

5 Applications

We now turn to applications of the approach introduced in the previous section, that is, using semantic weights for comparing action descriptions. We illustrate how it can be applied for updating action descriptions, as well as for identifying elaboration tolerant action descriptions, and discuss benefits that can be achieved in comparison to syntactic approaches.

5.1 Updating an action description

Suppose that an action description D consists of two parts: D_u (unmodifiable causal laws) and D_m (modifiable causal laws); and a set C of conditions is partitioned into two: C_o (obligatory) and C_p (preferable). We define an *Action Description Update* (*ADU*) problem by an action description $D = (D_u, D_m)$, a set Q of causal laws, a set $C = (C_o, C_p)$ of queries, all with the same signature, and a weight function *weight* mapping an action description to a number. The weight function can be defined relative to a set of queries, a utility function, or a transition model, as seen in the previous section. We say that a consistent action description D' is a *solution* to the ADU problem $(D, Q, C, weight)$ if

(1) $Q \cup D_u \subseteq D' \subseteq D \cup Q$,
(2) $D' \models C_o$,
(3) there is no other consistent action description D'' such that $Q \cup D_u \subseteq D'' \subseteq D \cup Q$, $D'' \models C_o$, and $weight(D'') > weight(D')$.

The definition of an ADU problem in [2] is different from the one above mainly in two ways. First, $C_p = \emptyset$. Second, instead of (3) above, the following syntactic condition is considered: there is no consistent action description D'' such that $D' \subset D'' \subseteq D \cup Q$, and $D'' \models C$.

The semantic approach above has mainly two benefits, compared to the syntactic approach of [2]. First, there may be more than one solution to some ADU problems with the syntactic approach. In such cases, a semantic approach can be applied to further prune the set of admissible solutions and may, eventually, allow to pick one of these solutions. Example 6 illustrates this benefit. Second, for an ADU problem, if no consistent action description D' satisfying (1) satisfies the obligatory queries (C_o), there is no solution to this problem with either syntactic or semantic approach. In such a case, we can use the semantic approach with weighted queries, to relax some obligatory queries in C_o (e.g., move them to C_p). The idea is first to solve the ADU problem $((D_u, D_m), Q, (\emptyset, C_o'), weight)$, where C_o' is obtained from C_o by

complementing each query, and where the weights of queries in $C_o{}'$ are equal to some very small negative integer; and then to identify the queries of $C_o{}'$ satisfied in a solution and add them to C_p, with weights multiplied by -1. This process of relaxing some conditions of C_o to find a solution is illustrated in Example 7.

Example 6 Consider, for instance, the action description D given in Fig. 2 as $D = (D_m, D_u)$, where $D_m = \{(1), (2)\}$ (and thus $D_u = D \setminus D_m$), that describes a TV system with a remote control. Suppose that, later the following information, Q, is obtained:

caused $TvON$ **after** $PushPB_{RC} \wedge PowerON \wedge \neg TvON$
caused $\neg TvON$ **after** $PushPB_{RC} \wedge TvON$.

Suppose that we are given the set $C = (C_o, C_p)$ of queries where C_o consists of the queries (3) and

$$\textbf{SOMETIMES evolves } \neg TvON; \{PushPB_{TV}\}; \neg TvON, \tag{22}$$

and C_p consists of the queries (4), (15), (18), (20), (21), denoted by $c_1, \ldots, c_5$ respectively. When Q is added to D, the meaning of $D \cup Q$ can be represented by a transition diagram almost the same as that of D (Fig. 1), except that there is no outgoing edge from the state $\{PowerON, TvON\}$ with the label $\{PushPB_{RC}\}$; thus only (3), (15) and (22) in C are entailed by $D \cup Q$. The question is how to update D by Q such that the obligatory conditions, C_o, are satisfied, and the preferable conditions, C_p, are satisfied as much as possible.

The consistent action descriptions for which (1) holds are

$$\begin{aligned} D^{(1)} &= D \cup Q, \\ D^{(2)} &= D_u \cup Q \cup \{(2)\}, \\ D^{(3)} &= D_u \cup Q \cup \{(1)\}1, \\ D^{(4)} &= D_u \cup Q. \end{aligned}$$

With the syntactic approach of [2], we have to choose between $D^{(2)}$ and $D^{(3)}$, since they have more causal laws. Consider the semantic approach based on weighted histories (i.e., $weight = weight_h$), with (17) as the utility function g, (19) as the transition model m, and

$$f(c_1) = 3,\ f(c_2) = 1,\ f(c_3) = 4,\ f(c_4) = 3,\ f(c_5) = 2.$$

Let us consider the states

$$\begin{aligned} s_0 &= \{PowerON, TvON\}, \\ s_1 &= \{PowerON, \neg TvON\}, \\ s_2 &= \{\neg PowerON, \neg TvON\}; \end{aligned}$$

and the histories

$$\begin{aligned} w_0 &= s_0, \{PushPB_{RC}\}, s_1, \\ w_1 &= s_1, \{PushPB_{RC}\}, s_0, \\ w_2 &= s_0, \{PushPB_{TV}\}, s_2, \{PushPB_{TV}\}, s_1, \\ w_3 &= s_1, \{PushPB_{TV}\}, s_2, \{PushPB_{TV}\}, s_1 \end{aligned}$$

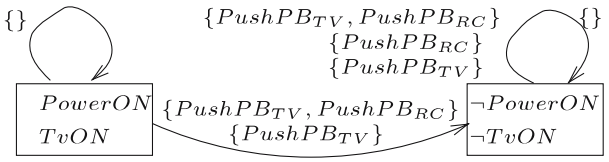

Fig. 3 Transition diagram of $D^{(2)} = D_u \cup Q \cup \{(2)\}$

whose utilities, $u(w_i) = u_0(w_i)$, can be computed as follows:

w	i	$u_i(w)$
w_0	1	$g(s_1) = 2$
w_0	0	$g(s_0) + m(\langle s_0, \{PushPB_{RC}\}, s_1\rangle) \times u_1(w_0) = 3$
w_1	1	$g(s_0) = 2$
w_1	0	$g(s_1) + m(\langle s_1, \{PushPB_{RC}\}, s_0\rangle) \times u_1(w_1) = 4$
w_2	2	$g(s_1) = 2$
w_2	1	$g(s_2) + m(\langle s_2, \{PushPB_{TV}\}, s_1\rangle) \times u_2(w_2) = 3$
w_2	0	$g(s_0) + m(\langle s_0, \{PushPB_{TV}\}, s_2\rangle) \times u_1(w_2) = 3.5$
w_3	2	$g(s_1) = 2$
w_3	1	$g(s_2) + m(\langle s_2, \{PushPB_{TV}\}, s_1\rangle) \times u_2(w_3) = 3$
w_3	0	$g(s_1) + m(\langle s_1, \{PushPB_{TV}\}, s_2\rangle) \times u_1(w_3) = 5$

That is,

$$u(w_0) = 3, u(w_1) = 4, u(w_2) = 3.5, u(w_3) = 5.$$

For $D^{(2)}$ (see Fig. 3), since $H_{C_p} = \emptyset$, $weight_h(D^{(2)}) = 0$.
For $D^{(3)}$ (see Fig. 4), since H_{C_p} contains (w_0, c_5), (w_1, c_5), (w_2, c_3), and (w_3, c_3),

$$\begin{aligned} weight_h(D^{(3)}) &= u(w_0) \times f(c_5) + u(w_1) \times f(c_5) \\ &\quad + u(w_2) \times f(c_3) + u(w_3) \times f(c_3) \\ &= 3 \times 2 + 4 \times 2 + 3.5 \times 4 + 5 \times 4 = 48. \end{aligned}$$

Thus $D^{(3)}$ is the solution.

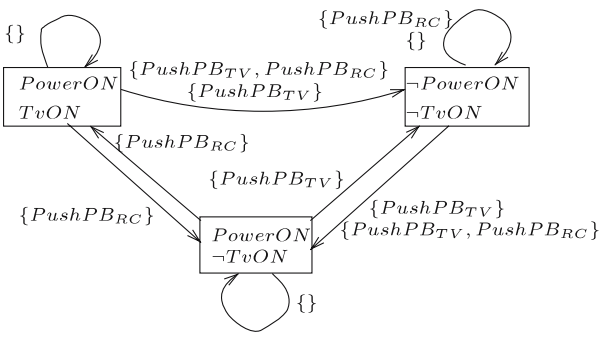

Fig. 4 Transition diagram of $D^{(3)} = D_u \cup Q \cup \{(1)\}$

Example 7 Take D, Q, C_p, and $D^{(1)}$–$D^{(4)}$ as in Example 6, and C_o as the set consisting of the queries

$$\textbf{SOMETIMES} \neg \bigvee_{A \in 2^{\mathbf{A}}} \textbf{executable}\ A, \tag{23}$$

$$\textbf{ALWAYS}\ \neg\textbf{evolves}\ \neg TvON; \{PushPB_{TV}\}; \neg TvON, \tag{24}$$

denoted by c_1' and c_2' respectively. None of the descriptions $D^{(1)}$ – $D^{(4)}$ entails C_o. Therefore, there is no solution to the ADU problem above with either the syntactic approach of [2] or any of the semantic approaches above. To identify which queries in C_o we shall move to C_p, first we obtain C_0' from C_o by negating each query in C_0, and assigning a very small negative integer, say -100, as their weights. So C_0' consists of the queries (3) and (22), denoted by c_1'' and c_2'', with weights -100. With the semantic approach based on weighted queries (i.e., $weight = weight_q$),

$$\begin{aligned} weight_q(D^{(1)}) &= f\left(c_1''\right) = -100, \\ weight_q(D^{(2)}) &= weight_q(D^{(3)}) = f\left(c_1''\right) + f\left(c_2''\right) = -200, \\ weight_q(D^{(4)}) &= f\left(c_1''\right) + f\left(c_2''\right) = -200, \end{aligned}$$

the description $D^{(1)}$ then is the preferred solution to the ADU problem $((D_u, D_m), Q, (\emptyset, C_o'), weight_q)$. This suggests relaxing the obligatory query (23) (i.e., adding the query (23) to C_p with the weight 100) and solving the new ADU problem, $((D_u, D_m), Q, \{(24)\}, C_p \cup \{(23)\}, weight_q)$, for which the description $D_u \cup Q$ is the solution.

5.1.1 Other semantic approaches to action description updates

Given a consistent action description E, condition (3) of an ADU problem $(D, Q, C, weight)$ can be replaced by

> (3)′ there is no other consistent action description D'' such that $Q \cup D_u \subseteq D'' \subseteq D \cup Q$, $D'' \models C_o$, and $|weight(D'') - weight(E)| < |weight(D') - weight(E)|$

to express that, among the consistent action descriptions D' for which (i) and (ii) hold, an action description that is "closest" to (or most "similar" to) E is picked. Here, for instance, E may be $D \cup Q$, to incorporate as much of the new information as possible, although $D \cup Q$ may not entail C. What is meant by closeness or similarity is based on the particular definition of the weight function. For instance, based on the weights of states only, with $g(s) = 1$ if s is a state of E, and 0 otherwise, the closeness of an action description to E is defined in terms of the common world states.

5.2 Elaboration tolerance

Suppose that we are given a set $\mathcal{D}$ of action descriptions; and a set C of conditions, each describing a possible elaboration. We say about two action descriptions D and D' in $\mathcal{D}$ that D is *more elaboration tolerant than* D' with respect to C, if $weight(D) >$

weight(D'), where *weight* is defined relative to C among other things (e.g., *weight* can be $weight_q$, $weight_h$, or $weight_{qs}$). The question we are interested in is *which action description in $\mathcal{D}$ is the most elaboration tolerant with respect to C.*

Weight functions defined relative to possible elaborations, which might also take into account the significance of those elaborations, are reasonable measures of difficulty of modifying the action descriptions to entail possible elaborations: if an action description entails higher number of elaborations, then it is more tolerant to elaborations; in other words, if an action description does not entail a lower number of elaborations, then it is easier to modify it by adding or modifying its causal laws (e.g., like in [9]).

The following is a variation of the example in [6].

Example 8 Consider the following two action descriptions D and D', that contain the laws **caused** F **if** F and **caused** $\neg F$ **if** $\neg F$, for every fluent $F \in \{Cold, Cloudy, Sunny, Wet, Winter, Tropical\}$, together with the laws below describing how the weather changes when it rains.

D :	D' :
caused *Cold* **after** *Rain*	**caused** *Wet* **after** *Rain*
caused *Wet* **after** *Rain*	**caused** $\bot$ **after** $Rain \wedge \neg Cloudy$
caused $\bot$ **after** $Rain \wedge \neg Cloudy$	**caused** $\neg Sunny$ **after** *Sunset*
caused $\neg Sunny$ **after** *Sunset*	**caused** $\neg Sunny$ **if** *Cloudy*
caused $\neg Sunny$ **if** *Cloudy*	**caused** *Cold* **if** $\neg Sunny \vee (Wet \wedge Cold)$

Consider also the following possible elaborations on how the weather changes relative to seasons and regions:

e_1 : **ALWAYS holds** *Winter* $\supset$ **necessarily** *Cold* **after** *Sunset*
e_2 : **SOMETIMES evolves** *Tropical*; {*Rain*}; $\neg Cold$
e_3 : **ALWAYS holds** $\neg Winter \supset$ **possibly** *Cold* **after** *Rain*.

Suppose that $weight = weight_q$, and $f(e_1) = f(e_3) = 3$, $f(e_2) = 2$ (because, e.g., if someone does not travel much, then elaborations relative to seasons are more important for her). Then D' is more elaboration tolerant than D because $weight_q(D) = 3$ whereas $weight_q(D') = 8$.

Amir compares in [6] two axiomatic theories Σ and Σ' with respect to a target axiomatic theory Σ_{target}, in terms of a syntactic transformation (e.g., the number of additions and deletions of sentences). This idea might be captured with respect to a semantic measure, by means of comparing how similar/diverse the action descriptions are as discussed at the end of Section 5.1. On the other hand, we usually do not know the target theory (resp., the most elaboration tolerant action description), but may have an idea of possible elaborations based on our observations in different circumstances (as in the example above). In such cases, it is reasonable to decide which action description in $\mathcal{D}$ is the most elaboration tolerant one, by comparing action descriptions semantically with respect to some weight function that takes into account C.

6 Computational aspects

We confine here to discuss the complexity, in order to shed light on the cost of computing the weight measures. To this end, we define:

Definition 9 A *weighted action domain*, $\mathcal{D}_w$, is a quintuple $\langle D, C, g, f, m\rangle$, of

- An action description D,
- A set C of queries,
- A function g mapping every state to a real number,
- A function f mapping every query in C to a real number, and
- A function m mapping every transition to a probability.

We furthermore assume that the basic functions, that is, $g(s)$, $f(q)$, and $m(\langle s, A, s'\rangle)$ are computable in polynomial time, and define

- $weight_s(\mathcal{D}_w) = weight_s(D)$ relative to g,
- $weight_q(\mathcal{D}_w) = weight_q(D)$ relative to C and f,
- $weight_h(\mathcal{D}_w) = weight_h(D)$ relative to C, f, g and m,
- $weight_{qs}(\mathcal{D}_w) = weight_{qs}(D)$ relative to C, f, g and m.

For a background on complexity, we refer to the literature (see e.g. [10]).[2]

Apparently, none of the different weights above is polynomially computable from an input action description D and a set C of queries in general. Indeed, deciding whether S has any states is easily seen to be NP-complete, thus intractable: Given a propositional formula $f = \bigwedge_{i=1}^{n} c_i$ in conjunctive normal form (CNF), let $G_i = \neg c_i$, $1 \leq i \leq n$, denote the conjunction that is obtained by negating the clause c_i, and consider the action description consisting of the static laws **caused** G_i **if** *False*, for $1 \leq i \leq n$. It is easily verified that for this action description S is nonempty if and only if f is satisfiable.

Furthermore, evaluating arbitrary queries q on D (deciding $D \models q$) is a PSPACE-complete problem. Indeed, q can be evaluated on D in polynomial space. On the other hand, evaluating quantified boolean formulas (QBFs), which is PSPACE-complete, can be reduced to deciding $D \models q$. (See, e.g., p.142 in [10] for the class PSPACE, p.455 for the definition QBFs and the corresponding satisfiability problem – termed QSAT there – and Theorem 19.1 on p.456 for its PSPACE-completeness.)

Proposition 1 *Given an action description D and a query q, the problem of deciding whether $D \models q$ is PSPACE-complete.*

Proof Both checking whether a static query (7) holds at a state s, and whether a sequence (11) is a history of $T(D)$ is feasible in polynomial time (indeed, note that given s, A, s' and D, deciding whether $\langle s, A, s'\rangle \in R$, where $T(D) = \langle S, V, R\rangle$, is polynomial). Hence, a recursive procedure for deciding $D, s \models Q$ given D, s, and a basic query Q in polynomial space (by exploring all possible histories), is straightforward. Consequently, deciding $D \models q$ where q is an existential or a universal query, or

[2] See also http://qwiki.caltech.edu/wiki/Complexity_Zoo.

any propositional combination of such queries, is also feasible in polynomial space. This shows membership in PSPACE.

To show PSPACE-hardness, we reduce QSAT, i.e., deciding whether a given QBF is true, to this problem. Let $f = Q_1 X_1 \dots Q_n X_n E$ be a QBF on propositional atoms $X = \{x_1, \dots, x_n\}$, where $X_1, \dots, X_n$ is a partitioning of X, $Q_i \in \{\exists, \forall\}$, and E is a propositional formula on X. Take X as the set of fluents and $A_1, \dots, A_n$, as action symbols in an action description D consisting of statements

$$\textbf{caused} \quad x \textbf{ if } \quad x \textbf{ after } A_i,$$

$$\textbf{caused } \neg x \textbf{ if } \neg x \textbf{ after } A_i,$$

$$\textbf{caused} \quad x_j \textbf{ after } A_i \wedge x_j,$$

$$\textbf{caused } \neg x_j \textbf{ after } A_i \wedge \neg x_j,$$

for $1 \leq i \leq n$, $x \in X_i$, and $x_j \notin X_i$. Let furthermore

$$q = \textbf{SOMETIMES } N_1 \, (\dots (N_n \; E \textbf{ after } A_n) \dots) \textbf{ after } A_1, \tag{25}$$

where $N_i =$ **necessarily** if $Q_i = \forall$ and $N_i =$ **possibly** otherwise, for $1 \leq i \leq n$. Then, the QBF f is true iff $D \models q$. This is seen easily, noting that the transition diagram $T(D)$ contains all (consistent) truth assignments to the propositional variables as states, and a transition $\langle s, A_i, s' \rangle$ iff s and s' coincide on all fluents not in X_i, i.e., on all propositional variables not bound by quantifier Q_i. □

6.1 Computation given D and C

As it turns out, all four weights are computable in polynomial space. This is because each weight is a sum of (in some cases exponentially many) terms, each of which can be easily computed in polynomial space, using exhaustive enumeration. In some cases, the computation is also PSPACE-hard, but in others supposedly easier. Completeness results are wrt. logarithmic space reductions (see, e.g., Def. 8.1, p.160 in [10]).

Theorem 1 *Let $\mathcal{D}_w = \langle D, C, g, f, m \rangle$ be a weighted action domain. Then the following hold:*

(1) *Computing $weight_s(\mathcal{D}_w)$ is #P-complete;*
(2) *Computing $weight_q(\mathcal{D}_w)$ is FPSPACE-complete;*
(3) *Computing $weight_h(\mathcal{D}_w)$ is (modulo some normalization) #P-complete, if the range of f and g are nonnegative numbers, and GapP-complete for arbitrary f and g;*
(4) *Computing $weight_{qs}(\mathcal{D}_w)$ is FPSPACE-complete.*

These results are also shown in the first row of Table 1. Here #P (cf. Def. 18.1, p.441 in [10]) is the class of problems where the output is an integer that can be obtained as the number of runs of an NP Turing machine which accepts the input; problems polynomially solvable with an #P oracle are believed not to be PSPACE-hard. GapP [11, 12] is the closure of #P under subtraction (equivalently, it contains the functions which are expressible as number of accepting computation minus the number of rejecting computations of an NP Turing machine).

Table 1 Complexity of computing weights (completeness)

Input / Weight	$weight_s$	$weight_q$	$weight_h$	$weight_{qs}$
D, C	#P	FPSPACE	GapP *	FPSPACE
D, C, S	polynomial	polynomial	polynomial	polynomial
D_{pol}**, C	in $FP_{\|}^{NP}$	in $FP_{\|}^{NP}$	in $FP_{\|}^{NP}$	in $FP_{\|}^{NP}$

* #P for non-negative $g(s), f(q)$;

** $|S|$ is polynomially bounded

Proof Let $\mathcal{D}_w = \langle D, C, g, f, m \rangle$ be a weighted action domain.

(1) Computing $weight_s(\mathcal{D}_w)$ amounts to counting the number of states s such that $g(s) > l$. This problem is thus easily seen to be in #P. Moreover, it is also #P-complete, since the canonical #P-complete problem #SAT of counting the models of a propositional formula is readily reduced to it: consider an action description that entails for each clause c the static causal law **caused** $\bot$ **if** $\neg c$, let $g(s) = 1$, for all states s, and $l = 0$.

(2) As for $weight_q(\mathcal{D}_w)$, we must evaluate each query $q \in C$ on D and then take a sum. As testing $D \models q$ is PSPACE-complete, computing $weight_q(D)$ is in FPSPACE, i.e., the class of functions computable in polynomial space.[3] Moreover, the problem can also be shown to be hard for this class. Consider any function $f \in$ FPSPACE . For a given input x, to decide whether the i-th bit of $f(x)$ is 1, is a PSPACE problem and can be reduced to deciding $D \models q$ as in Proposition 1. Since $f(x)$ is polynomial, a polynomial number of queries of form (25) is needed to decide each bit of $f(x)$. Moreover, w.l.o.g., the same action description D can be used for each of these queries. Let $f(q) = 2^i$, for q deciding the i-th bit of $f(x)$, then $weight_q(D) = f(x)$.

(3) Computing $weight_h(\mathcal{D}_w)$ modulo some normalization (which casts the problem to one with integer values), can like computing $weight_s(\mathcal{D}_w)$ be seen to be in #P, if the functions $g(s)$ and $f(q)$ are non-negative. Indeed, each relevant history w can be nondeterministically generated in polynomial time, and $u(w)$ and $h(w)$ are easily computed from w; to account for $h(w)$, simply that many accepting computation branches are nondeterministically generated. On the other hand, #SAT is reducible to computing $weight_h(\mathcal{D}_w)$.
We give a simple reduction, which is as follows. Let E be a SAT instance on propositional atoms $x_1, \ldots, x_n$, which without loss of generality is not satisfied if all atoms are assigned false. We let $x_1, \ldots, x_n$ be the fluents and a the single action symbol in an action description D, which consists of all statements

$$\textbf{caused} \quad x_i \ \textbf{if} \quad x_i \ \textbf{after} \ \top,$$

$$\textbf{caused} \ \neg x_i \ \textbf{if} \ \neg x_i \ \textbf{after} \ \top,$$

[3] Note that the output has to be generated on the working tape (and thus is of polynomial size), i.e., we consider machines that do not have an extra output tape. Sometimes this class is also denoted FPSPACE(poly) = #PSPACE = $\#P^{PSPACE}$.

where $\top$ stands for a tautology, and let C consist of the single query

$$c = \textbf{evolves} \bigwedge_{i=1}^{n} \neg x_i; \emptyset; \top.$$

Informally, the transition diagram of D for the empty action $\emptyset$ is the complete graph whose nodes are given by all truth assignments to $x_1, \ldots, x_n$. Note also, that c captures the transitions from the assignment in which all atoms are false to some arbitrary assignment via the empty action $\emptyset$. Now we define that $g(s) = 2^n$ if s satisfies E, and $g(s) = 0$ if s does not satisfy E, for each s. Furthermore, we define that transitions have uniform probability, i.e., $m(\langle s, A, s' \rangle) = 1/2^n$ for each transition $\langle s, A, s' \rangle$ in $T(D)$. Let $f(c) = 1$.

It is easy to see that H_C contains all pairs (w, c) where $w = s_0, \emptyset, s_1$ such that s_0 is the state in which all x_i are false and s_1 is an arbitrary state: A history w is desired wrt. c, iff it is of length $n = 2$ and the only action involved is the empty action $\emptyset$, that is $w = s_0, \emptyset, s_1$, such that the interpretation of s_0 satisfies $\bigwedge_{i=1}^{n} \neg x_i$, and the interpretation of s_1 satisfies $\top$. Furthermore, $h(w) = 1$ if s_1 satisfies E and $h(w) = 0$ otherwise. Therefore, $weight_h(D)$ is the number of satisfying assignments of E. Since D, C, m, f, and g are obviously constructible in polynomial time, and since moreover m, f, and g are computable in polynomial time, we obtain #P-hardness of computing $weight_h(\mathcal{D}_w)$.

In case of arbitrary (possibly negative) $g(s)$ and $f(q)$, $weight_h(\mathcal{D}_w)$ is computable as the difference of two #P functions. Therefore, computing $weight_h(\mathcal{D}_w)$ is in the class GapP. Indeed, we have that

$$weight_h(D) = \sum_{(w,c) \in H_C^+} u(w) \times f(c) \; - \sum_{(w,c) in H_C^-} -u(w) \times f(c),$$

where H_C^+ contains all pairs (w, c) from H_C such that $u(w) \times f(c)$ is non-negative and H_c^- contains all pairs (w, c) from H_C such that $u(w) \times f(c)$ is negative. Both $\sum_{(w,c) \in H_C^+} u(w) \times f(c)$ and $\sum_{(w,c) \in H_C^-} -u(w) \times f(c)$ can be computed in #P. On the other hand, computing the difference $f_1 - f_2$ of two #P functions f_1 and f_2 can be polynomially reduced to computing $weight_h(\mathcal{D}_w)$ for some weighted action domain $\mathcal{D}_w$ in polynomial time. More precisely, with a slight adaption of the above construction, we can reduce computing the difference of the number of satisfying assignments $\#(E_1)$ and $\#(E_2)$ of two SAT instances E_1 and E_2 on atoms $x_1, \ldots, x_n$, respectively, (which is GapP-hard) to computing $weight_h(\mathcal{D}_w)$. For this, we assume without loss of generality that both E_1 and E_2 are not satisfied if all atoms x_i are false, and redefine $g(s)$ to

$$g(s) = \begin{cases} 2^n & \text{if } s \text{ satisfies } E_1 \wedge \neg E_2, \\ -2^n & \text{if } s \text{ satisfies } \neg E_1 \wedge E_2, \\ 0 & \text{otherwise.} \end{cases}$$

This has the effect that any history $w = s_0, A, s_1$ where $(w, c) \in C$, will contribute zero to $weight_h(D)$ if E_1 and E_2 have the same value for the assignment s_1, and contribute $h(w) \times f(c) = 1$ (resp., $h(w) \times f(c) = -1$) if E_1 is satisfied but not E_2 (resp. E_2 is satisfied but not E_1). In total, $weight_h(D)$ amounts then to $\#(E_1) - \#(E_2)$. As consequence, computing $weight_h(\mathcal{D}_w)$ for general f and g is (modulo some normalization) complete for GapP.

(4) Computing $weight_{qs}(D)$ is more involved than computing weighted histories, that is, computing $weight_h(\mathcal{D}_w)$. Here, we must take modified state rewards $g'(s)$ into account and normalize with $|S_D(B)|$ for certain queries. However, both values are computable in polynomial space, and thus also $f'(q)$ for each query q. Consequently, computing $weight_{qs}(\mathcal{D}_w)$ is in FPSPACE. Like computing $weight_q(\mathcal{D}_w)$, it is also FPSPACE-complete. Note that in order to obtain FPSPACE-hardness, we just require queries of the form (25). For such queries, setting $g(s) = 1$, for all states s, we get $f'(q) = f(q)$ if the query is satisfied, and $f'(q) = 0$ otherwise. This implies that $weight_{qs}(\mathcal{D}_w)$ is a generalization of $weight_q(\mathcal{D}_w)$, and they coincide on instances, where $g(s) = 1$, for all states s, and where $c \in C$ implies c is of form (25). □

In comparison, $weight_s(\mathcal{D}_w)$ and $weight_h(\mathcal{D}_w)$ are of the same computational degree of difficulty, while $weight_q(\mathcal{D}_w)$ and $weight_{qs}(\mathcal{D}_w)$ are harder under common complexity hypotheses. For queries where nesting of formulas is bounded by a constant, the complexity drops below FPSPACE.

6.2 Computation given D, C, and states S of D

Informally, a source of complexity is that D may specify an exponentially large transition diagram $T(D)$. If $T(D)$ is given, then each of the four weights can be calculated in polynomial time. In fact, not the whole transition diagram is needed, but only a *relevant part*, denoted $T_C(D)$, which comprises all states and all transitions that involve actions appearing in C.

Now if the state set S is known (e.g., after computation with CCALC [13]) or computable in polynomial time, then $T_C(D)$ is constructible in polynomial time. Indeed, for all states $s, s' \in S$ and every action A occurring in some query, we can test in polynomial time whether $\langle s, A, s' \rangle$ is a legal transition with respect to D; the total number of such triples is polynomial in $|S|$. Then the following result (the second row of Table 1) holds.

Theorem 2 *Let $\mathcal{D}_w = \langle D, C, g, f, m \rangle$ be a weighted action domain, and suppose we are given the set S of states described by D. Then $weight_p(\mathcal{D}_w)$ can be computed in polynomial time, for $p \in \{s, q, h, qs\}$.*

Proof Obviously, computing $weight_s(\mathcal{D}_w)$ on $T_C(D)$ is polynomial. Similarly, computing $weight_q(\mathcal{D}_w)$ is polynomial since for each query q, testing $D \models q$ is polynomial on $T_C(D)$: label each state $s \in S$ bottom up with the subformulas q' of q that are true at s, and evaluate every dynamic query of form (8) by considering all reachable nodes at distance n.

For computing $weight_h(\mathcal{D}_w)$, we can also exploit a labeling technique to avoid considering exponentially many paths in $T_C(D)$ explicitly. First, for a query q of form (14), we label all states s with p_i, $i \in \{0, \ldots, n\}$, such that $s = s_i$ for some history $w = s_0, A_1, s_1, \ldots, A_n, s_n$ satisfying q, in polynomial time. An algorithm for labeling states in this way is given in Fig. 5. It implements a two pass procedure as follows:

1) First label, for each state s, all states s' at distance $i = 0, 1, \ldots, n$ with r_i^s that respect the prefix of some w desired with respect to q such that $s = s_0$ and $s' = s_i$.

Algorithm LABEL$(D, S, q) : L(q)$

Input: An action description, D, its set of states, S, and a query, q, of form (14).
Output: A set, $L(q)$, of states labeled wrt. q.

$R(q) := \{\langle s, r_0^s\rangle \mid s \in S, s \models F_0\}$;
for $i := 1$ to n **do**
  $R(q) := R(q) \cup \{\langle s, r_{i-1}^{s_0}\rangle \in R(q), \langle s, A_i, s'\rangle \in R, s' \models F_i\}$;
$L(q) := \{\langle s, p_n\rangle \mid \langle s, r_n^{s_0}\rangle \in R(q)\}$;
for $i := n - 1$ to 0 **do**
  $L(q) := L(q) \cup \{\langle s, r_i^{s_0}\rangle \in R(q), \langle s', p_{i+1}\rangle \in L(q), \langle s, A_{i+1}, s'\rangle \in R\}$;
return $L(q)$;

Fig. 5 An algorithm for labeling states

2) Then, going backwards from states labeled with r_s^n, turn each r_i^s $(i = n, n - 1, \ldots, 0)$ into p_i.

Now for $i = n, n - 1, \ldots, 0$ and each state s labeled with p_i, we can easily compute the sum of the utilities $u(w')$ of all suffixes $w' = s_i, A_{i+1}, s_{i+1}, \ldots, A_n, s_n$ of some history w satisfying c such that $s_i = s$. In particular, $u_0^*(s)$ is the sum of all utilities $u(w)$ of histories that start at s and satisfy q. Exploiting this, $weight_h(D)$ is then readily computed by rearranging the sum of its definition: For each relevant query c of form (14), sum up the $u_0^*(\cdot)$ values at all states and multiply the result with $f(c)$. This gives one summand of a sum to build over all relevant queries [i.e., queries of form (14)]. This algorithm to compute $weight_h(\mathcal{D}_w)$ is depicted in Fig. 6. Note that, for each decision problem of the form $\langle s, A, s'\rangle \in R$, s and A are fixed. Hence, it is solvable in polynomial time.

Algorithm WEIGHT$_{\mathrm{H}}(\mathcal{D}_w, S) : w_h$

Input: A weighted action domain, $\mathcal{D}_w$, and its set of states, S.
Output: The weight $w_h = weight_h(\mathcal{D}_w)$.

$w_h := 0$;
for $q \in C$ such that q of form (14) **do**
  $L(q) :=$ LABEL(D, S, q);
  $U(q) := \{\langle u_n^*(s), g(s)\rangle \mid \langle s, p_n\rangle \in L(q)\}$;
  for $i := n - 1$ to 0 **do**
    for $\langle s, p_i\rangle \in L(q)$ **do**
      $tr(s) := \{\langle t, v\rangle \mid t = \langle s, A_{i+1}, s'\rangle \in R, \langle u_{i+1}^*(s'), v\rangle \in U(q)\}$;
      $U(q) := U(q) \cup \{\langle u_i^*(s), v_i\rangle \mid v_i = \Sigma_{tr(s)} g(s) + m(t) \times v\}$;
  $w_h \mathrel{+}= f(q) \times \Sigma_{\langle u_0^*(s), v\rangle \in U(q)} v$;
return w_h;

Fig. 6 An algorithm for computing $weight_h(\mathcal{D}_w)$, given S

Using the same techniques as for $weight_h(\mathcal{D}_w)$, we can compute $g'(s)$ for each state s in polynomial time on $T_C(D)$ and also $|S_D(B)|$. Therefore, also $weight_{qs}(\mathcal{D}_w)$ is computable in polynomial time in this case. □

Example 9 Consider, for instance, the action description $D^{(3)}$ (Fig. 4) in Example 6; take s_0 and s_1 as specified in Example 6. For query (4), in the first pass of the labeling process, state s_0 is labeled with $r_0^{s_0}, r_1^{s_1}$; and state s_1 is labeled with $r_0^{s_1}, r_1^{s_0}$; in the second pass, both states s_0 and s_1 are labeled with p_0 and p_1. Given the utility function and transition model as in Example 6 (i.e., as (17) and as (19), respectively), and assuming a weight of $f(c) = 3$ for the query, summing up we obtain:

$$
\begin{aligned}
u_1^*(s_0) &= g(s_0) = 2,\\
u_1^*(s_1) &= g(s_1) = 2,\\
u_0^*(s_0) &= g(s_0) + m(\langle s_0, \{PushPB_{RC}\}, s_1\rangle) \times u_1^*(s_1) = 3,\\
u_0^*(s_1) &= g(s_1) + m(\langle s_1, \{PushPB_{RC}\}, s_0\rangle) \times u_1^*(s_0) = 4.
\end{aligned}
$$

And in total

$$f(c) \times \left(u_0^*(s_0) + u_0^*(s_1)\right) = 21,$$

as the summand for the query, c, considered (and therefore as the value for $weight_h(D^{(3)})$ since c is the only query considered in this example).

Finally, if the state space S is not large, i.e., $|S|$ is polynomially bounded, S can be computed with the help of an NP-oracle in polynomial time; in fact, this is possible with parallel NP oracles queries, and thus computing S is in the respective class $\mathrm{FP}_{\parallel}^{\mathrm{NP}}$. The following theorem summarizes these results (the third row of Table 1):

Theorem 3 *Given a weighted action domain $\mathcal{D}_w = \langle D, C, g, f, m\rangle$, and the set S of states described by D, such that $|S|$ is polynomially bounded. Then computing $weight_p(\mathcal{D}_w)$ is in $FP_{\parallel}^{\mathrm{NP}}$, for $p \in \{s, q, h, qs\}$.*

On the other hand, it is unlikely that any of the weight functions is tractable if $|S|$ is polynomially bounded. Indeed, solving SAT under the assertion that the given formula has at most one model (which is still considered to be intractable, cf. [10]) is reducible to computing $weight_p(\mathcal{D}_w)$, for each $p \in \{s, q, h, qs\}$.

7 Discussion and conclusion

In this paper, we have considered the issue of assigning weights to action descriptions, so that one can compare action descriptions by means of their weights. To this end, we have presented four different weights which are defined for the semantics of action descriptions in terms of their transition diagrams, based on preferences over states and transitions, on preferences over conditions, and on probabilities of transitions. We have then illustrated the usefulness of such a semantically oriented approach of comparing action descriptions, for the problem of updating an action

description, in comparison with the syntactic approach of [2], and for the problem of elaboration tolerance. Finally, we have studied computational aspects of the measures, and we have characterized their computational complexity in terms of well-known classes from the literature.

The use of weights for expressing preferences among different solutions to a problem is ubiquitous in AI and has been used in various domains, including configuration, planning, diagnosis, knowledge base integration, abductive reasoning, and agent decision making cf. [8, 14–16]. The desire for expressing preferences in this way also led to generic formalisms and tools for KR which offer this capability, including weighted CSP, optimal answer set engines, and (weighted) MAX-SAT solvers [17–20]. However, to our knowledge, no weight-based approaches for assessing action descriptions had been proposed so far. The semantic view which we have taken here brings in a static and a dynamic component, in terms of states and (sequences of) transitions, which is more complex than in other problems like configuration or diagnosis, where usually only a static component plays a role. Although we have restricted ourselves to a fragment of action language $\mathcal{C}$, since every transition diagram can be represented, the approach equally applies to (richer) action languages with transition diagram semantics. Considering the full language $\mathcal{C}$ will increase the complexity of computing the weights in some cases (like, e.g., in the case of the second row of Table 1, where D, C, and S are part of the input); note that verifying whether a given transition belongs to the transition diagram of a given action description is no longer polynomial for full $\mathcal{C}$ (in fact, this is coNP-hard). However, in other cases our proofs can be suitably adapted to establish a similar result for full $\mathcal{C}$ (like, e.g., computing $weight_q$ and $weight_{qs}$ in the case of the first row of Table 1, where D and C are part of the input). A detailed study of the full language $\mathcal{C}$ and further fragments of it remains for future work.

A more general approach to comparing action descriptions would be in terms of a preference relation (i.e., a reflexive and transitive binary relation) $\preceq$ on the set of action descriptions for a given action signature, and to single out the desired action descriptions D as the most preferred ones, such that no action description D' exists where $D \preceq D'$ but $D' \not\preceq D$. Weight-based comparison as in the update approach can then be modeled (on a restricted set of elements) by $D \preceq D'$ iff $weight(D) \leq weight(D')$. However, under a general preference relation, many action descriptions D and D' may be incomparable, and thus intuitively many action descriptions might be most preferred. In particular, this may happen if $\preceq$ is pointwise composed of several "local" preference relations $\preceq_1, \ldots, \preceq_n$ which do not break ties between different alternatives; in this case, 2^n many most preferred action descriptions may emerge. On the other hand, weight-based comparison induces a modular ordering of the action descriptions, where each pair D, D' of action descriptions is comparable. Therefore, weight-based comparison is intuitively more discriminative, and yields less most preferred action descriptions.

Several issues remain for future work. The manual computation of the weights above quickly becomes cumbersome, and an implementation, which can be based on the complexity characterizations and algorithms which we have outline here, will be useful. It will enable a study of the measures on larger actions descriptions, and to gain experience about their behavior under different parameter settings. To this end, an interactive tool for defining and evaluating the weight measures would be

desirable. Other issues concern efficient algorithms for restricted problem classes, and to search for further weight measures.

Acknowledgements This work has been supported by the Austrian Science Fund (FWF) grant P16536-N04 and the European Commission grant FET-2001-37004 WASP. We are grateful to the reviewers (also of preliminary versions) of this paper for their constructive and helpful comments.

Appendices

Appendix 1: Example Yale shooting domain

Consider the following three formalizations of the Yale shooting domain [21]:

D_1 :
caused *Loaded* **after** *Load*
caused ¬*Loaded* **after** *Shoot*
caused ¬*Alive* **after** *Shoot* ∧ *Loaded*
caused *False* **after** *Shoot* ∧ *Load*
inertial *Loaded*, ¬*Loaded*, *Alive*, ¬*Alive*

D_2 :
caused *Loaded* **after** *Load*
caused ¬*Loaded* **after** *Shoot*
caused ¬*Alive* **after** *Shoot*
caused *False* **after** *Shoot* ∧ ¬*Loaded*
caused *False* **after** *Load* ∧ *Loaded*
inertial *Loaded*, ¬*Loaded*, *Alive*, ¬*Alive*

D_3 :
caused *Loaded* **if** *Loaded* **after** *Loaded*
caused ¬*Loaded* **after** *Shoot*
caused ¬*Alive* **after** *Shoot*
caused *False* **after** *Shoot* ∧ ¬*Loaded*
caused *False* **after** *Load* ∧ *Loaded*
inertial *Loaded*, ¬*Loaded*, *Alive*, ¬*Alive*

and the following set C of queries:

c_1 : **SOMETIMES evolves** *Load*; *True*; *Shoot*; *True*; *Load*; *Loaded*
c_2 : **ALWAYS holds** *Loaded* ⊃ **necessarily** *False* **after** *Load*

where $f(c_1) = 5$ and $f(c_2) = -10$. The first query expresses a desired property: after loading the gun, shooting, and loading again, the gun is loaded. The second query above expresses an undesired property: the gun can not be loaded when it is already loaded.

Take $weight = weight_{qs}$. Consider the following utility function g:

$$g(s) = \begin{cases} 3 \text{ if } Alive \in s \\ 1 \text{ otherwise,} \end{cases}$$

for every state s. Suppose that for each transition $\langle s, A, s' \rangle$ in R,

$$m(\langle s, A, s' \rangle) = \begin{cases} 0.5 \text{ if } Alive \in s \text{ and } Shoot \notin A \\ 0.3 \text{ otherwise.} \end{cases}$$

Let us denote by $s_0, \ldots, s_3$ the following states:

$$s_0 = \{Alive, Loaded\}, s_1 = \{Alive, \neg Loaded\},$$
$$s_2 = \{\neg Alive, Loaded\}, s_3 = \{\neg Alive, \neg Loaded\};$$

and $w_0, \ldots, w_3$ the following histories:

$$w_0 = s_0, \mathit{Load}, s_0, \mathit{Shoot}, s_3, \mathit{Load}, s_2$$
$$w_1 = s_1, \mathit{Load}, s_0, \mathit{Shoot}, s_3, \mathit{Load}, s_2$$
$$w_2 = s_2, \mathit{Load}, s_2, \mathit{Shoot}, s_3, \mathit{Load}, s_2$$
$$w_3 = s_3, \mathit{Load}, s_2, \mathit{Shoot}, s_3, \mathit{Load}, s_2$$
$$w_4 = s_1, \mathit{Load}, s_0, \mathit{Shoot}, s_3, \mathit{Load}, s_3$$

for which the utilities are computed as follows:

w	i	$u_i(w)$
w_0	3	$g(s_2) = 1$
	2	$g(s_3) + m(\langle s_3, \{\mathit{Load}\}, s_2\rangle) \times u_3(w_0) = 1 + 0.3 = 1.3$
	1	$g(s_0) + m(\langle s_0, \{\mathit{Shoot}\}, s_3\rangle) \times u_2(w_0) = 3 + 0.3 \times 1.3 = 6.9$
	0	$g(s_0) + m(\langle s_0, \{\mathit{Load}\}, s_0\rangle) \times u_1(w_0) = 3 + 0.5 \times 6.9 = 6.45$
w_1	3	$g(s_2) = 1$
	2	$g(s_3) + m(\langle s_3, \{\mathit{Load}\}, s_2\rangle) \times u_3(w_0) = 1 + 0.3 = 1.3$
	1	$g(s_0) + m(\langle s_0, \{\mathit{Shoot}\}, s_3\rangle) \times u_2(w_0) = 3 + 0.3 \times 1.3 = 6.9$
	0	$g(s_1) + m(\langle s_0, \{\mathit{Load}\}, s_0\rangle) \times u_1(w_0) = 3 + 0.5 \times 6.9 = 6.45$
w_2	3	$g(s_2) = 1$
	2	$g(s_3) + m(\langle s_3, \{\mathit{Load}\}, s_2\rangle) \times u_3(w_0) = 1 + 0.3 = 1.3$
	1	$g(s_2) + m(\langle s_2, \{\mathit{Shoot}\}, s_3\rangle) \times u_2(w_0) = 1 + 0.3 \times 1.3 = 4.9$
	0	$g(s_2) + m(\langle s_2, \{\mathit{Load}\}, s_2\rangle) \times u_1(w_0) = 1 + 0.3 \times 4.9 = 2.47$
w_3	3	$g(s_2) = 1$
	2	$g(s_3) + m(\langle s_3, \{\mathit{Load}\}, s_2\rangle) \times u_3(w_0) = 1 + 0.3 = 1.3$
	1	$g(s_2) + m(\langle s_2, \{\mathit{Shoot}\}, s_3\rangle) \times u_2(w_0) = 1 + 0.3 \times 1.3 = 4.9$
	0	$g(s_3) + m(\langle s_3, \{\mathit{Load}\}, s_2\rangle) \times u_1(w_0) = 1 + 0.3 \times 4.9 = 2.47$
w_4	3	$g(s_3) = 1$
	2	$g(s_3) + m(\langle s_3, \{\mathit{Load}\}, s_3\rangle) \times u_3(w_0) = 1 + 0.3 = 1.3$
	1	$g(s_0) + m(\langle s_0, \{\mathit{Shoot}\}, s_3\rangle) \times u_2(w_0) = 3 + 0.3 \times 1.3 = 6.9$
	0	$g(s_1) + m(\langle s_1, \{\mathit{Load}\}, s_0\rangle) \times u_1(w_0) = 3 + 0.5 \times 6.9 = 6.45$

That is,

$$u(w_0) = u(w_1) = u(w_4) = 6.45, u(w_2) = u(w_3) = 2.47.$$

For D_1, $H_C = \{(w_0, c_1), (w_1, c_1), (w_2, c_1), (w_3, c_1)\}$. Then, the new utility function g' can be computed as follows:

$$g'(s_0) = 6.45 \quad g'(s_1) = 6.45 \quad g'(s_2) = 2.47 \quad g'(s_3) = 2.47;$$

and the new weights f' of queries are computed as follows:

$$f'(c_1) = f(c_1) \times (g'(s_0)+g'(s_1)+g'(s_2)+g'(s_3))/4 = 5 \times 17.84/4 = 22.3$$
$$f'(c_2) = f(c_2) \times 0 = 0.$$

Then $\mathit{weight}_{qs}(D_1) = 22.3$.

For D_2, $H_C = \{(w_1, c_1), (w_3, c_1)\}$. Then, the new utility function g' can be computed as follows:

$$g'(s_0) = 3 \quad g'(s_1) = 6.45 \quad g'(s_2) = 1 \quad g'(s_3) = 2.47;$$

and the new weights f' of queries are computed as follows:

$$f'(c_1) = f(c_1) \times (g'(s_1) + g'(s_3))/2 = 5 \times 4.46 = 22.3$$
$$f'(c_2) = f(c_2) \times (g'(s_0) + g'(s_2)) = -10 \times 4 = -40.$$

Then $weight_{qs}(D_2) = -17.7$.

For D_3, $H_C = \{(w_1, c_1), (w_3, c_1), (w_4, c_1)\}$. Then, the new utility function g' can be computed as follows:

$$g'(s_0) = 3 \quad g'(s_1) = 6.45 + 2.47 = 8.92 \quad g'(s_2) = 1 \quad g'(s_3) = 2.47;$$

and the new weights f' of queries are computed as follows:

$$f'(c_1) = f(c_1) \times (g'(s_1) + g'(s_3))/2 = 5 \times (8.92 + 2.74)/2 = 5 \times 5.83 = 29.15$$
$$f'(c_2) = f(c_2) \times (g'(s_0) + g'(s_2)) = -10 \times 4 = -40.$$

Then $weight_{qs}(D_3) = -10.85$. Therefore, D_1 is more preferable than D_2 and D_3. Indeed, although these descriptions satisfy the desired property (c_1) to some extent, only D_1 does not satisfy the undesired property.

---

caused $At(Robot, r)$ **after** $Walk(r)$
caused $IsHolding(b, g)$ **after** $PickUp(b, g) \wedge \neg \bigvee_{b_1} IsHolding(b_1, g)$
caused $\neg IsHolding(b, g)$ **after** $Drop(b, g) \wedge IsHolding(b_1, g)$
caused $OnFloor(b)$ **after** $Drop(b, g)$
caused $Color(b, c)$ **after** $PaintBall(b, c) \wedge At(Robot, Room\ 3) \wedge At(b, Room\ 3)$
caused $OnFloor(b)$ **if** $OnFloor(b)$
caused $\neg At(o, r_1)$ **if** $At(o, r) \qquad (r \neq r_1)$
caused $\neg Color(b, c_1)$ **if** $Color(b, c) \qquad (c \neq c_1)$
caused $False$ **if** $\neg \bigvee_r At(o, r)$
caused $False$ **if** $\neg \bigvee_c At(b, c)$
inertial $At(o, r), Color(b, c), IsHolding(b, g), \neg IsHolding(b, g)$
caused $False$ **after** $a \wedge a_1 \qquad (a < a_1)$

Fig. 7 A description for the grippers domain (D_1)

caused $At(Robot, r)$ **after** $Walk(r)$
caused $False$ **after** $Walk(r) \wedge At(Robot, r)$

caused $IsHolding(b, g)$ **after** $PickUp(b, g)$
caused $False$ **after** $PickUp(b, g) \wedge \bigvee_{b_1} IsHolding(b_1, g)$
caused $False$ **after** $PickUp(b, g) \wedge \neg(\bigvee_{r}(At(Robot, r) \wedge At(b, r)))$

caused $\neg IsHolding(b, g)$ **after** $Drop(b, g)$
caused $False$ **after** $Drop(b, g) \wedge \neg IsHolding(b_1, g)$

caused $Color(b, c)$ **after** $PaintBall(b, c)$
caused $False$ **after** $PaintBall(b, c) \wedge \neg(At(Robot, Room3) \wedge At(b, Room3))$

caused $OnFloor(b)$ **if** $OnFloor(b)$
caused $\neg OnFloor(b)$ **if** $\bigvee_{g} IsHolding(b, g)$

caused $\neg At(o, r_1)$ **if** $At(o, r) \qquad (r \neq r_1)$
caused $\neg At(b, r)$ **if** $At(Robot, r) \wedge \bigvee_{g} IsHolding(b, g)$

caused $\neg Color(b, c_1)$ **if** $Color(b, c) \qquad (c \neq c_1)$

caused $False$ **if** $\neg \bigvee_{r} At(o, r)$
caused $False$ **if** $\neg \bigvee_{c} At(b, c)$

inertial $At(o, r), Color(b, c), IsHolding(b, g), \neg IsHolding(b, g)$

caused $False$ **after** $a \wedge a_1 \qquad (a < a_1)$

Fig. 8 Another description for the grippers domain (D_2)

Appendix 2: Example gripper domain

Consider the following variation of the gripper domain [22]. There are three balls, each located in one of three rooms. There is a robot with two grippers. It can carry a

caused $At(Robot, r)$ **after** $Walk(r) \wedge \neg At(Robot, r)$
caused $IsHolding(b, g)$ **after** $PickUp(b, g) \wedge \neg \bigvee_{b_1} IsHolding(b_1, g)$
caused $False$ **after** $PickUp(b, g) \wedge \neg(\bigvee_{r}(At(Robot, r) \wedge At(b, r)))$

caused $\neg IsHolding(b, g)$ **after** $Drop(b, g)$
caused $False$ **after** $Drop(b, g) \wedge \neg IsHolding(b_1, g)$

caused $Color(b, c)$ **after** $PaintBall(b, c)$
caused $False$ **after** $PaintBall(b, c) \wedge \neg(At(Robot, Room3) \wedge At(b, Room3))$

caused $OnFloor(b)$ **if** $OnFloor(b)$
caused $\neg OnFloor(b)$ **if** $\bigvee_{g} IsHolding(b, g)$

caused $\neg At(o, r_1)$ **if** $At(o, r) \qquad (r \neq r_1)$
caused $\neg At(b, r)$ **if** $At(Robot, r) \wedge \bigvee_{g} IsHolding(b, g)$

caused $\neg Color(b, c_1)$ **if** $Color(b, c) \qquad (c \neq c_1)$

caused $False$ **if** $\neg \bigvee_{r} At(o, r)$
caused $False$ **if** $\neg \bigvee_{c} At(b, c)$

inertial $At(o, r), Color(b, c), IsHolding(b, g), \neg IsHolding(b, g)$

caused $False$ **after** $a \wedge a_1 \qquad (a < a_1)$

Fig. 9 Yet another description for the grippers domain (D_3)

ball in each. The available actions are picking up, dropping, and painting balls, and moving between rooms. Suppose that the paint is available only in $Room_3$. Consider the descriptions D_1, D_2, D_3 of this domain shown in Figs. 7–9, which use schematic variables that range over constants as follows:

Variable	Constants
o	$Robot, Ball_1, Ball_2, Ball_3$
b, b_1	$Ball_1, Ball_2, Ball_3$
g, g_1	$Gripper_1, Gripper_2$
r, r_1, r_2	$Room_1, Room_2, Room_3$
c, c_1	$Red, White, Blue.$

Consider a set C consisting of the following queries:

$$p_1 : \textbf{ALWAYS } \bigwedge_r (\textbf{holds } At(Robot, r) \supset \textbf{necessarily } \neg At(Robot, r) \textbf{ after } Walk(r))$$

$$p_2 : \textbf{SOMETIMES } \bigwedge_{b,g} (\textbf{holds } \neg IsHolding(b, g) \wedge \textbf{possibly } True \textbf{ after } Drop(b))$$

$$p_3 : \textbf{ALWAYS holds } \bigwedge_b (At(b, Room_1) \wedge Color(b, White) \wedge OnFloor(b)) \wedge At(Robot, Room_3) \supset \textbf{necessarily } \bigwedge_b (At(b, Room_2) \wedge At(Robot, Room_3)) \wedge \bigvee_b Color(b, Red) \wedge \bigvee_b Color(b, White) \wedge \bigvee_b Color(b, Blue)$$
$$\textbf{after } Walk(Room_1); PickUp(Ball_1, Gripper_1); PickUp(Ball_3, Gripper_2); Walk(Room_2); Drop(Ball_1, Gripper_1); Walk(Room_1); PickUp(Ball_2, Gripper_1); Walk(Room_3); PaintBall(Ball_2, Blue); PaintBall(Ball_3, Red); Walk(Room_2); Drop(Ball_2, Gripper_1); Drop(Ball_3, Gripper_2); Walk(Room_3),$$

where $f(p_1) = 2$, $f(p_2) = -3$, and $f(p_3) = 1$. The first query expresses that after the robot walks to some location, it is not at its current location anymore. The second one expresses that the action of dropping a ball is possible even when the robot is not holding that ball. The third one expresses the presence of a trajectory. The first and the third condition are desired while the second is not.

With the semantic approach based on weighted queries (taking $weight = weight_q$), the weights of the action descriptions are computed as follows:

$$\begin{aligned} weight_q(D_1) &= f(p_2) = -3 \\ weight_q(D_2) &= f(p_1) + f(p_3) = 2 + 1 = 3 \\ weight_q(D_3) &= f(p_3) = 1. \end{aligned}$$

Therefore, D_2 is the most preferable description. Indeed, it is the only description that entails both desired queries and does not entail the undesired one.

References

1. Gelfond, M., Lifschitz, V.: Action languages. ETAI **3**, 195–210 (1998)
2. Eiter, T., Erdem, E., Fink, M., Senko, J.: Updating action domain descriptions. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 418–423 (2005)

3. Alferes, J.J., Banti, F., Brogi, A.: From logic programs updates to action description updates. In: Proceedings Computational Logic in Multi-Agent Systems (CLIMA V). LNCS, vol. 3487, pp. 52–77. Springer (2004)
4. Herzig, A., Perrussel, L., Varzinczak, I.: Elaborating domain descriptions. In: Brewka, G., et al. (eds.) Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 397–401. IOS Press (2006)
5. McCarthy, J.: Elaboration tolerance. In: Proceedings of the Symposium on Logical Formalizations of Commonsense Reasoning (CommonSense) (1998)
6. Amir, E.: Towards a formalization of elaboration tolerance: adding and deleting axioms. In: Frontiers of Belief Revision. Kluwer (2000)
7. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: preliminary report. In: Proceedings of the National Conference on AI (AAAI), pp. 623–630 (1998)
8. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn Prentice Hall (2002)
9. Eiter, T., Erdem, E., Fink, M., Senko, J.: Resolving conflicts in action descriptions. In Brewka, G., et al. (eds.) Proceedings ECAI, pp. 367–371. IOS Press (2006)
10. Papadimitriou, C.: Computational Complexity. Addison-Wesley (1994)
11. Fenner, S.A., Fortnow, L., Kurtz, S.A.: Gap-definable counting classes. J. Comput. Syst. Sci. **48**, 116–148 (1994)
12. Gupta, S.: Closure properties and witness reduction. J. Comput. Syst. Sci. **50**, 412–432 (1995)
13. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artif. Intell. **153**, 49–104 (2004)
14. Lin, J.: Integration of weighted knowledge bases. Artif. Intell. **83**, 363–378 (1996)
15. Eiter, T., Gottlob, G.: The Complexity of Logic-Based Abduction. ACM Journal **42**, 3–42 (1995)
16. Stroe, B., Subrahmanian, V.S., Dasgupta, S.: Optimal status sets of heterogeneous agent programs. In: Dignum, F., et al. (eds.) Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 709–715 (2005)
17. Larrosa, J., Schiex, T.: Solving weighted CSP by maintaining arc consistency. Artif. Intell. **159**, 1–26 (2004)
18. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**, 499–562 (2006)
19. Syrjänen, T., Niemelä, I.: The Smodels system. In: Eiter, T., et al. (eds.): Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). LNCS, vol. 2173, pp. 434–438. Springer (2001)
20. Xing, Z., Zhang, W.: Maxsolver: an efficient exact algorithm for (weighted) maximum satisfiability. Artif. Intell. **164**, 47–80 (2005)
21. Hanks, S., McDermott, D.: Nonmonotonic logic and temporal projection. Artif. Intell. **33**, 379–412 (1987)
22. McDermott, D.: AIPS-98 planning competition results. (1998)