A Language for Default Reasoning about Actions

Hannes Strass[1] and Michael Thielscher[2]

[1] Computer Science Institute, University of Leipzig
strass@informatik.uni-leipzig.de
[2] School of Computer Science and Engineering, The University of New South Wales
mit@cse.unsw.edu.au

Abstract. Action languages allow for a concise representation of actions and their effects while at the same time being easily readable and writable for humans. In this paper, we introduce $\mathcal{D}$, the first action language that centres around *default* reasoning about actions and change. It allows to specify normal Reiter-style defaults and provides a semantics that has at its core the groundedness of conclusions. This is not only of use for default conclusions, but also for $\mathcal{D}$'s solution to the ramification problem. Additionally, our language does not suffer from Yale Shooting-like counterexamples since it uses different mechanisms for default persistence and default change. The answer set programming paradigm is used as the basis of $\mathcal{D}$'s implementation, which we prove sound and complete with respect to the language's semantics. We finally present a showcase application for the language: its straightforward solution to the qualification problem.

1 Introduction

Action languages are simple declarative languages for describing actions and their effects on the world. The first ever action language, $\mathcal{A}$ [1], was introduced in 1993 by Vladimir Lifschitz, to whom we dedicate this article on the occasion of his 65th birthday, and his colleague Michael Gelfond to enhance the traditionally example-oriented mode of operation of reasoning about actions research.

This original language $\mathcal{A}$ allowed only for very basic action descriptions, and so was soon extended to $\mathcal{B}$ [2], which handles indirect effects through static laws. A further extension came with $\mathcal{C}$ [3], enabling convenient description of concurrency and non-determinism. The language $\mathcal{E}$ [4] then introduced an explicit notion of time into action languages, which had hitherto only possessed an implicit time structure defined through state transition systems.

In this paper, we introduce the action language $\mathcal{D}$ for *default* reasoning about actions. It is a useful addition to the existing $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{E}$ – but not just as "yet another action language": it is the first language that centres around default reasoning about actions. We argue that this is of practical relevance for agents with incomplete knowledge about their domain.

Two existing action languages do allow for some forms of default reasoning, $\mathcal{C}+$ [5] and $\mathcal{K}$ [6]. Alas, in both languages defaults can be at odds with the solution of the frame problem, since the same mechanism is used for static defaults and temporal persistence defaults. We have found that certain intuitive ways of modelling a domain do not yield the intuitively expected answers: As an example, imagine a simple domain where two inertial fluents, Bird and Flies, express whether there is a bird in the domain and whether it flies. We want to state that birds fly by default and look at the interaction of this default with temporal persistence. A seemingly straightforward $\mathcal{C}+$ specification of this domain is

$$\begin{array}{l}\textbf{inertial}\ \mathsf{Bird}\\ \textbf{inertial}\ \mathsf{Flies}\\ \textbf{default}\ \mathsf{Flies}\ \textbf{if}\ \mathsf{Bird}\end{array}$$

Now consider an abnormal initial time point 0 where Bird is true and Flies is false, $\{0{:}\mathsf{Bird}, 0{:}\neg\mathsf{Flies}\}$. What holds at the next time point when no action occurs at 0? Intuition suggests that the world does not change when nothing happens, hence $\neg\mathsf{Flies}$ should still hold. However, $\mathcal{C}+$ does not conform with this intuition in that it admits – along with the model where Flies still does not hold after waiting (due to persistence) – an additional model where the abnormal bird magically learns to fly (by default).[3] In $\mathcal{D}$, the straightforward specification of this simple domain will yield the intuitively expected inference that abnormality persists when not caused otherwise.

Another observation we made is that some conclusions in $\mathcal{C}+$ circularly justify themselves and are thus not grounded: The statements

$$\begin{array}{l}\textbf{default}\ \mathsf{Rain}\ \textbf{if}\ \mathsf{Wet}\\ \textbf{default}\ \mathsf{Wet}\ \textbf{if}\ \mathsf{Rain}\end{array}$$

say that rain and wet grass usually go hand in hand. For time point 0, they expand to the formulas $0{:}\mathsf{Rain} \Leftarrow 0{:}\mathsf{Rain} \wedge 0{:}\mathsf{Wet}$ and $0{:}\mathsf{Wet} \Leftarrow 0{:}\mathsf{Wet} \wedge 0{:}\mathsf{Rain}$ of nonmonotonic causal logic [5]. If nothing further is known about the time point, $\{0{:}\mathsf{Rain}, 0{:}\mathsf{Wet}\}$ is a model for the two formulas, where rain and wet grass appear without any evidence for either. In contrast, all conclusions in $\mathcal{D}$ are grounded in the sense that they have a non-cyclic derivation from definite knowledge. For the same reason, $\mathcal{D}$ also offers a solution to the ramification problem that can deal with cyclic effect dependencies.[4]

Together with our new action language $\mathcal{D}$, we introduce a query language for prediction. We present $\mathcal{D}$'s implementation, which uses answer set programming (ASP) as back-end, and is provably sound and complete for the semantics of $\mathcal{D}$. Finally, we demonstrate the capabilities of $\mathcal{D}$ by developing a solution for the qualification problem [7] entirely inside the language itself.

[3] Of course, this does not suggest that the domain cannot otherwise be modelled in $\mathcal{C}+$. If the cause of the bird's abnormality is explicitly named in the language (say, by inertial fluent AbBird) and $0{:}\mathsf{AbBird}$ is added to the domain along with the static law **caused** $\neg\mathsf{Flies}$ **if** AbBird, then $\mathcal{C}+$ treats the example right.

[4] It is even possible to use more general ramification rules than the ones considered here. Ramification is however not our main topic here, so we keep that simple.

2 Action Language $\mathcal{D}$

We assume a sorted logical signature that contains the sorts FLUENT (for domain properties that change over time) and ACTION (for actions that affect the state of the world). Like its predecessor $\mathcal{A}$, the language $\mathcal{D}$ assumes inertia for all fluent properties unless there is an explicit cause for change.

2.1 Syntax

A specification of an action domain in $\mathcal{D}$ consists of two parts: the first part contains general knowledge about the domain – action preconditions, action effects, ramifications, state defaults. The second part contains information about the initial time point of a specific domain instance. The vocabulary to describe a domain is given by non-empty sets FLUENT and ACTION of constant symbols.

Definition 1. *Assume a fixed logical signature with sorts* FLUENT *and* ACTION. *For a fluent F, a* fluent literal *is of the form F or $\neg F$. Define $\overline{F} \overset{\text{def}}{=} \neg F$ and $\overline{\neg F} \overset{\text{def}}{=} F$, and extend this notation to sets of fluent literals.*

Let A be an ACTION*, F be a* FLUENT*, K, L be fluent literals and C be a finite set of fluent literals. A* statement *can be:*

- *a* precondition statement*:* <u>`possible`</u> A <u>`if`</u> C
- *a* direct effect statement*:* <u>`action`</u> A <u>`causes`</u> L <u>`if`</u> C
- *an* indirect effect statement*:* <u>`effect`</u> K <u>`causes`</u> L <u>`if`</u> C
- *a* default statement*:* <u>`normally`</u> L <u>`if`</u> C

An initial state axiom *is of the form* <u>`initially`</u> L*. An* action domain specification*, or* domain *for short, is a finite set $\Sigma = \Upsilon \cup \Omega$ where Υ is a set of statements and Ω is a set of initial state axioms.*

In statements of the above form, we will refer to literal L as the *consequent.* If $C = \emptyset$ for a statement, we omit the <u>`if`</u> part in writing. To illustrate $\mathcal{D}$, we use the following running example throughout the paper.

Example 2 (Swipe Card Domain). Imagine an office building where some parts have restricted access through swipe cards. It is possible for an agent who has a card to swipe it through the reader. Normally, the agent has a card, and swiping the card through the reader unlocks the door. Subsequently pushing an unlocked door opens it, unless it is jammed, which may be caused by pushing a locked door. In $\mathcal{D}$, the domain specification of this environment is as follows:

$$\begin{aligned}
\Upsilon_{\mathbb{D}} = \{&\underline{\texttt{possible}}\ \textsf{Swipe}\ \underline{\texttt{if}}\ \{\textsf{HasCard}\},\\
&\underline{\texttt{normally}}\ \textsf{HasCard},\\
&\underline{\texttt{action}}\ \textsf{Swipe}\ \underline{\texttt{causes}}\ \textsf{Swiped},\\
&\underline{\texttt{normally}}\ \neg\textsf{Locked}\ \underline{\texttt{if}}\ \textsf{Swiped},\\
&\underline{\texttt{action}}\ \textsf{Push}\ \underline{\texttt{causes}}\ \textsf{Open}\ \underline{\texttt{if}}\ \{\neg\textsf{Locked}, \neg\textsf{Jammed}\},\\
&\underline{\texttt{action}}\ \textsf{Push}\ \underline{\texttt{causes}}\ \textsf{Jammed}\ \underline{\texttt{if}}\ \{\textsf{Locked}\}\}
\end{aligned}$$

There is no precondition statement for Push, we thus take it to be the trivial one – $\underline{\texttt{possible}}\ \mathsf{Push}\ \underline{\texttt{if}}\ \emptyset$. At an initial time point, the door is locked and not jammed, and the card has not been swiped:

$$\varOmega_{\text{▯}} = \{\underline{\texttt{initially}}\ \mathsf{Locked}, \underline{\texttt{initially}}\ \neg\mathsf{Jammed}, \underline{\texttt{initially}}\ \neg\mathsf{Swiped}\}$$

This concludes the specification of the swipe card domain $\varSigma_{\text{▯}} = \varUpsilon_{\text{▯}} \cup \varOmega_{\text{▯}}$.

$\mathcal{D}$ domains state how the world normally behaves. Due to their syntax being close to natural language, we already have an intuitive understanding of their meaning. In the next section, we will develop a mathematical semantics for $\mathcal{D}$.

2.2 Semantics

Traditional action languages use transition systems to interpret action domains and a Tarski-style entailment relation over so-called histories to solve reasoning problems in these domains [2]. In $\mathcal{D}$, we pay special attention to the *groundedness* of conclusions and thus use a fixed-point based semantics much in the spirit of default logic [8]. The time structure of $\mathcal{D}$'s semantics as defined below is branching. However, this is only for the purpose of this paper and all the definitions we give here can be adjusted to the case of more general time structures.

Definition 3. *Assume a fixed logical signature with sorts* FLUENT *and* ACTION. *An* action sequence α *is a word over the alphabet of* ACTION*s, that is, either* ε *(the empty sequence) or of the form* $\alpha' A$ *for an action sequence* α'.

A scenario S *is a set of pairs* (α, L) *where* α *is an action sequence and* L *is a fluent literal. A scenario* S *is* consistent *iff* $\{(\alpha_1, F), (\alpha_2, \neg F)\} \subseteq S$ *implies* $\alpha_1 \neq \alpha_2$ *for all fluents* F.

Intuitively, a pair (α, L) in a scenario means that L holds after execution of α. For a scenario S and an action sequence α, we use the notation $S(\alpha) \overset{\text{def}}{=} \{L \mid (\alpha, L) \in S\}$ to refer to the set of fluent literals to which α is related. Read as a mapping, the alternative notation $S(\alpha)$ assigns to each action sequence a (possibly incomplete) knowledge state [6], where incompleteness of the state only reflects incomplete knowledge about the real world. Consider again the example:

Example 2 (Continued). In the first scenario S_1, everything works as expected: swiping the card unlocks an initially not jammed door which is then opened by pushing it. Had the agent initially pushed the locked door, it would be jammed.

$$\begin{aligned}
S_1(\varepsilon) &= \{\mathsf{HasCard}, \neg\mathsf{Open}, \mathsf{Locked}, \neg\mathsf{Jammed}, \neg\mathsf{Swiped}\}\\
S_1(\mathsf{Swipe}) &= \{\mathsf{HasCard}, \neg\mathsf{Open}, \neg\mathsf{Locked}, \neg\mathsf{Jammed}, \mathsf{Swiped}\}\\
S_1(\mathsf{SwipePush}) &= \{\mathsf{HasCard}, \mathsf{Open}, \neg\mathsf{Locked}, \neg\mathsf{Jammed}, \mathsf{Swiped}\}\\
S_1(\mathsf{Push}) &= \{\mathsf{HasCard}, \neg\mathsf{Open}, \mathsf{Locked}, \mathsf{Jammed}, \neg\mathsf{Swiped}\}
\end{aligned}$$

In scenario S_2, the card reader does not work and the door remains locked:

$$\begin{aligned}
S_2(\varepsilon) &= \{\mathsf{HasCard}, \neg\mathsf{Open}, \mathsf{Locked}, \neg\mathsf{Jammed}, \neg\mathsf{Swiped}\}\\
S_2(\mathsf{Swipe}) &= \{\mathsf{HasCard}, \neg\mathsf{Open}, \mathsf{Locked}, \neg\mathsf{Jammed}, \mathsf{Swiped}\}
\end{aligned}$$

While scenarios talk about states of the world after execution of actions, there is yet no mention of changes due to action effects. In the following, we define how to determine these effects from the statements of a domain specification. This is easy in the case of direct and indirect effects. For the case of default effects, we use an additional scenario T, which represents a particular context against which default conclusions are checked. We first define when statements are considered applicable in a scenario S with respect to context scenario T.

Definition 4. *Let Σ be a domain, S, T be scenarios for Σ's signature, α be an action sequence and A an action with precondition statement* $\underline{\texttt{possible}}$ A $\underline{\texttt{if}}$ P. *A statement $\sigma \in \Sigma$ is* applicable in S after αA *iff $P \subseteq S(\alpha)$ and*

- $\sigma = \underline{\texttt{action}}\ A\ \underline{\texttt{causes}}\ L\ \underline{\texttt{if}}\ C$ *and* $C \subseteq S(\alpha)$*; or*
- $\sigma = \underline{\texttt{effect}}\ K\ \underline{\texttt{causes}}\ L\ \underline{\texttt{if}}\ C$ *and* $\overline{K} \in S(\alpha)$, $K \in S(\alpha A)$, $C \subseteq S(\alpha)$*; or*
- $\sigma = \underline{\texttt{normally}}\ L\ \underline{\texttt{if}}\ C$ *and (a)* $C \subseteq S(\alpha A)$, *(b)* $\overline{L} \notin T(\alpha A)$ *and (c) one of* $\overline{C} \cap S(\alpha) \neq \emptyset$ *or* $L \in S(\alpha)$.

The direct effects $\Delta_{\Sigma}^{Dir}(S, \alpha A)$, indirect effects $\Delta_{\Sigma}^{Ind}(S, \alpha A)$ *and* default effects $\Delta_{\Sigma,T}^{Def}(S, \alpha A)$ of A after α in S *are the sets of consequent literals L of direct and indirect effect and default statements from Σ that are applicable in S after αA.*

$$\Delta_{\Sigma,T}(S, \alpha A) \stackrel{\text{def}}{=} \Delta_{\Sigma}^{Dir}(S, \alpha A) \cup \Delta_{\Sigma}^{Ind}(S, \alpha A) \cup \Delta_{\Sigma,T}^{Def}(S, \alpha A)$$

The set $\Delta_{\Sigma,T}(S, \alpha A)$ contains all the effects of the action A when it is executed in the state $S(\alpha)$ of domain Σ. Direct effects occur whenever their precondition C was satisfied in the starting state. Indirect effects materialise when the precondition C was satisfied and additionally the trigger literal K changed its truth value during action execution. A default effect L appears whenever the precondition C was satisfied after executing A, the effect is consistent with the context scenario T and the starting state was not abnormal with respect to this default. Note that default effects are the only ones that require the context scenario T.

Example 2 (Continued). For scenario S_1, where the world behaves normally, we obtain $\Delta_{\Sigma}^{Dir}(S_1, \mathsf{Swipe}) = \{\mathsf{Swiped}\}$ and $\Delta_{\Sigma}^{Ind}(S_1, \mathsf{Swipe}) = \emptyset$ along with $\Delta_{\Sigma,S_1}^{Def}(S_1, \mathsf{Swipe}) = \{\neg \mathsf{Locked}\}$. But note that using S_2 as a context scenario yields $\Delta_{\Sigma,S_2}^{Def}(S_1, \mathsf{Swipe}) = \emptyset$, since $\mathsf{Locked} \in S_2(\mathsf{Swipe})$. For the empty scenario we have $\Delta_{\Sigma}^{Dir}(\emptyset, \mathsf{Swipe}) = \emptyset$ since the action precondition $\mathsf{HasCard}$ of Swipe is not satisfied.

Next, we define the set of *non-effects* of an action. Intuitively, the set $\Delta_{\Sigma,T}(S, \alpha A)$ just defined contains all the fluent literals that are guaranteed to hold in S after αA. The non-effects of an action, on the other hand, will contain all the fluent literals that are guaranteed to never possibly hold. For example, if for a fluent F there is no indirect effect statement with F as effect, no default statement that concludes F and also no direct effect statement with effect F for an action A, then we can be sure that A will never make F true.

The next definition formalises this intuition. Note that "inapplicable" below is not the negation of applicable – it rather means "not applicable now or ever." In addition, (in)applicability of a statement is only meaningful for time-points that are connected by applicable actions, that is, actions whose preconditions are fulfilled in the starting state.

Definition 5. *Let Σ be a domain, S be a scenario for Σ's signature, α be an action sequence and A be an action. A statement $\sigma \in \Sigma$ is* inapplicable in S after αA *iff one of*

- $\sigma = \underline{\texttt{action}}\ A\ \underline{\texttt{causes}}\ L\ \underline{\texttt{if}}\ C$ *and* $\overline{C} \cap S(\alpha) \neq \emptyset$*; or*
- $\sigma = \underline{\texttt{effect}}\ K\ \underline{\texttt{causes}}\ L\ \underline{\texttt{if}}\ C$ *and one of: (a)* $K \in S(\alpha)$ *or (b)* $\overline{K} \in S(\alpha A)$ *or (c)* $\overline{C} \cap S(\alpha) \neq \emptyset$*; or*
- $\sigma = \underline{\texttt{normally}}\ L\ \underline{\texttt{if}}\ C$ *and one of (a)* $\overline{C} \cap S(\alpha A) \neq \emptyset$ *or (b)* $\overline{L} \in S(\alpha A)$ *or (c)* $C \subseteq S(\alpha)$ *and* $\overline{L} \in S(\alpha)$.

The non-effects of A after α in S *are* $\Psi_{\Sigma}^{Dir}(S, \alpha A)$, $\Psi_{\Sigma}^{Ind}(S, \alpha A)$ *and* $\Psi_{\Sigma}^{Def}(S, \alpha A)$, *defined as containing those literals L for which all direct and indirect effect and default statements with consequent L are inapplicable in S after αA. Then*

$$\Psi_{\Sigma}(S, \alpha A) \stackrel{\text{def}}{=} \Psi_{\Sigma}^{Dir}(S, \alpha A) \cap \Psi_{\Sigma}^{Ind}(S, \alpha A) \cap \Psi_{\Sigma}^{Def}(S, \alpha A)$$

In particular, $\Psi_{\Sigma}(S, \alpha A)$ will contain a literal L if there is no statement with consequent L. With the presumptions of the previous definition, we finally use the set of non-effects to solve the frame problem: whenever we know F holds before executing A and A cannot make F false, it will still hold afterwards. We thus conclude persistence only in light of evidence that change is impossible.

Definition 6. *Let Σ be a domain, S be a scenario for Σ's signature, α be an action sequence and A an action with precondition statement* $\underline{\texttt{possible}}\ A\ \underline{\texttt{if}}\ P$. *The* persisting fluents of A after α in S *are*

$$\Delta_{\Sigma}^{Frame}(S, \alpha A) \stackrel{\text{def}}{=} \left\{(\alpha A, L) \;\middle|\; (\alpha, L) \in S, P \subseteq S(\alpha), \overline{L} \in \Psi_{\Sigma}(S, \alpha A)\right\}$$

This very cautious treatment of persistence is rooted in allowing incomplete knowledge while dealing with defaults. If we allowed to conclude persistence solely on the fact that the contrary effect did not occur in the context scenario, then undesired interaction between persistence and defaults would be supported as described in the introduction. Taken together, the two previous definitions constitute $\mathcal{D}$'s solution to the frame problem in the presence of defaults.

Example 2 (Continued). In S_1, $\underline{\texttt{action}}$ Push $\underline{\texttt{causes}}$ Jammed $\underline{\texttt{if}}$ {Locked} is inapplicable after SwipePush since $\neg$Locked $\in S_1$(Swipe), which implies that Jammed $\in \Psi_{\Sigma}^{Dir}(S_1, \text{SwipePush})$. Since there are no indirect effect or default statements with consequent Jammed, we get Jammed $\in \Psi_{\Sigma}(S, \text{SwipePush})$, which means that Jammed is a non-effect of the action Push after Swipe. Now $\neg$Jammed holds after Push in S_1 – formally, $\neg$Jammed $\in S_1$(Push) – and therefore $\neg$Jammed $\in \Delta_{\Sigma,S_1}^{Frame}(S_1, \text{SwipePush})$.

The next definition provides the main "workhorse" for $\mathcal{D}$'s semantics. It is an operator that takes a scenario as argument and transforms it into another scenario using statements and axioms from an action domain specification. We will see later that this operator corresponds to the one-step consequence operator from logic programming. First, the set $\mathit{\Delta}_{\mathit{\Omega}}$ puts together the information about the initial state of the domain. Then, $\mathit{\Delta}^{\alpha}_{\mathit{\Sigma},T}(S)$ computes the additional domain information about action sequence α that can be derived from S and the statements in the domain. There, $\mathit{\Delta}^{\varepsilon}_{\mathit{\Sigma},T}(S)$ applies default statements to the initial state of S, while $\mathit{\Delta}^{\alpha A}_{\mathit{\Sigma},T}(S)$ computes all pairs of action sequences and fluent literals that hold due to application of action A after α in S – be they direct effects, indirect effects, default effects or persisting fluents. Finally, $\mathit{\Gamma}_{\mathit{\Sigma},T}(S)$ accumulates the world knowledge and puts it into the resulting scenario. The additional scenario T is the context against which default application is checked.

Definition 7. *Let $\mathit{\Sigma} = \mathit{\Upsilon} \cup \mathit{\Omega}$ be a $\mathcal{D}$ action domain specification and S and T be scenarios.*

$$\begin{aligned}
\mathit{\Delta}_{\mathit{\Omega}} &\overset{\text{def}}{=} \{(\varepsilon, L) \mid \underline{\texttt{initially}}\ L \in \mathit{\Omega}\} \\
\mathit{\Delta}^{\varepsilon}_{\mathit{\Sigma},T}(S) &\overset{\text{def}}{=} \left\{(\varepsilon, L) \;\middle|\; \underline{\texttt{normally}}\ L\ \underline{\texttt{if}}\ C \in \mathit{\Sigma}, C \subseteq S(\varepsilon),\ \overline{L} \notin T(\varepsilon)\right\} \\
\mathit{\Delta}^{\alpha A}_{\mathit{\Sigma},T}(S) &\overset{\text{def}}{=} \left\{(\alpha A, L) \;\middle|\; L \in \mathit{\Delta}_{\mathit{\Sigma},T}(S, \alpha A)\right\} \cup \mathit{\Delta}^{\mathit{Frame}}_{\mathit{\Sigma}}(S, \alpha A) \\
\mathit{\Gamma}_{\mathit{\Sigma},T}(S) &\overset{\text{def}}{=} \mathit{\Delta}_{\mathit{\Omega}} \cup \bigcup_{\alpha \in \text{ACTION}^*} \mathit{\Delta}^{\alpha}_{\mathit{\Sigma},T}(S)
\end{aligned}$$

It is tedious but not hard to show that $\mathit{\Gamma}_{\mathit{\Sigma},T}$ is a monotone operator for any given $\mathit{\Sigma}, T$. We are now ready to define *possible scenarios*, the central notion of $\mathcal{D}$'s semantics. As we will see later on, it corresponds to the notion of an answer set of a logic program. A possible scenario can be reconstructed from the initial state specification using the domain rules at hand.

Definition 8. *Let $\mathit{\Sigma}$ be a domain. A scenario S is* possible *for $\mathit{\Sigma}$ iff it is consistent and the least fixed point of $\mathit{\Gamma}_{\mathit{\Sigma},S}$.*

It is a direct consequence of this definition that possible scenarios of a given domain $\mathit{\Sigma}$ need not be unique and need not necessarily exist at all.

Example 2 (Continued). Our example scenario S_1 for the swipe card domain can be extended to a scenario S_1^* that is possible for $\mathit{\Sigma}_{▯}$ by repeatedly applying $\mathit{\Gamma}_{\mathit{\Sigma}_{▯},S_1}$ until reaching a fixed point. S_2, on the other hand, cannot be extended to a possible scenario. Roughly, one cannot explain $\mathsf{Locked} \in S_2(\mathsf{Swipe}) \subseteq S_2^*(\mathsf{Swipe})$ for any potential candidate S_2^*: there is no statement with consequent Locked, hence persistence would be the only possibility. However, the default statement $\underline{\texttt{normally}}\ \neg\mathsf{Locked}\ \underline{\texttt{if}}\ \mathsf{Swiped}$ is not inapplicable after Swipe in S_2^*, thus $\neg\mathsf{Locked} \notin \mathit{\Psi}_{\mathit{\Sigma}_{▯}}(S_2^*, \mathsf{Swipe})$. The intuition here is that in S_2 the world does not behave normally: the default effect of unlocking the door might materialise, but has not done so in the context. Indeed, S_1^* is the only possible scenario of $\mathit{\Sigma}_{▯}$.

Our action language $\mathcal{D}$ shares with its predecessors $\mathcal{A}$ and $\mathcal{B}$ the principle that the semantics solves the frame problem [2].[5] Domain specifications in $\mathcal{A}$ consist solely in effect statements (written as A `causes` L `if` C), and the semantics is defined by a state transition system that obeys these effect laws and otherwise assumes inertia. Lack of space does not permit to go into details, but it is easy to verify that $\mathcal{D}$ is a proper generalisation of $\mathcal{A}$ in the following sense.

Theorem 9. *Consider a domain in which all statements are of the form* $\underline{\texttt{action}}$ A $\underline{\texttt{causes}}$ L $\underline{\texttt{if}}$ C*, then any possible scenario (in the sense of* $\mathcal{D}$*) corresponds to a sequence of state transitions (in the sense of* $\mathcal{A}$*) and vice versa.*

2.3 Query Answering

In $\mathcal{D}$, we do not ask what definitely holds after a sequence of actions in a domain, but instead what *normally* holds. We next introduce the *normality* version of the query language $\mathcal{P}$ [2].

Definition 10. *Let* Σ *be a* $\mathcal{D}$ *action domain description,* α *be an action sequence and* L *be a fluent literal. A* query *is of the form* $\underline{\texttt{normally}}$ L $\underline{\texttt{after}}$ α*. We say that* Σ entails *the query and write* $\Sigma \mathrel{|\!\!\approx} \underline{\texttt{normally}}$ L $\underline{\texttt{after}}$ α *if and only if* $L \in S(\alpha)$ *for all possible scenarios* S *of* Σ*.*

When solving projection problems for a domain Σ, we specify an initial state and then ask what is true after a sequence of actions has been performed.

Example 2 (Continued). Possible queries about the swipe card domain $\Sigma_{\square}$ are "does the agent initially have a swipe card?" – $\underline{\texttt{normally}}$ $\mathsf{HasCard}$ $\underline{\texttt{after}}$ ε, or "is the door open after swiping the card and pushing the door?" – $\underline{\texttt{normally}}$ Open $\underline{\texttt{after}}$ $\mathsf{SwipePush}$. Both queries are entailed by $\Sigma_{\square}$ since they hold in the only possible scenario S_1.

3 Implementation

We now present the implementation of action language $\mathcal{D}$. It is based on logic programming, more specifically the answer set semantics, which has also been conceived by Vladimir Lifschitz together with his colleague Michael Gelfond [9]. The translation we define below transforms $\mathcal{D}$ action domain specifications into extended logic programs. These rules will contain first-order variables, thereby representing the set of the rule's well-sorted ground instances. Before delving into the technical details, we recall the necessary notions.

Definition 11. *Let* $\mathfrak{P}$ *be a propositional signature. An* extended logic program rule *is of the form*

$$L_0 \leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_{m+n} \tag{1}$$

[5] Language $\mathcal{C}$ deviates from this principle in that it assumes explicit inertia statements to be included in a domain specification [3].

where $L_0, L_1, \ldots, L_{m+n}$ *are literals over* $\mathfrak{P}$ *and* $m, n \geq 0$. *A program rule is* definite *iff* $n = 0$. *For a set* P *of definite rules and a set* M *of literals, define*

$$\Gamma_P(M) \stackrel{\text{def}}{=} \{L_0 \mid L_0 \leftarrow L_1, \ldots, L_m \in P, \{L_1, \ldots, L_m\} \subseteq M\}$$

For a set P *of extended logic program rules and a set* M *of literals, define the* Gelfond-Lifschitz reduct P^M *as the logic program obtained from* P *by*

1. *eliminating each rule containing an expression not* L *with* $L \in M$, *and*
2. *deleting all expressions not* L *from the remaining clauses.*

M *is an* answer set for P *iff* M *is consistent and the least fixed point of* Γ_{P^M}.

Note that this definition of answer sets requires consistency, while [9] admitted the set of all literals as an answer set for inconsistent programs. There will be no difference for query entailment, since this depends on containment in *all* answer sets. Additionally, in practice we are naturally only interested in possible *consistent* belief states for an agent.

For specifying the extended logic program P_Σ with variables for a domain Σ, we assume a sorted, first-order logical language Π. To the sorts FLUENT and ACTION used by a $\mathcal{D}$ domain, we add the sort TIME for time-points along with the predicates $\mathit{Holds}(f, t)$ (saying that fluent f is true at time-point t) and $\mathit{Poss}(a, s, t)$ (saying that it is possible to execute an action from time-point s to t). The notation $(\neg)F[\tau]$ for a fluent F and term τ : TIME abbreviates $(\neg)\mathit{Holds}(F, \tau)$. For a set $C = \{L_1, \ldots, L_m\}$ of fluent literals, $C[\tau]$ denotes the rule body $L_1[\tau], \ldots, L_m[\tau]$. We use a term-based encoding of action sequences in the logic program: for an action sequence α, its *ending time-point* τ_α is defined inductively by $\tau_\varepsilon \stackrel{\text{def}}{=} \varepsilon$ and $\tau_{\alpha A} \stackrel{\text{def}}{=} \tau_\alpha \cdot A$.

The answer set program defined below implements the principle of universal causation by reifying possible causes. In accord with $\mathcal{D}$'s semantics (cf. Definition 7), there is a pair of predicates for each type of effect: $\mathit{DirT}(f, a, s, t)$ and $\mathit{DirF}(f, a, s, t)$, e.g., express that f is a positive (negative) direct effect of action a from s to t. $\mathit{Dir}(L, a, s, t)$ for a fluent literal L then abbreviates $\mathit{DirF}(F, a, s, t)$ if $L = \neg F$ for a fluent F and $\mathit{DirT}(L, a, s, t)$ otherwise. Predicates $\mathit{Frame}(L, s, t)$, $\mathit{Ind}(L, s, t)$ and $\mathit{Def}(L, s, t)$ are used for the other causes.

The effect rules below now simply say that a fluent holds (or does not hold) after an action was possible leading to the time-point if there was a cause for the fluent to have its respective truth value. These causes, in turn, are derived from additional rules: for persistence, two special rules expressing what persistence means; the rules for direct effects are created from $\mathcal{D}$ direct effect statements and implement an explanation closure assumption; default effects are derived by rules using negation as failure. Disjunction ";" in rule bodies is syntactic sugar for multiplying out its arguments into multiple rules with the same head.

Definition 12. *Let* Υ *be a set of statements and* f : FLUENT, a : ACTION, s, t : TIME *be fresh variables and assume w.l.o.g.* $C = \{C_1, \ldots, C_n\}$ *for sets of fluent literals. The extended logic program* P_Υ *contains the following.*

For each `possible` A `if` $C \in \Upsilon$*, the rules*

$$Poss(A, s, s \cdot A) \leftarrow C[s] \tag{2}$$
$$\neg Poss(A, s, s \cdot A) \leftarrow \overline{C_1}[s];\ \ldots;\ \overline{C_n}[s] \tag{3}$$

For each `action` A `causes` L `if` $C \in \Upsilon$*, the rules*

$$Dir(L, A, s, t) \leftarrow Poss(A, s, t),\ C[s] \tag{4}$$
$$\neg Dir(L, A, s, t) \leftarrow \overline{C_1}[s];\ \ldots;\ \overline{C_n}[s] \tag{5}$$

For each `effect` K `causes` L `if` $C \in \Upsilon$*, the rules*

$$Ind(L, s, t) \leftarrow Poss(a, s, t),\ \overline{K}[s],\ K[t],\ C[s] \tag{6}$$
$$\neg Ind(L, s, t) \leftarrow K[s];\ \overline{K}[t];\ \overline{C_1}[s];\ \ldots;\ \overline{C_n}[s] \tag{7}$$

For each `normally` L `if` $C \in \Upsilon$*, the rules*

$$L[\varepsilon] \leftarrow C[\varepsilon],\ not\ \overline{L}[\varepsilon] \tag{8}$$
$$Def(L, s, t) \leftarrow Poss(a, s, t),\ C[t],\ (\overline{C_1}[s];\ \ldots;\ \overline{C_n}[s];\ L[s]),\ not\ \overline{L}[t] \tag{9}$$
$$\neg Def(L, s, t) \leftarrow \overline{C_1}[t];\ \ldots;\ \overline{C_n}[t];\ \overline{L}[t];\ (C[s],\ \overline{L}[s]) \tag{10}$$

Finally, the rules

$$\begin{aligned}
FrameT(f, s, t) \leftarrow{} & Poss(a, s, t),\ Holds(f, s), \\
& \neg DirF(f, a, s, t),\ \neg IndF(f, s, t),\ \neg DefF(f, s, t) && (11) \\
FrameF(f, s, t) \leftarrow{} & Poss(a, s, t),\ \neg Holds(f, s), \\
& \neg DirT(f, a, s, t),\ \neg IndT(f, s, t),\ \neg DefT(f, s, t) && (12) \\
Holds(f, t) \leftarrow{} & FrameT(f, s, t);\ DirT(f, a, s, t); \\
& IndT(f, s, t);\ DefT(f, s, t) && (13) \\
\neg Holds(f, t) \leftarrow{} & FrameF(f, s, t);\ DirF(f, a, s, t); \\
& IndF(f, s, t);\ DefF(f, s, t) && (14)
\end{aligned}$$

along with facts $\neg Dir(L, A, s, t)$ *for each* A : ACTION *and fluent literal* L *without direct effect statement; and facts* $\neg Ind(L, s, t)$ *and* $\neg Def(L, s, t)$ *for all fluent literals* L *that do not appear as consequent of an indirect effect statement or default statement, respectively.*

As a last but one step, we define how to transform a set Ω of $\mathcal{D}$ axioms into a set P_Ω of ground facts: initial state axioms just become ground *Holds* literals in the initial time-point ε.

Definition 13. *Let* Ω *be a set of* $\mathcal{D}$ *axioms.* $P_\Omega \stackrel{\text{def}}{=} \{L[\varepsilon] \mid$ `initially` $L \in \Omega\}$.

Example 2 (Continued). For the swipe card domain $\Upsilon_{\Box}$, the translation yields

$$\begin{aligned}
\mathit{Poss}(\mathsf{Swipe}, s, s \cdot \mathsf{Swipe}) &\leftarrow \mathit{Holds}(\mathsf{HasCard}, s) \\
\neg\mathit{Poss}(\mathsf{Swipe}, s, s \cdot \mathsf{Swipe}) &\leftarrow \neg\mathit{Holds}(\mathsf{HasCard}, s) \\
\mathit{Poss}(\mathsf{Push}, s, s \cdot \mathsf{Push}) & \\
\mathit{DirT}(\mathsf{Swiped}, \mathsf{Swipe}, s, t) &\leftarrow \mathit{Poss}(\mathsf{Swipe}, s, t) \\
\mathit{DirT}(\mathsf{Open}, \mathsf{Push}, s, t) &\leftarrow \mathit{Poss}(\mathsf{Push}, s, t), \\
&\qquad \neg\mathit{Holds}(\mathsf{Locked}, s),\ \neg\mathit{Holds}(\mathsf{Jammed}, s) \\
\mathit{DirT}(\mathsf{Jammed}, \mathsf{Push}, s, t) &\leftarrow \mathit{Poss}(\mathsf{Push}, s, t),\ \mathit{Holds}(\mathsf{Locked}, s) \\
\mathit{Holds}(\mathsf{HasCard}, \varepsilon) &\leftarrow \mathit{not}\ \neg\mathit{Holds}(\mathsf{HasCard}, \varepsilon) \\
\mathit{DefT}(\mathsf{HasCard}, s, t) &\leftarrow \mathit{Poss}(a, s, t), \\
&\qquad \mathit{Holds}(\mathsf{HasCard}, s),\ \mathit{not}\ \neg\mathit{Holds}(\mathsf{HasCard}, t) \\
\neg\mathit{Holds}(\mathsf{Locked}, \varepsilon) &\leftarrow \mathit{Holds}(\mathsf{Swiped}, \varepsilon),\ \mathit{not}\ \mathit{Holds}(\mathsf{Locked}, \varepsilon) \\
\mathit{DefF}(\mathsf{Locked}, s, t) &\leftarrow \mathit{Poss}(a, s, t), \\
&\qquad \neg\mathit{Holds}(\mathsf{Swiped}, s),\ \mathit{not}\ \mathit{Holds}(\mathsf{Locked}, t) \\
\mathit{DefF}(\mathsf{Locked}, s, t) &\leftarrow \mathit{Poss}(a, s, t), \\
&\qquad \neg\mathit{Holds}(\mathsf{Locked}, s),\ \mathit{not}\ \mathit{Holds}(\mathsf{Locked}, t)
\end{aligned}$$

along with $\neg\mathit{Dir}(L, A, s, t)$ for all fluent literals L and actions A without direct effect statement, $\neg\mathit{Ind}(L, s, t)$ for all fluent literals L and (11–14). The initial state axioms $\Omega_{\Box}$ become $\{\mathit{Holds}(\mathsf{Locked}, \varepsilon), \neg\mathit{Holds}(\mathsf{Jammed}, \varepsilon), \neg\mathit{Holds}(\mathsf{Swiped}, \varepsilon)\}$.

To complete our translation, we have to make sure the resulting program admits only groundings that are well-sorted with respect to the first-order language Π. Consequently, we define a fresh predicate name Q_σ for each sort σ of Π; we add a set of ASP rules that define the domain of each Q_σ; lastly, for all (first-order) variables $x : \sigma$ occurring in rules, we add the atom $Q_\sigma(x)$ to the rule body. So for a $\mathcal{D}$ action domain specification $\Sigma = \Upsilon \cup \Omega$, its ASP translation P_Σ is simply $P_\Upsilon \cup P_\Omega$ along with the rules for the sort domains. Since the signature Π contains function symbols of positive arity (e.g. in terms of the form $\tau_\alpha \cdot A$), the well-sorted grounding of P_Σ is obviously infinite. In the actual implementation, we introduce a finite horizon, that is, a maximal term depth to which the program is grounded.

The most remarkable property of the translation we presented above is that Υ (general workings of the domain) and Ω (a specific starting time-point for an agent in the domain) can be translated completely independently. This has the advantage that for any given domain, we need to compile the general domain knowledge only when it changes, which should arguably happen less often. In particular, the rules in P_Υ make no mention of actual states and can therefore easily be extended by any P_Ω. Information about actual domain instances (which arguably changes more often during online control of an agent) can be translated to ASP in linear time, which is immediate from Definition 13.

When posing a query to a $\mathcal{D}$ action domain, it is straightforward to reduce sceptical query entailment to answer set existence: we add an integrity constraint that forbids L being true at the ending time point τ_α of the action sequence.

Definition 14. *Let $\zeta = \underline{\texttt{normally}}\ L\ \underline{\texttt{after}}\ \alpha$ be a query. Its corresponding extended logic program is $P_\zeta \overset{\text{def}}{=} \{Q \leftarrow L[\tau_\alpha], \mathit{not}\ Q\}$ for a fresh predicate Q.*

Now for a query $\zeta = \underline{\texttt{normally}}\ L\ \underline{\texttt{after}}\ \alpha$, the literal $L[\tau_\alpha]$ is contained in every answer set of P iff $P \cup P_\zeta$ admits no answer set.

3.1 Proof of Correctness

In this section, we take the propositional signature $\mathfrak{P}$ to contain all ground instances of predicates from the signature Π of the previous section, hence P_Σ represents its ground instantiation. The proof now establishes a one-to-one correspondence between possible scenarios of Σ and answer sets of P_Σ.

To develop the correspondence, we first define how to obtain a scenario from a given set of literals. For a set M of literals over the domain signature of a domain specification Σ, we define a scenario $S_M \overset{\text{def}}{=} \{(\alpha, L) \mid L[\tau_\alpha] \in M, \alpha \in \text{ACTION}^*\}$. The scenario and M now agree on all fluent literals at all time-points (i.e., after all action sequences). Our first formal result states that the one-step consequence operators for domain Σ and program P_Σ agree likewise during their iterations, adjusted for a small difference/lag due to action applicability. For an operator Γ on sets, define $\Gamma^0 \overset{\text{def}}{=} \emptyset$, for $i \geq 0$ set $\Gamma^{i+1} \overset{\text{def}}{=} \Gamma(\Gamma^i)$ and $\Gamma^\infty \overset{\text{def}}{=} \bigcup_{i=0}^{\infty} \Gamma^i$.

Theorem 15. *Let $\Sigma = \Upsilon \cup \Omega$ be a domain, M be a set of literals, α be an action sequence and L be a fluent literal. For all $i \geq 0$,*

1. *$L[\tau_\alpha] \in \Gamma^i_{P^M_\Sigma}$ implies $(\alpha, L) \in \Gamma^i_{\Sigma, S_M}$, and*
2. *$(\alpha, L) \in \Gamma^i_{\Sigma, S_M}$ implies there is a $j \geq i$ with $L[\tau_\alpha] \in \Gamma^j_{P^M_\Sigma}$.*

As an immediate corollary, we obtain that the least fixed points of the two operators coincide with respect to literals after all action sequences.

Corollary 16. *Let Σ be a domain, M be a set of literals, α be an action sequence and L be a fluent literal. Then $(\alpha, L) \in \Gamma^\infty_{\Sigma, S_M}$ iff $L[\tau_\alpha] \in \Gamma^\infty_{P^M_\Sigma}$.*

Our main result now states that computing the answer sets of the logic program P_Σ as a matter of fact computes the possible scenarios of the action domain specification Σ.

Theorem 17. *Let Σ be a $\mathcal{D}$ action domain specification.*

1. *For each answer set M for P_Σ, scenario S_M is possible for Σ.*
2. *For each possible scenario S for Σ, there exists an answer set M for P_Σ such that $S_M = S$.*

Proof. *1. Let α be an action sequence and L be a ground literal. We have*

$$\begin{aligned} & (\alpha, L) \in \Gamma^{\infty}_{\Sigma,S_M} & \\ \textit{iff } & L[\tau_\alpha] \in \Gamma^{\infty}_{P^M_\Sigma} & \textit{(Corollary 16)} \\ \textit{iff } & L[\tau_\alpha] \in M & \textit{(M is an answer set for } P_\Sigma\textit{)} \\ \textit{iff } & (\alpha, L) \in S_M & \textit{(Definition of } S_M\textit{)} \end{aligned}$$

This yields $\Gamma^{\infty}_{\Sigma,S_M} = S_M$ and hence S_M is a possible scenario for Σ.

2. Define $M^0 \overset{\text{def}}{=} \{L[\tau_\alpha] \mid (\alpha, L) \in S\}$ and $M^\infty \overset{\text{def}}{=} \Gamma^{\infty}_{P^{M^0}_\Sigma}$. We prove that M^∞ is an answer set for P_Σ, that is, $\Gamma^{\infty}_{P^{M^\infty}_\Sigma} = M^\infty = \Gamma^{\infty}_{P^{M^0}_\Sigma}$ by showing $P^{M^\infty}_\Sigma = P^{M^0}_\Sigma$. Observe that all default-negated literals in P_Σ are of the form "not $L[\tau]$" for some time-point τ, hence it suffices to show that M^0 and M^∞ agree on all literals of the form $L[\tau_\alpha]$ for an action sequence α:

$$\begin{aligned} & L[\tau_\alpha] \in M^0 & \\ \textit{iff } & (\alpha, L) \in S & (S = S_{M^0}) \\ \textit{iff } & (\alpha, L) \in \Gamma^{\infty}_{\Sigma,S} & \textit{(S is possible)} \\ \textit{iff } & L[\tau_\alpha] \in \Gamma^{\infty}_{P^{M^0}_\Sigma} & \textit{(Corollary 16)} \\ \textit{iff } & L[\tau_\alpha] \in M^\infty & \textit{(Definition of } M^\infty\textit{)} \end{aligned}$$

Additionally, this shows that $S_{M^\infty} = S$ and concludes the proof. □

The main result implies as an immediate corollary the correctness of our implementation of query answering in $\mathcal{D}$.

Corollary 18. *Let Σ be a $\mathcal{D}$ action domain specification, and consider a query $\zeta =$ `normally` L `after` α, then $\Sigma \mathrel{|\!\!\approx} \zeta$ if and only if $L[\tau_\alpha]$ is contained in each answer set of P_Σ.*

The one-to-one correspondence of possible scenarios and answer sets gives us important information about the type of nonmonotonic reasoning performed by $\mathcal{D}$: there is a similar one-to-one correspondence between answer sets of extended logic programs and extensions of Reiter's default logic [9]. Combining these two results, we can see that the action language presented here indeed allows for Reiter-style default reasoning.

4 The Qualification Problem

In $\mathcal{D}$ as defined so far, we make the simplifying assumption that necessary preconditions of actions are fully known. This is unrealistic insofar as there are potentially infinitely many circumstances that can prevent the successful execution of an action, which cannot be foreseen let alone specified [7]. For practical reasoning, this constitutes at least two tasks: (1) assume away all of the unlikely circumstances that prevent action execution and (2) correctly predict action disqualification if an exceptional circumstance is known to arise.

Moreover, an action might not always downright fail, but only *fail to produce a certain outcome*, for which Vladimir Lifschitz and his colleagues coined the term *weak* qualification [10] (vs. *strong* qualification, which means that an action fails altogether).

To solve the strong qualification problem in $\mathcal{D}$, we introduce a new function symbol $\mathsf{Disqualified} : \textsc{action} \to \textsc{fluent}$ with the obvious intuitive meaning [11]. For each action A, we turn its precondition P_A into $P_A \cup \{\neg\mathsf{Disqualified}(A)\}$ – that is, to the known, definite preconditions we add another one that represents all additional conceivable preconditions. These are then assumed away by default through the statement $\underline{\texttt{normally}}\ \neg\mathsf{Disqualified}(A)$. An action A is then possible after α iff $\Sigma \mathrel{|\!\!\approx} \underline{\texttt{normally}}\ L\ \underline{\texttt{after}}\ \alpha$ for all $L \in P_A$. Particular causes for disqualification of A can now be expressed modularly by adding ramification rules $\underline{\texttt{effect}}\ K\ \underline{\texttt{causes}}\ \mathsf{Disqualified}(A)\ \underline{\texttt{if}}\ C$.

For weak qualification, we introduce a similar function symbol $\mathsf{Ab} : \textsc{fluent} \to \textsc{fluent}$. The term $\mathsf{Ab}(F)$ now says that fluent F is abnormally disqualified at a time-point. For each direct effect statement $\underline{\texttt{action}}\ A\ \underline{\texttt{causes}}\ (\neg)F\ \underline{\texttt{if}}\ C$ we add $\neg\mathsf{Ab}(F)$ to the effect's precondition C. This expresses that effect $(\neg)F$ only occurs if there was nothing abnormal about F at the starting time point. As before, abnormality is assumed away by default using the statements $\underline{\texttt{normally}}\ \neg\mathsf{Ab}(F)$ for all fluents F.

Notice that $\mathcal{D}$'s solution to the qualification problem is formulated entirely within the language itself and requires no language extension or external machinery.

5 Discussion

We introduced the action language $\mathcal{D}$ for default reasoning about actions. The language solves the frame problem in the presence of defaults as well as the ramification and qualification problems. $\mathcal{D}$ is efficiently implemented, which makes it immediately applicable for practical problems.

Our work can be seen as a restriction of [12] where the restriction is solely for computational benefits: in a domain with n fluents, there are potentially $O(2^n)$ knowledge states, all of which we would have to inspect for query answering. What we trade off here, however, is expressiveness: the version of $\mathcal{D}$ presented in this paper is essentially "disjunction-free." For example, from the statements $\underline{\texttt{action}}\ A\ \underline{\texttt{causes}}\ G\ \underline{\texttt{if}}\ \{F\}$ and $\underline{\texttt{action}}\ A\ \underline{\texttt{causes}}\ G\ \underline{\texttt{if}}\ \{\neg F\}$, we cannot conclude $\underline{\texttt{normally}}\ G\ \underline{\texttt{after}}\ A$.[6] This is however common for the implementation of action languages [1]. In particular, $\mathcal{A}$ and $\mathcal{E}$ only come with translations to logic programming that are sound but incomplete. In our case of a nonmonotonic formalism, this would not work since incompleteness is bound to lead to unsoundness (imagine a default being wrongly applicable because a justification could not be derived due to incompleteness).

[6] In $\mathcal{A}$, such a pair of statements would not even be admissible for translation [1].

Generally speaking, surprisingly few approaches exist that deal with default reasoning about actions. The Discrete Event Calculus [13] offers simple default reasoning about time using circumscription, which however allows for default conclusions with circular justifications. Action language $\mathcal{C}+$ [5] provides the default statements we have seen in the introduction, which however have an underlying intuition that is different from ours. Additionally, $\mathcal{C}+$ also allows the circularly justified conclusions we have seen in the introduction. [14] use a modal variant of the Situation Calculus to define a semantics for reasoning about actions in the presence of defaults. They however "forget" default conclusions and reapply defaults after each action without keeping track of extensions. Also, there is no mention of an implementation. [15] offer an argumentation-based semantics for what they call "knowledge qualification." There, each piece of knowledge about a domain is encoded as an argument in favor of some conclusion. Preferences between different kinds of arguments determine what is believed about the domain. These preferences are declared by the axiomatiser and are themselves arguments, which makes the formalism quite complex and delegates the specification of the semantics to the user. An implementation is not mentioned in the paper. [16] extend modal logics with preferences in order to incorporate defeasible inferences into modal-based formalisms for reasoning about actions. However, they do not provide a solution to the frame problem in this framework nor an implementation.

There are many implemented systems that employ answer set programming for reasoning about actions – [1, 6, 17–19] to name only a few. However, none of these systems aspire to combine temporal reasoning with default reasoning. They use the nonmonotonicity of ASP to implement the explanation closure assumption, to solve the frame problem or compute circumscription.

Acknowledgements

This paper is based entirely on Vladimir Lifschitz's groundbreaking research in knowledge representation, reasoning about actions and logic programming. None of what we described would have been possible without his seminal work on action languages and the foundations for answer set programming that he laid, for which we thank him profoundly! We also thank our second reader Fangkai Yang for the for helpful comments on an earlier version of this paper.

This research was supported under Australian Research Council's (ARC) *Discovery Projects* funding scheme (project number DP 120102144) The second author is the recipient of an ARC Future Fellowship (project number FT 0991348). He is also affiliated with the University of Western Sydney.

References

1. Gelfond, M., Lifschitz, V.: Representing Action and Change by Logic Programs. Journal of Logic Programming **17**(2/3&4) (1993) 301–321
2. Gelfond, M., Lifschitz, V.: Action Languages. Electronic Transactions on Artificial Intelligence **3** (1998)
3. Giunchiglia, E., Lifschitz, V.: An Action Language Based on Causal Explanation: Preliminary Report. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Menlo Park, CA, USA, American Association for Artificial Intelligence (1998) 623–630
4. Kakas, A., Miller, R.: A Simple Declarative Language for Describing Narratives with Actions. Journal of Logic Programming **31**(1–3) (1997) 157–200 Reasoning about Actions and Change.
5. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic Causal Theories. Artificial Intelligence **153**(1-2) (2004) 49–104
6. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. ACM Transactions on Computational Logic **5** (April 2004) 206–263
7. McCarthy, J.: Epistemological Problems of Artificial Intelligence. In: Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77). (1977) 1038–1044
8. Reiter, R.: A Logic for Default Reasoning. Artificial Intelligence **13** (1980) 81–132
9. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
10. Gelfond, M., Lifschitz, V., Rabinov, A.: What Are the Limitations of the Situation Calculus? In Boyer, R.S., Pase, W., eds.: Automated Reasoning. Volume 1 of Automated Reasoning Series. Springer Netherlands (1991) 167–179
11. Thielscher, M.: Causality and the Qualification Problem. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR), Cambridge, MA (November 1996) 51–62
12. Baumann, R., Brewka, G., Strass, H., Thielscher, M., Zaslawski, V.: State Defaults and Ramifications in the Unifying Action Calculus. In: Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning, Toronto, Canada (May 2010) 435–444
13. Mueller, E.: Commonsense Reasoning. Morgan Kaufmann (2006)
14. Lakemeyer, G., Levesque, H.: A Semantical Account of Progression in the Presence of Defaults. In: Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09). (2009) 842–847
15. Michael, L., Kakas, A.: A Unified Argumentation-Based Framework for Knowledge Qualification. In Davis, E., Doherty, P., Erdem, E., eds.: Proceedings Commonsense, Stanford, CA (March 2011)
16. Britz, K., Meyer, T., Varzinczak, I.: Preferential Reasoning for Modal Logics. Electronic Notes in Theoretical Computer Science **278** (2011) 55–69
17. Kim, T.W., Lee, J., Palla, R.: Circumscriptive Event Calculus as Answer Set Programming. In: IJCAI. (July 2009) 823–829
18. Lee, J., Palla, R.: Situation Calculus as Answer Set Programming. In: Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10). (July 2010) 309–314
19. Casolary, M., Lee, J.: Representing the Language of the Causal Calculator in Answer Set Programming. In: Proceedings of the Twenty-Seventh International Conference on Logic Programming (ICLP-11). (July 2011)