The draculasp System: Default Reasoning about Actions and Change Using Logic and Answer Set Programming

Hannes Strass
Institute of Computer Science
University of Leipzig
strass@informatik.uni-leipzig.de

Abstract

We present an implementation for nonmonotonic reasoning about action domains. Its name draculasp stands for "**d**efault **r**easoning about **a**ctions and **c**hange **u**sing **l**ogic and **a**nswer **s**et **p**rogramming." The system compiles action domain specifications from a logic-based input language into answer set programs and invokes an ASP solver to answer queries about the domain. Intelligent agents can use draculasp to predict what normally holds in incompletely known action domains.

Introduction

Intelligent agents will virtually never have complete knowledge about the domain they are situated in. Still, they constantly have to decide on which course of action to take. This problem of reasoning about incompletely known domains can be tackled by making justified assumptions about the world, which are retracted should the agent gain knowledge to the contrary. The theoretical foundations of this solution were introduced by (Baumann et al. 2010), that embed action theories into default logic (Reiter 1980).

In this paper, we present a practical implementation of their approach. Of course, we cannot expect to be able to implement full-fledged reasoning in Reiter's default logic: after all, extension existence for closed normal first-order default theories is not even semi-decidable (Reiter 1980). A high computational complexity is retained even through restriction to propositional logic: sceptical reasoning is Π_2^{P}-complete for propositional normal default theories (Gottlob 1992). Thus, we will have to make some restricting assumptions on our input domains that make an implementation feasible in principle (that is, make the relevant reasoning problems decidable) and in practice (that is, allow for efficient implementations of reasoning procedures).

But even for a sufficiently expressive yet simple enough input language, implementing a reasoner from scratch is complicated and error-prone. Although it could be adapted and tuned for efficiency, its range of applicability would be rather limited. Fortunately, there exist efficient general-purpose reasoners for default theories of a particular form: answer set programming solvers.

(Marek and Truszczyński 1989) discovered a close connection between default logic and answer set programming: roughly, a normal logic program can be modularly translated into a default theory such that there is a one-to-one correspondence between the answer sets of the logic program and the extensions of the default theory. This connection will form the basis of our implementation. For finite domains and a finite time horizon, current ASP technology can then be used off the shelf to efficiently reason about action domains. In the last decade, answer set programming has made significant advancements in terms of efficiency, and current solvers can now treat programs with millions of variables despite the theoretical worst-case NP-hardness. Still, if the polynomial hierarchy does not collapse there remains an exponential gap between Π_2^{P} (sceptical reasoning in propositional normal default theories) and NP (answer set existence for propositional normal logic programs).

When aiming for an approach that compiles action domain specifications into logic programs and reduces reasoning about the domains to reasoning in these programs, we therefore have to expect a worst-case exponential blowup or at least a worst-case exponential runtime for these translations. This is acceptable if we compile only once to ask multiple queries afterwards. But in the case of an agent situated in a changing domain that constantly executes actions and makes new observations, it may be too costly to recompile its whole world knowledge after adding to it the observations associated to each action. Consequently, we are looking for a translation where the domain description is compiled once and additional information about action executions and observations can be added in a *modular* way.[1]

Also, since we are dealing with a non-monotonic formalism, another issue is getting into the way of a straightforward implementation: for monotonic semantics, there exists the possibility of an efficient implementation that is sound, but not complete. With non-monotonic semantics, this possibility does not exist in general: for any conclusion that follows semantically but is not made by the system due to incompleteness, there may exist a default that relies on absence of this conclusion and makes an unsound default inference.

The system we present in this paper is motivated by these reflections and rests on an input language for which a sound and complete modular translation from the resulting action default theories into answer set programs exists.

[1] By modular, we mean the following: for two languages L_1, L_2, a translation function $f : 2^{L_1} \to 2^{L_2}$ is *modular* iff for all $A \subseteq L_1$, the function satisfies $f(A) = \bigcup_{a \in A} f(\{a\})$, that is, sets of language elements can be translated element-wise.

Background: Default Reasoning about Actions

The approach of (Baumann et al. 2010) embeds action theories formulated in the language of the unifying action calculus (UAC) (Thielscher 2011) into default logic (Reiter 1980). The action theories are viewed as incomplete knowledge bases that are completed by defaults. It takes as input a description of an action domain containing (1) a domain signature, that defines the vocabulary of the domain; (2) a description of the preconditions and effects of actions; (3) a set of *state defaults* $\Phi \rightsquigarrow \psi$, statements that specify conditions (in form of a fluent formula Φ) under which a fluent literal ψ normally holds in the domain. The state defaults from the domain description are translated into Reiter defaults, where the special predicates $\mathit{DefT}(f, s, t)$ and $\mathit{DefF}(f, s, t)$ are used to express that a fluent f becomes normally true (false) from s to t. In this paper we are chiefly concerned with the implementation of their approach, so we refer the interested reader to the original paper and only illustrate their approach with an example.

Example 1 (Swipe Card Domain)**.** The objective of this domain is to open an electronically Locked door using a swipe card. If the agent has a card (HasCard), it can Swipe the card to unlock the door; if the door is unlocked and not Jammed, it can be Pushed Open. Normally, the door is not jammed. The default theory of the swipe card domain is $(\Sigma^{sc}, \Delta^{sc})$ below. In Σ^{sc}, action preconditions are expressed by the axioms

$$\mathit{Poss}(\mathsf{Swipe}, s, t) \equiv (\mathit{Holds}(\mathsf{HasCard}, s) \wedge \mathit{Occurs}(\mathsf{Swipe}, s, t))$$
$$\mathit{Poss}(\mathsf{Push}, s, t) \equiv (\neg\mathit{Holds}(\mathsf{Locked}, s) \wedge \neg\mathit{Holds}(\mathsf{Jammed}, s) \wedge \mathit{Occurs}(\mathsf{Push}, s, t))$$

The abstract predicate $\mathit{Occurs}(a, s, t)$ denotes occurrence of action a from s to t and enables us to use different time structures. The effect axiom below now says that all that holds at the resulting time point t of the action must have a cause among direct effects, defaults or persistence.

$$\mathit{Poss}(a, s, t) \supset$$
$$\mathit{Holds}(f, t) \equiv (\mathit{FrameT}(f, s, t) \vee \mathit{DirT}(f, a, s, t) \vee \mathit{DefT}(f, s, t)) \wedge$$
$$\neg\mathit{Holds}(f, t) \equiv (\mathit{FrameF}(f, s, t) \vee \mathit{DirF}(f, a, s, t) \vee \mathit{DefF}(f, s, t))$$

The formulas expressing direct effects are $\mathit{DirT}(\mathsf{Open}, \mathsf{Push}, s, t)$, $\mathit{DirF}(\mathsf{Locked}, \mathsf{Swipe}, s, t)$ and $\neg\mathit{DirT}(F, A, s, t)$ and $\neg\mathit{DirF}(F, A, s, t)$ for all other fluents F and actions A. The meaning of persistence itself is catered for by the axioms

$$\mathit{FrameT}(f, s, t) \equiv (\mathit{Holds}(f, s) \wedge \mathit{Holds}(f, t))$$
$$\mathit{FrameF}(f, s, t) \equiv (\neg\mathit{Holds}(f, s) \wedge \neg\mathit{Holds}(f, t))$$

and the default closure axioms $\mathit{Holds}(\mathsf{Jammed}, s) \supset \neg\mathit{DefF}(\mathsf{Jammed}, s, t)$ and $\neg\mathit{Def}(F, s, t)$ for all other fluents F. Together, they implement a solution to the notorious frame problem (McCarthy 1977) in the presence of defaults. The Reiter defaults Δ^{sc} obtained from $\top \rightsquigarrow \neg\mathsf{Jammed}$ are given by

$$\mathit{Init}(t) : \neg\mathit{Holds}(\mathsf{Jammed}, t) / \neg\mathit{Holds}(\mathsf{Jammed}, t)$$
$$\neg\mathit{Holds}(\mathsf{Jammed}, s) : \mathit{DefF}(\mathsf{Jammed}, s, t) / \mathit{DefF}(\mathsf{Jammed}, s, t).$$

The generality of the approach allows it to deal with different time structures using one and the same axiomatisation technique. We will later see how this is instantiated by concrete time structures like situations or natural numbers.

From Action Domains to ASPs

For the purpose of this paper, we treat the formulas occurring in the default theories of (Baumann et al. 2010) as quantifier-free. This is possible because we will be dealing with a class of domains whose default theories do not use existential quantification. To simplify the definitions in the sequel, we also assume that all predicate macros have been replaced by respective predicate symbols and the formulas in domain axiomatisations are in conjunctive normal form.

In addition to the general domain information in form of a default theory (Σ, Δ), the user can provide information about a specific instance of the domain. An instance is first characterised by a time structure, situations or linear time. Technically, this extends the signature of the domain by the appropriate function symbols into sort TIME. Second, the instance information contains additional world knowledge whose form depends on the chosen notion of time. For situations, the user can provide a characterisation of the initial situation via a conjunction of literals $(\neg)\mathit{Holds}(F, S_0)$. For linear time, they can specify a narrative consisting of action occurrence statements and *Holds* literals. This user-specified information is easily transformed into a set of formulas Σ_{inst}, which is added to the theory about the domain resulting in the default theory $(\Sigma \cup \Sigma_{\mathit{inst}}, \Delta)$.

Example 1 (Continued)**.** For situations as time structure and an initial time point where the agent has a card, we get $\Sigma^{sc}_{\mathit{inst}}$ containing the three axioms $\mathit{Init}(S_0)$, $\mathit{Holds}(\mathsf{HasCard}, S_0)$ and $\mathit{Occurs}(a, s, t) \equiv t = \mathit{Do}(a, s)$.

Although the semantics defined in (Baumann et al. 2010) can in principle deal with infinite domains, the same cannot be expected in general from an implementation. In particular, answer set programming systems are based on propositional logic, which precludes the use of function symbols of positive arity in any way that leads to infinite domains. So for the implementation, we restrict our attention to domains with a finite number of objects. This *domain closure* is expressed in the translation as follows. Current answer set solvers allow to specify normal logic programs *with variables*. In a preprocessing step, the *grounder* of an ASP system then replaces rules with variables by their ground instances (provided the rules are safe). Any formula with (implicitly universally quantified) free variables is thereby replaced by a conjunction of its ground instances, which is semantically equivalent for finite domains. Due to the modularity of our translation, this also means that the general characterisation of the domain and information about a specific instance can be compiled independently. For the reasoning agent this has the benefit that the general domain information only has to be compiled once, while subsequent observations and queries can be translated separately.

Now let a finite action domain be given as a quantifier-free default theory (Σ, Δ) where Σ is a set of clauses. For each clause $L_1 \vee \ldots \vee L_m \in \Sigma$ (possibly containing free variables), we write the extended logic program rules

$$L_i \leftarrow \neg L_1, \ldots, \neg L_{i-1}, \neg L_{i+1}, \ldots, \neg L_m \quad (i = 1, \ldots, m)$$

They express that "L_i must be true if all the other L_j are false." It can be proved that the ground instances of these

rules P_Σ are sound with respect to logical consequences and preserve the clause's potential for unit resolution.

In a similar way, we translate the defaults in Δ to extended logic program rules. For example, the two defaults that arise from $\top \rightsquigarrow \neg\mathsf{Jammed}$ are turned into

$\neg \mathit{Holds}(\mathsf{Jammed}, t) \leftarrow \mathit{Init}(t),\ \mathit{not}\ \mathit{Holds}(\mathsf{Jammed}, t)$
$\mathit{DefF}(\mathsf{Jammed}, s, t) \leftarrow \neg\mathit{Holds}(\mathsf{Jammed}, s),\ \mathit{not}\ \neg\mathit{DefF}(\mathsf{Jammed}, s, t)$

where *not* is negation as failure. As is usual in answer set programming, we then replace negated predicates $\neg Q$ by a new predicate symbol Q' standing for the (classical) negation of Q (Gelfond and Lifschitz 1991). The resulting rules now form the corresponding answer set program $P_{\Sigma,\Delta}$ of the action domain given by default theory (Σ, Δ).

It remains to make sure that the program can be grounded in a well-sorted way. Recall that the user specifies an action domain using a signature that may contain function symbols of positive arity. Since this may however immediately lead to an infinite ground instantiation, we restrict the term depth to a certain natural number *#maxDepth* specified along with the domain. That way, it is much easier for the user to write elaboration-tolerant action domain specifications, while the theory works as before. In the implementation, the creation of the domains of the sorts is dynamically done by the solver via meta-programming, using additional ASP rules. For example, the sort SIT for situations is defined by the rules

$\mathrm{SIT}(0, S_0)$
$\mathrm{SIT}(i+1, \mathit{Do}(a, s)) \leftarrow \mathrm{ACTION}(i, a), \mathrm{SIT}(i, s), i < \mathit{\#maxDepth}$
$\mathrm{SIT}(s) \leftarrow \mathrm{SIT}(i, s)$

This results in the program P_{sorts} containing the rules for all sorts of the underlying Situation Calculus signature. By means of these newly introduced predicates S, it is straightforward to ensure safety of the rules in P_Σ: for any unsafe variable x of sort S in a rule, we add an atom $\mathrm{S}(x)$ to the rule body. This way of creating the sort domains easily allows us to reason about situations with a finite horizon. In this case, the value of *#maxDepth* limits the program's lookahead into the situation tree, which can be dynamically increased by simply adjusting the depth. The ASP solver then grounds the domain and thereby produces the well-sorted grounding with a built-in unique-names assumption.

Correctness for a class of domains Although the translation presented here is in principle defined for any quantifier-free default theory, it does not always preserve logical consequences. Fortunately the translation is sound and complete for a certain class of domains which we call *admissible*. For an action domain, this requires that all state defaults are prerequisite-free and all action preconditions are disjunction-free. The instance information Σ_{inst} is admissible if (1) there is a unique ground atom $\mathit{Init}(\tau_0) \in \Sigma_{\mathit{inst}}$ and (2) all state formulas in Σ_{inst} are ground literals in τ_0. (E.g., the swipe card domain from Example 1 is admissible.)

The structure of the ground clauses can be used to argue that the Horn translation preserves their meaning with respect to entailment of *Holds* literals. Due to the restricted form of the defaults $\mathit{Init}(t) : \psi[t]/\psi[t]$ and $\psi[s] : \mathit{Def}(\psi, s, t)/\mathit{Def}(\psi, s, t)$, this correspondence can be generalised to extensions of consistent action default theories. For inconsistent Σ, the extensions of (Σ, Δ) and answer sets of $P_{\Sigma,\Delta}$ differ: while the original default theory has a single inconsistent extension, its translation has no answer set due to the integrity constraints. In any case, an inconsistent domain axiomatisation must be considered erroneous and makes determining consequences unnecessary.

The Implemented System draculasp

The name draculasp alludes to the system's usage for nonmonotonic reasoning about action domains, the semantics of the input language being defined in terms of (default) logic and the actual reasoning being done via ASP.

The draculasp system is written in $\mathrm{ECL}^i\mathrm{PS}^e$ Prolog[2] and implements the translation of the previous section. It takes as input an action domain specification and instance information for that domain and transforms it into an answer set program P with variables. This program can then be queried by invoking an external solver. Technically, a query φ is added to the program P as an integrity constraint, that is, we create the new program $P' \stackrel{\text{def}}{=} P \cup \{\bot \leftarrow \varphi\}$. Now P' admits no answer set if and only if the query φ is contained in each answer set of P. The system and some example domains can be downloaded from the author's web page.[3]

Usage Information about the vocabulary and other general domain properties is stored in text files with a special syntax, *action domain specifications*. Below we can see the representation of the swipe card domain from Example 1.

```
sort action:  swipe, push.
sort fluent:  hasCard, locked, open, jammed.
precondition swipe:  hasCard.
precondition push:  and(not(locked),
not(jammed)).
effects swipe:  not(locked).
effects push:  open.
normally not(jammed).
```

The first two statements define the domain signature. Next, action preconditions, action effects and state defaults are specified, where action effects are grouped by actions. Since the translation implemented by draculasp is modular, this part of the description can be processed in isolation, leading to a program P_{sc} that contains general domain knowledge.

Information about a specific domain configuration is given in an *action domain instance* file. Each such file refers to an action domain specification, defines a time structure and additionally states an initial situation or a narrative (depending on the chosen time structure). Here is a branching-time instance of the swipe card domain:

```
instance of "swipecard.ads".
time structure:  situations.  term depth:  3.
initially hasCard.
```

The first line refers to the domain of which the file defines an instance. The next lines declare time structure and term depth, and the last line characterises the initial situation. Translating this action domain instance file yields an answer set program P^1_{inst}. Together with P_{sc} from above, it allows to reason about what normally holds in the swipe card domain. Below is another example, that uses a linear time structure:

[2] http://eclipseclp.org

[3] http://informatik.uni-leipzig.de/~strass/draculasp/

```
instance of "swipecard.ads".
time structure: linear time 0..3.  term depth: 2.
narrative:  holds(hasCard, 0),
     occurs(swipe, 0, 1), occurs(push, 1, 2).
```

The second line expresses that the TIME domain is the set $\{0, \ldots, 3\} \subseteq \mathbb{N}$ and defines the term depth. The block at the end specifies a narrative in which the initial time point is as in the branching time instance and the door is unlocked from 0 to 1 before being pushed from 1 to 2. Our system translates this action domain instance file into a program P^2_{inst}. Reusing the separately transformed P_{sc} from above, we obtain the answer set program $P_{sc} \cup P^2_{inst}$ about this linear-time instance of the swipe card domain, which can be used for logical default reasoning. We exemplify this in the following by writing the respective successful queries in typewriter font. So, for instance, in the domain above, the agent initially has a card, `holds(hasCard,0)` (hence can swipe it, `poss(swipe,0,1)`), and the door is not jammed, `-holds(jammed,0)`. This persists through swiping the card from `0` to `1`, whence `-holds(jammed,1)`. As an effect of swiping, we get `-holds(locked,1)`, which enables the agent to push the door, `poss(push,1,2)`. After pushing it, the door is indeed open, `holds(open,2)`.

Conclusion

We have presented the draculasp System for reasoning about actions in domains with state defaults. The system implements a compilation from action domain specifications into answer set programs with variables and then invokes an answer set solver to answer queries.

Using the theoretical relationship between default logic and answer set programming to implement one via the other is not new. (Junker and Konolige 1990) provided a translation from propositional default theories into normal logic programs[4] which has recently been rediscovered by (Chen et al. 2010). However, their translation is not modular – after adding new information about the domain, the whole theory has to be recompiled. In particular, the theory has to be recompiled for each query that is asked, since queries are modelled by defaults. More significantly, their approach only works for propositional default theories, which necessitates a first-ground-then-translate approach, which we dismissed based on performance considerations.

The causal calculator (CCALC)[5] was developed by Norman McCain as part of his PhD thesis and since then maintained by the Texas Action Group at the University of Austin. It implements the action language $\mathcal{C}+$ (Giunchiglia et al. 2004). Like draculasp, the causal calculator offers the specification of action domains and answers queries about these domains by translation into a logical language. Indeed, the functionality of CCALC was an inspiration for draculasp. Similarly, the system $\texttt{dlv}^{\mathcal{K}}$ implements the action language $\mathcal{K}$ (Eiter et al. 2000) on top of the `dlv` answer set solver (Eiter et al. 1997). However, the default semantics of $\mathcal{C}+$ and $\mathcal{K}$ have an underlying intuition that greatly differs from ours. We consider state defaults as saying that something normally holds, but may be exceptionally untrue, where this exception persists. $\mathcal{C}+$ and $\mathcal{K}$ regard default statements as causes on a par with all others, fluents that have a default truth value may become true (or false) by default without an obvious immediate cause. This view allows them to use defaults to model causes that are not known, not observable or too cumbersome to axiomatise.

[4] In fact, they translated default theories and theories of autoepistemic logic into truth maintenance systems, which can however equivalently be seen as normal logic programs under the stable model semantics (Reinfrank, Dressler, and Brewka 1989).

[5] `http://www.cs.utexas.edu/users/tag/cc/`

References

[Baumann et al. 2010] Baumann, R.; Brewka, G.; Strass, H.; Thielscher, M.; and Zaslawski, V. 2010. State Defaults and Ramifications in the Unifying Action Calculus. In *KR*, 435–444.

[Chen et al. 2010] Chen, Y.; Wan, H.; Zhang, Y.; and Zhou, Y. 2010. dl2asp: Implementing Default Logic via Answer Set Programming. *JELIA*, vol. 6341 of *LNCS*, 104–116. Springer.

[Eiter et al. 1997] Eiter, T.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1997. A deductive system for nonmonotonic reasoning. *LPNMR*, vol. 1265 of *LNCS*, 364–375. Dagstuhl Castle, Germany: Springer.

[Eiter et al. 2000] Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2000. Planning under Incomplete Knowledge. *CL*, vol. 1861 of *LNCS*, 807–821. Springer.

[Gelfond and Lifschitz 1991] Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Gen. Comp.* 9:365–385.

[Giunchiglia et al. 2004] Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic Causal Theories. *AIJ* 153(1-2):49–104.

[Gottlob 1992] Gottlob, G. 1992. Complexity Results for Nonmonotonic Logics. *J. Log. Comp.* 2(3):397–425.

[Junker and Konolige 1990] Junker, U., and Konolige, K. 1990. Computing the Extensions of Autoepistemic and Default Logics with a Truth Maintenance System. In *AAAI*, 278–283. AAAI Press / The MIT Press.

[Marek and Truszczyński 1989] Marek, V. W., and Truszczyński, M. 1989. Stable semantics for logic programs and default theories. In *NACLP*, 243–256. The MIT Press.

[McCarthy 1977] McCarthy, J. 1977. Epistemological Problems of Artificial Intelligence. In *IJCAI*, 1038–1044.

[Reinfrank, Dressler, and Brewka 1989] Reinfrank, M.; Dressler, O.; and Brewka, G. 1989. On the relation between truth maintenance and autoepistemic logic. In *IJCAI*, 1206–1212. Morgan Kaufmann Publishers Inc.

[Reiter 1980] Reiter, R. 1980. A Logic for Default Reasoning. *AIJ* 13:81–132.

[Reiter 2001] Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

[Thielscher 2011] Thielscher, M. 2011. A Unifying Action Calculus. *AIJ* 175(1):120–141.