On the Relative Expressiveness of ADL and Golog: The Last Piece in the Puzzle

Gabriele Röger and **Malte Helmert** and **Bernhard Nebel**
University of Freiburg, Germany
{roeger,helmert,nebel}@informatik.uni-freiburg.de

Abstract

Integrating agent programming languages and efficient action planning is a promising approach because it combines the expressive power of languages such as Golog with the possibility of searching for plans efficiently. In order to integrate a Golog interpreter with a planner, one has to understand, however, which part of the expressiveness of Golog can be captured by the planning language. Using Nebel's compilation framework, we identify a maximal fragment of *basic action theories*, the formalism Golog is based on, that is expressively equivalent to the ADL subset of PDDL. As we will show, almost all features that permit to specify incomplete information in basic action theories cannot be compiled to ADL.

Introduction

The integration of agent programming languages with efficient planning algorithms is a tantalizing idea. With such an integration, it becomes possible to specify the behavior of a system using a very rich specification language and still benefit from algorithmic advances in automated action planning.

Indeed, there are several recent papers that deal with the integration of Golog-style (Levesque et al. 1997) agent programming languages and PDDL planning (Fox and Long 2003) in various forms. Most, but not all, of these papers only deal with the most commonly considered ADL fragment of PDDL, also called PDDL level 1. As is customary in the planning literature, we will simply refer to this PDDL fragment as "ADL" throughout this paper, despite some differences to Pednault's (1989) original ADL formalism.

Claßen et al. (2007) discuss a compilation from PDDL planning domains to basic action theories, which form the semantic base of Golog. They show how the semantics of ADL can be understood as progression in the situation calculus. They also present a proof-of-concept integration of an efficient automated ADL planner (the FF system by Hoffmann and Nebel, 2001) into a Golog interpreter. However, their integration requires an ADL specification of the planning domain, and thus cannot be used for a wider class of basic action theories.

To make automated planning technology more generally useful in the Golog context, it is thus necessary to develop a compilation from basic action theories to ADL. In this direction, Baier, Fritz and McIlraith (2007) present a method for incorporating Golog-style procedural domain control knowledge into ADL planning instances. The control knowledge is compiled into the original ADL domain description to form a new ADL specification, which has the same transition semantics as the original, but only accepts action sequences that comply with the given control rules. However, their approach still requires that the domain description is specified in the (compared to basic action theories, less expressive) ADL formalism. Golog-style rules are only used for *restricting* the given transition semantics, not for defining them.

To overcome this limitation, one needs a general compilation from basic action theories to ADL. However, such a compilation is not generally possible, at least not in an efficient way, because basic action theories exhibit several features that cannot be compactly represented in ADL; for example, all ADL representations equivalent to a given basic action theory might be exponentially larger, or require exponentially longer plans, which renders a compilation infeasible in practice.

This difference in expressivity naturally gives rise to the question *which* basic action theories can (or cannot) be compiled to ADL. A first answer to this question has been provided by Eyerich et al. (2006), who identify a fragment of basic action theories that is expressively equivalent to ADL.

However, it is not clear whether this fragment is maximal. Eyerich et al. propose six restrictions to basic action theories, and it is not clear whether all of them are actually necessary to allow compilability to ADL. As a first step in analyzing the maximality of the identified fragment, our earlier paper (Röger and Nebel 2007) examines the role of functions (one of the six restrictions). We show that functions add expressive power as soon as they can denote objects besides a finite set of constants. In this paper, we examine the remaining five restrictions and, thus, identify a maximal subset of basic action theories with the same expressivity as ADL.

The rest of the paper is structured as follows: First, we introduce the framework used to compare the expressivity of planning formalisms before we briefly sketch the two formalisms, ADL and basic action theories. The next four sections contain our results about the questions that were still

unclear: the restrictions on successor state axioms, the specification of the predicates in the initial database, the necessity of unique names axioms for constants and the existence of a domain closure axiom. We close with a brief conclusion.

For most results we only present proof sketches. Detailed proofs can be found in a technical report (Röger and Nebel 2008).

Compilation Schemes

In order to compare the expressive power of different planning formalisms we use *compilation schemes*, a technique introduced by Nebel (2000).

In the formalisms we consider, a planning instance $\Pi = \langle \Xi, \mathbf{C}, \mathbf{I}, \mathbf{G} \rangle$ consists of the *domain structure* Ξ (which contains schematic descriptions of the possible actions), a finite set of constant symbols $\mathbf{C}$, the *initial state specification* $\mathbf{I}$, and the *goal specification* $\mathbf{G}$. A *plan* is a sequence of actions that leads from the initial states to a goal state.

Compilation schemes compare how concisely planning domains and plans can be expressed in different formalisms, not how concisely an individual planning instance can be expressed. The intuition behind compilation schemes is that it is justifiable to perform significant work on translating a domain description from one formalism to another, as long as this remains a one-off effort, and individual *instances* of the domain can subsequently be transformed efficiently.

Indeed, as we are measuring the *expressivity* of a formalism, the mapping may use arbitrary computational resources; it does not even need to be computable. However, we do require that the result is at most polynomially sized, and that the transformation of the domain description must not depend on the initial state and the goal. The translation of the initial state and the goal is done by so-called *state translation functions* which are very limited: they should be efficiently computable and not depend on the whole specification.

To compare the expressive power of two planning formalisms we moreover have to measure the size of the generated plans. Before we can state concretely which demands we make on the plan length, we first need to define the notion of compilation schemes formally.

A compilation scheme maps each planning instance Π of the source formalism $\mathcal{X}$ to an instance $F(\Pi)$ of the target formalism $\mathcal{Y}$.

Definition 1. *Let* $\mathbf{f}$ *be a tuple* $\langle f_\xi, f_c, f_i, f_g, t_c, t_i, t_g \rangle$ *of functions that induces a function* F *from* $\mathcal{X}$*-instances* $\Pi = \langle \Xi, \mathbf{C}, \mathbf{I}, \mathbf{G} \rangle$ *to* $\mathcal{Y}$*-instances* $F(\Pi)$*:*

$$F(\Pi) = \langle f_\xi(\Xi), f_c(\Xi) \cup t_c(\mathbf{C}), \\ f_i(\Xi) \cup t_i(\mathbf{C}, \mathbf{I}), f_g(\Xi) \cup t_g(\mathbf{G}) \rangle.$$

We call $\mathbf{f}$ *a compilation scheme from* $\mathcal{X}$ *to* $\mathcal{Y}$ *iff*

1. *there exists a plan for* Π *iff there exists a plan for* $F(\Pi)$*,*
2. *the* state translation functions t_c, t_i *and* t_g *are* polynomial-time computable*,*
3. *and the size of the results of* f_ξ*,* f_c*,* f_i *and* f_g *is* polynomial in the size of the arguments*.*

If a compilation scheme $\mathbf{f}$ has the property that for each plan P of instance Π there is a plan P' solving $F(\Pi)$ such that $\|P'\| \leq c\|P\| + k$ for some positive integers c and k, we say that $\mathbf{f}$ is a *compilation scheme preserving plan size linearly*, and conclude that the target formalism is at least as expressive as the source formalism. If plans are required to grow polynomially and there is no other compilation scheme preserving plan size linearly, this indicates that the source formalism is more expressive than the target formalism. If there is even a super-polynomial blow-up required, there must be a huge difference in expressive power.

The ADL Fragment of PDDL

In 1998, McDermott et al. introduced the Planning Domain Definition Language (PDDL), which, with some revisions (Fox and Long 2003; Gerevini and Long 2005), has since become a standard for the representation of planning domains. Two aspects that distinguish PDDL from basic action theories (see below) are that all constants denote distinct objects and that there are no objects in addition to the constants. In this article we are interested in the ADL fragment of PDDL, i. e., the language we get if exactly the `:adl` requirement is set. Beyond the definition of standard STRIPS operators, preconditions and goals may contain negation, disjunction and quantification and there are quantified and conditional effects. The main difference to full PDDL 2.1 is that numeric state variables and durative actions are not allowed. The PDDL syntax is quite self-explanatory and it is not crucial for the understanding of this paper to know all the details. Thus, we do not explain it here but only refer to Fox and Long (2003) where a detailed description of the syntax and semantics of PDDL can be found.

Basic Action Theories

Basic action theories are a logical approach to modeling dynamically changing worlds, based on the *situation calculus*. They were introduced by Reiter (2001) who also formed the version of the situation calculus which is in use today and which we briefly introduce before the presentation of basic action theories.

The situation calculus is a second-order language designed for representing dynamic systems. All changes to the world are the result of *actions* and each action leads to a new *situation*, which, hence, can be identified with a sequence of actions. The empty sequence corresponds to the *initial situation*, denoted by s_0. Besides *actions* and *situations* there is a third sort *object* that is used for everything else. In the following, we use variables a for actions, s for situations and $x, y, \ldots$ for objects (each with subscripts and superscripts). There is a special predicate *Poss*(a, s) meaning that it is possible to perform action a in situation s. All situations except s_0 are formed with a function *do*(a, s) meaning that action a is performed in situation s. Functions and predicates whose values vary from one situation to the next are called *fluents* and take a situation term as their last argument. There are also some foundational axioms for situations, which ensure that dynamic worlds in the situation calculus behave like one assumes intuitively.

In the following we omit leading universal quantifiers in sentences. The convention is that any free variables in such expressions are implicitly universally quantified.

For the definition of the notion of basic action theories we need to introduce some further concepts (Reiter 2001).

Unique names axioms for actions state whether two actions are equal. Distinct action names A and B define distinct actions and identical actions have identical arguments:

$$A(\bar{x}) \neq B(\bar{y})$$

$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n$$

A formula is called *uniform* in situation s if it does not mention the predicate *Poss* and the only permitted occurrence of a situation term is the occurrence of situation s in the situation argument position of a fluent.

Whether it is possible to perform an action is stated by so-called *action precondition axioms*, which are of the form

$$Poss(A(x_1, \ldots, x_n), s) \equiv \Pi_A(x_1, \ldots, x_n, s),$$

where A is an action function symbol with arity n and $\Pi_A(x_1, \ldots, x_n, s)$ is a formula that is uniform in s and whose free variables are among $x_1, \ldots, x_n, s$.

The value of a relational fluent after performing an action a is given by a *successor state axiom* of the form

$$F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, \ldots, x_n, a, s), \quad (1)$$

where $\Phi_F(x_1, \ldots, x_n, a, s)$ is a formula uniform in s whose free variables are among $x_1, \ldots, x_n, a, s$. Similarly, a successor state axiom for a functional fluent is of the form

$$f(x_1, \ldots, x_n, do(a, s)) = y \equiv \Phi_f(x_1, \ldots, x_n, y, a, s)$$

with conditions analogous to those of the relational fluents.

After the introduction of these concepts we now can state the definition of basic action theories from Reiter (2001):

Definition 2. *A basic action theory T is a theory of the form*

$$T = \Sigma \cup T_{SSA} \cup T_{APA} \cup T_{UNA} \cup T_{s_0},$$

where

- Σ *are the foundational axioms for situations,*
- T_{SSA} *is a set of successor state axioms for functional and relational fluents,*
- T_{APA} *is a set of action precondition axioms,*
- T_{UNA} *is the set of unique names axioms for actions, and*
- T_{s_0} *is the* initial database*, a set of first-order sentences that are uniform in s_0.*

*The state successor axiom for a functional fluent f must actually define a value for f in the next situation, and this value must be unique (*functional fluent consistency property*).*

Such a basic action theory T together with a goal formula $G(s)$ whose only free variable is s form a planning task. A situation s is a plan for G (relative to T) iff

$$T \models executable(s) \wedge G(s),$$

where $executable(s)$ means that the action sequence s can be executed with respect to *Poss*.

Starting from this definition, Eyerich et al. added several restrictions to achieve the same expressivity as ADL.

Restricted Basic Action Theories

A restricted basic action theory (RBAT) is a basic action theory that satisfies the following restrictions:

R1 Usage of functions is restricted to two sorts: constants (functions of sort $\epsilon \rightarrow$ *object*), and action functions with object arguments (functions of sort *object*$^n \rightarrow$ *action*).

R2 Successor state axioms are of a certain form. The successor state axiom of a relational fluent F fits the schema

$$F(x_1, \ldots, x_n, do(a, s)) \equiv \bigvee_{l=1}^{p} \psi_l \quad (2)$$

for some $p > 0$, where exactly one ψ_l is of the form

$$F(x_1, \ldots, x_n, s) \Big[\wedge \neg \Big([\exists \ldots] \left(a = A_1 \left(y_{11}, \ldots, y_{1m_1}\right) [\wedge \phi_1]\right) \vee \cdots \vee [\exists \ldots] \left(a = A_q \left(y_{q1}, \ldots, y_{qm_q}\right) [\wedge \phi_q]\right) \Big) \Big]. \quad (3)$$

All the other ψ_l are of the form

$$[\exists \ldots] \left(a = A \left(y_1, \ldots, y_m\right) [\wedge \phi_l]\right). \quad (4)$$

The existential quantification ranges over all y_i for which there is no x_j with $x_j = y_i$, and thus over all variables that are parameters of the action but not of the fluent. The parts between square brackets are optional. Each action may appear in at most one subformula $a = A_i \left(y_{i1}, \ldots, y_{im_i}\right)$ in (3) and in at most one subformula $a = A_i \left(y_1, \ldots, y_m\right)$ in the expressions of form (4).

R3 The initial database must consist of exactly the following sentences:

1. For each relational fluent F there is either an expression
$$\neg F(x_1, \ldots, x_n, s_0) \quad (5)$$
or an expression
$$F(x_1, \ldots, x_n, s_0) \equiv (x_1 = d_{11} \wedge \cdots \wedge x_n = d_{1n}) \vee \cdots \vee (x_1 = d_{m1} \wedge \cdots \wedge x_n = d_{mn}). \quad (6)$$
2. There are analogous expressions for all situation-independent predicates.
3. There is a *domain closure axiom* $(x = d_1) \vee \cdots \vee (x = d_n)$ for constants.
4. There are unique names axioms $c_i \neq c_j$ for each pair c_i, c_j of different constants.

Eyerich et al. show that these restrictions lead to the same expressive power as ADL by giving compilation schemes in both directions. For this paper, only the compilation scheme from RBAT to ADL is relevant. Most interesting is the compilation of the successor state axioms and action precondition axioms to the actions. The parameters and the precondition of the action are taken from the action precondition axioms whereas the effect of each action on the fluents is collected from the successor state axioms.

For the compilation Eyerich et al. use the fact that uniform first-order formulae can easily be expressed in PDDL syntax. We use the notation $\Theta^{\mathrm{PDDL}}(\phi)$ for the PDDL formula resulting from a uniform first-order formula ϕ.

Restrictions on Successor State Axioms

Of the six restrictions (R1, R2, R3.1–3.4), one has been studied before: in our earlier work, we show that the restrictions on functions (R1) are stronger than necessary (Röger and Nebel 2007). In particular, in the presence of the domain closure axiom (R3.3), arbitrary functions of sort *object*^{m} $\times$ *situation* $\rightarrow$ *object* may be allowed. However, this is not possible if R3.3 is weakened.

We now consider the remaining restrictions, beginning with the restrictions on successor state axioms (R2). The required schema in R2 is based on the idea that a fluent is true in a situation iff it has been made true by the last action or if it has been true in the previous situation and has not been made false. In the following we will show how the compilation scheme of Eyerich et al. can be altered to transform the general form of successor state axioms, too.

Theorem 1 (R2 is not necessary)**.** *Restriction R2 is not necessary for the compilability of RBAT to ADL.*

Proof sketch. We give a compilation of the general form of successor state axioms to PDDL action effects. Recall this general form (1):

$$F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, \ldots, x_n, a, s)$$

To compile an action $A(y_1, \ldots, y_m)$ we substitute a with $A(y_1, \ldots, y_m)$ and receive an expression

$$F(x_1, \ldots, x_n, do(A(y_1, \ldots, y_m), s)) \equiv \Phi_F^A(x_1, \ldots, x_n, y_1, \ldots, y_m, s),$$

where all free variables in Φ_F^A are among $x_1, \ldots, x_n$,, $y_1, \ldots, y_m, s$. The effect of action *A* on fluent *F* can then be translated to PDDL as

```
(forall (?x1 ... ?xn)
  (and (when
         Θ^PDDL(Φ^A_F(x1 ..., xn, y1, ..., ym, s))
         (F ?x1 ... ?xn))
       (when
         (not Θ^PDDL(Φ^A_F(x1 ..., xn, y1, ..., ym, s)))
         (not (F ?x1 ... ?xn)))))
```

The complete effect of each action is a conjunction of such expressions for each predicate. The precondition of the action is the same as in the compilation of Eyerich et al.

It is easy to see that the effect of an action on a predicate is the same in both formalisms. Note that the given compilation unnecessarily solves the frame problem for the relational fluents twice: the PDDL semantics leaves the value of each atom that is not explicitly set by an action unaltered. In addition, in each action effect each fluent explicitly gets assigned a value. In practice one would therefore further simplify the action effects. □

We have thus shown that we can easily get rid of the restriction on the successor state axioms. In the next sections we will examine whether this also holds for the restrictions on the initial database.

Predicate Specifications in the Initial Database

Restrictions R3.1 and R3.2 require that the initial database enumerates all true ground atoms for the initial situation and situation-independent predicates. There are two ways of loosening these restrictions: firstly, there could be no such expression for some of the predicates (leaving their interpretation unspecified) and secondly, less restricted formulae which still guarantee a unique model could be allowed in the predicate specifications.

Less Restricted Formulae

We begin with the latter case. One possibility is to allow arbitrary first-order formulae that are uniform in s_0 on the right-hand side of equivalence (6). This would also permit "recursive" definitions which are known to be non-compilable (Thiébaux, Hoffmann, and Nebel 2005). However, we can actually prove a stronger result which holds for *acyclic* specifications, i. e., for the case where there is a strict order $<$ on the set of predicates such that each specification of a predicate P depends only on atoms of predicates P' with $P' < P$.

Theorem 2 (R3.2 is necessary)**.** *If arbitrary acyclic predicate specifications are permitted in the initial database of a RBAT then there is no compilation scheme to ADL preserving plan size exponentially unless* PSPACE $=$ P.

Proof. Consider a family of planning tasks with one unary situation-independent predicate *isTrue* and a 0-ary predicate *goal*, but no fluent and no action. We use two constants $\mathbf{T}$ and $\mathbf{F}$ where *isTrue* is true exactly for $\mathbf{T}$. Further, the initial database contains the domain closure axiom and a unique names axiom. The goal formula requires predicate *goal*() to be true. Thus, the only aspect that differs from task to task is the specification of predicate *goal* in the initial database.

Let $\mathbf{f} = \langle f_\xi, f_c, f_i, f_g, t_c, t_i, t_g \rangle$ be a compilation scheme to ADL that preserves plan size exponentially. As the domain and the goal description are fixed, the results of f_g, t_g and f_ξ are fixed and provide us a fixed ADL domain and a fixed goal. As there are no actions in the original instance, plans of the original instance have a length bound of 0. Since $\mathbf{f}$ preserves plan size exponentially, this implies a constant bound k on shortest plan lengths for tasks of the ADL domain. Note that in this setting (fixed domain, fixed goal, constant length bound), ADL planning can be decided in P by testing all action sequences of length at most k.

With this compilation scheme $\mathbf{f}$ we can decide the quantified Boolean formula problem in polynomial time: Assume a quantified Boolean formula ϕ. We convert ϕ to a first-order formula ϕ' by substituting each occurrence of a variable x by *IsTrue*(x). Predicate *goal* is then initialized as *goal*$() \equiv \phi'$. We can now use the compilation scheme $\mathbf{f}$ to translate the initial database to an initial state for the ADL task and test (k-bounded) plan existence of that task in polynomial time. With the quantified Boolean formula problem being PSPACE-complete, this implies PSPACE $=$ P. □

There are surely other possibilities of weakening the restrictions on the right-hand side of equivalence (6). Every specification which defines a unique model whose true

ground atoms can be enumerated in polynomial time can clearly be compiled. Even in cases with more than polynomially many true ground atoms, it is often possible to compile a space-saving representation by means of initializing actions. However, examining this in more detail does not seem worthwhile to us.

Incomplete Information

A second possibility of weakening restriction R3.2 is to allow omitting a specification for a given predicate P altogether, so that a plan must work for all interpretations of P. This is both interesting in its own right and useful for the discussion of unique names axioms in the following section.

Here, we can show that we lose compilability to ADL even if there may only be a single unary situation-independent predicate whose truth values are not specified in the initial database. In the following, we denote the resulting formalism by $RBAT_{incomp}$.

We begin with a lemma on the complexity of plan existence for this formalism. The proof is a variation of Rintanen's (2004) proof on the complexity of propositional planning without observability.

Lemma 1. *The plan existence problem for $RBAT_{incomp}$ is* EXPSPACE*-hard.*

Proof sketch. Let M be a deterministic Turing machine with space bound 2^{n^k} for input strings of length n. It is possible to formulate (and generate in polynomial time) a family (T_w) of basic action theories, one for each possible input w to M, so that there is a plan for T_w iff M halts on w. Different basic action theories of the family only differ in the initial database and in the set of constants. The key idea is to keep track of only one randomly chosen tape cell (the *watched* tape cell) and to ensure that the Turing machine is reliably simulated relative to this tape cell. The cell is picked by means of the unspecified unary predicate. As a plan must hold for all models, the Turing machine must consequently be reliably simulated for the entire tape.

We use n^k constants $\mathbf{p}_0, \ldots, \mathbf{p}_{n^k-1}$ and an (unspecified) unary predicate *watched* and interpret the truth value of $watched(\mathbf{p}_i)$ as the value of the i-th bit of the number of the watched tape cell. Otherwise, we use the common way to formulate a Turing machine as a basic action theory. The only difference is that we follow the idea of Rintanen to ensure the correct simulation only relative to the watched tape cell. The actions in a plan exactly define the transitions of the Turing machine. Consequently, as a plan must work for all models (regardless of the choice of watched tape cell), the Turing machine is reliably simulated for the entire tape. □

Lemma 1 leads us directly to the theorem we actually want to show:

Theorem 3 (R3.2 is necessary)**.** *There is no compilation scheme from $RBAT_{incomp}$ to ADL.*

Proof. Let M be a Turing Machine with space bound 2^{n^k} that accepts an EXPSPACE-hard language. Let $\Xi = \Sigma \cup T_{SSA} \cup T_{APA} \cup T_{UNA}$ be the domain description of the basic action theory for M according to the proof of Lemma 1.

Assume there is a compilation scheme $\mathbf{f}$ to ADL. Then there exists an ADL domain description $f_\xi(\Xi)$ corresponding to M. We could, in polynomial time, translate each input of the Turing machine to a set of constants and an initial state such that there exists a plan iff M halts. As the plan existence problem for ADL with a fixed domain is PSPACE-complete this implies that EXPSPACE $\subseteq$ PSPACE, which is known to be false. □

The results in this section apply to the specification of situation-independent predicates. Obviously they directly carry over to the specification of relational fluents:

Corollary 1 (R3.1 is necessary)**.** *Omitting restriction R3.1 adds expressive power to RBAT.*

There are two remaining restrictions concerning the initial database: the domain closure axiom and the unique names axioms for constants. As the latter is closely related to the previous theorem, we examine it first.

Unique Names Axioms for Constants

Restriction R3.4 states that the initial database contains a unique names axiom $c_i \neq c_j$ for each pair of different constants c_i and c_j. We will see that omitting this restriction leads to at least the same expressive power as permitting a single unary unspecified predicate. In the previous section we have seen that the resulting formalism cannot be compiled to ADL.

In the following we denote the formalism resulting from RBAT by omitting restriction R3.4 by $RBAT_{noUNA}$.

Lemma 2. *There is a compilation scheme from $RBAT_{incomp}$ to $RBAT_{noUNA}$ preserving plan size linearly.*

Proof. Let P be the unspecified unary predicate. For each constant $\mathbf{c}$ in the set of constants C, we introduce two new auxiliary constants $\mathbf{c}'$ and $\mathbf{c}''$. The key idea is to replace atoms $P(x)$ where x denotes the same object as constant $\mathbf{c}$ by an equality test for $\mathbf{c}'$ and $\mathbf{c}''$. For this purpose, we introduce a predicate *partners* which is initialized as

$$partners(x, x', x'') \equiv \bigvee_{\mathbf{c} \in C} (x = \mathbf{c} \wedge x' = \mathbf{c}' \wedge x'' = \mathbf{c}'') .$$

We introduce unique names axioms in such a way that for all $\mathbf{c} \in C$, the equality of $\mathbf{c}'$ and $\mathbf{c}''$ remains unspecified. All other pairs of constants are different from each other.

We then substitute each occurrence of $P(x)$ by

$$\exists x', x'' \, (partners(x, x', x'') \wedge x' = x'') .$$

In addition, we have to prevent the usage of the newly introduced objects everywhere else. We do this by means of a new predicate *orig* identifying the original objects. All quantifications in the original theory are modified to only range over objects that satisfy *orig*.

Clearly, each plan of the original task is a plan for the resulting task and vice versa. Moreover, the modified basic action theory can be computed in polynomial time, and the independence requirements for compilation schemes are satisfied. Hence, we have presented a compilation scheme from $RBAT_{incomp}$ to $RBAT_{noUNA}$ preserving plan size linearly (in fact, exactly). □

The lemma shows that $RBAT_{noUNA}$ is at least as expressive as $RBAT_{incomp}$, which according to Theorem 3 is more expressive than ADL. This directly leads to the following theorem.

Theorem 4 (R3.4 is necessary)**.** *There is no compilation scheme from $RBAT_{noUNA}$ to ADL.*

The only restriction not examined so far is the domain closure axiom for constants.

Domain Closure Axiom

The domain closure axiom states that there are no objects except for the named constants. At first glance, one would assume that this restriction is necessary to stay within the same expressive power as ADL because in this formalism there only can be a finite number of objects which all must be declared. If we abandon the domain closure axiom, there are infinitely many models of the initial database. Furthermore, actions can behave qualitatively differently for different models; for example, a conditional effect might only trigger if there is at least one object that is different from all named constants. However, a closer look reveals that the other restrictions on the initial database are so restrictive that we can omit the domain closure axiom nevertheless.

Theorem 5 (R3.3 is not necessary)**.** *Omitting the domain closure axiom does not increase the expressive power of RBAT.*

Proof sketch. We refer to objects of the model which are different from all named constants as "unnamed objects". We begin our proof with two observations:

- In all situations that occur in a plan, all unnamed objects are interchangeable. For example, if $P(o_1, o_2, o_3, s)$ is true, o_1 and o_3 are different unnamed objects, and o_2 is a named object, then $P(o'_1, o_2, o'_3, s)$, where again o'_1 and o'_3 are different unnamed objects, is also true.
- For determining whether there is a plan, it is sufficient to consider only the models with up to k unnamed objects, where k is the maximum number of nested quantifiers in the theory (including implicit universal quantifiers).

The first observation can easily be shown by induction over the situations: In the initial situation (and for situation-independent predicates) all atoms containing unnamed objects are false due to R3.1 and R3.2.

For the inductive case, consider situation $s' = do(A, s)$. For each relational fluent P, the truth value of $P(\bar{x}, s')$ is determined by a formula $Q(\bar{x}, s)$ which is uniform in s (by the definition of successor state axioms). If we evaluate this formula for different tuples $\bar{x}$ which only differ by permuting unnamed objects, the same truth values are obtained: by the induction hypothesis, unnamed objects are interchangeable in situation s.

If all unnamed objects are interchangeable, then only the *number* of unnamed objects can affect the validity of a plan. For statements like "There are m distinct unnamed objects" it is necessary to bind at least m variables. With quantifier depth bounded by k, it is not possible to make statements about more than k objects, and hence all models with more than k unnamed objects behave the same as the model with exactly k such objects.

These observations lead quite directly to a compilation scheme from RBAT without a domain closure axiom to RBAT. We explicitly represent the discriminable models and modify the basic action theory in such a way that each action is executed in all models in parallel. The goal formulation is then modified such that the original goal must be true in all models. □

Omitting More than One Restriction

We have shown that some of the restrictions of Eyerich et al. are necessary to stay within the expressivity of ADL while others can be removed or weakened:

- Functions of more general sorts are permissible if their initial values are completely specified for the initial situation, i. e., R3.1 and R3.2 are extended to functions (R1; Röger and Nebel 2007).
- General successor state axioms can be permitted (R2).
- The domain closure axiom may be omitted (R3.3).

Each of these results has been achieved under the assumption that all other restrictions are retained. What happens if we omit more than one restriction at once?

If more general functions are allowed (R1 is weakened), the necessary modification to restriction R3.1 states that all initial function values must be enumerated explicitly (Röger and Nebel 2007, Eq. 23):

$$\begin{aligned} f(x_1, \ldots, x_n, s_0) = y \equiv \\ (x_1 = d_{11} \wedge \cdots \wedge x_n = d_{1n} \wedge y = d_1) \vee \cdots \vee \\ (x_1 = d_{m1} \wedge \cdots \wedge x_n = d_{mn} \wedge y = d_m), \end{aligned}$$

where the d_i are the named constants. Note that this specification *necessarily implies* that there are no objects besides the named constants. If there existed an unnamed object o in a given model, then the above equation implies $f(o, \ldots, o, s_0) \neq y$ for all objects y. However, functions must have a defined value. Hence, no such model exists.

Therefore, we can weaken R1 and remove R3.3 at the same time. Either the basic action theory contains no functions of the form forbidden by R1 (in which case it does not matter that R1 was weakened), or the removal of R3.3 does not matter because the domain closure axiom is implied.

Finally, the compilation of general successor state axioms (R2) clearly does not depend on or interfere with the domain closure axiom or the usage of functions. Thus, we can combine all three changes without adding expressive power.

Conclusion

We have seen that of the six restrictions of Eyerich et al., three can be omitted. (In the case of R1, this requires adjustments to R3.1 and R3.2 as discussed by Röger and Nebel, 2007.) The remaining restrictions define a certain schema for the initial database but do not affect the other parts of basic action theories. In summary, the initial database must consist of exactly the following sentences:

1. For each relational fluent F there is either an expression $\neg F(x_1, \ldots, x_n, s_0)$ or an expression

$$F(x_1, \ldots, x_n, s_0) \equiv (x_1 = d_{11} \wedge \cdots \wedge x_n = d_{1n}) \vee \cdots \vee (x_1 = d_{m1} \wedge \cdots \wedge x_n = d_{mn}) .$$

2. There are analogous expressions for all situation-independent predicates.
3. There are analogous expressions for all functions except constants and action functions with object arguments.
4. There are unique names axioms $c_i \neq c_j$ for each pair c_i, c_j of different constants.
5. Optionally, there may be a domain closure axiom of the form $(x = d_1) \vee \cdots \vee (x = d_n)$.

If a basic action theory complies with these restrictions, it can be compiled to ADL. Furthermore, all the required statements are necessary in the sense that removing them extends the expressiveness of the formalism beyond ADL.

As a final remark, we note that there may be other compilable classes of basic action theories, based on restrictions that are orthogonal to the ones introduced by Eyerich et al. In this work, we started from these restrictions because they are the only ones that are discussed in the current literature (Eyerich et al. 2006; Claßen et al. 2007; Röger and Nebel 2007), and the question which of these restrictions are necessary for compilability has been open. This open question is now resolved, bringing the theoretical analysis in this line of work to conclusion.

Nevertheless, there are still some interesting questions left open for future work. On the theoretical front, we saw that the most obvious reason for the mismatch between basic action theories and ADL (and PDDL) is the possibility of specifying incomplete information, which PDDL does not permit. However, there exist extensions of PDDL to nondeterministic planning, such as the non-probabilistic fragment of the PPDDL language, used for the nondeterministic track of the Fifth International Planning Competition (Gerevini, Bonet, and Givan 2006). It may be worth investigating in how far PPDDL can capture the aspects of basic action theories that ADL cannot.

On the practical side, although there is a proof of concept for the integration of a planning system into Golog (Claßen et al. 2007), a comprehensive empirical evaluation of a fully automated integration is still missing.

Acknowledgments

This work was supported by the German Research Foundation (DFG) by DFG grant NE 623/10-1 and as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See http://www.avacs.org/ for more information.

References

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. ICAPS-07*, 26–33.

Claßen, J.; Eyerich, P.; Lakemeyer, G.; and Nebel, B. 2007. Towards an integration of Golog and planning. In *Proc. IJCAI-07*, 1846–1851.

Eyerich, P.; Nebel, B.; Lakemeyer, G.; and Claßen, J. 2006. GOLOG and PDDL: What is the relative expressiveness? In *Proc. PCAR-06*, 93–104.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Univ. of Brescia, Italy.

Gerevini, A.; Bonet, B.; and Givan, B., eds. 2006. *Fifth International Planning Competition. Competition booklet*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1–3):59–83.

McDermott, D., et al. 1998. PDDL—the planning domain definition language. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *JAIR* 12:271–315.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR-89*, 324–332.

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

Rintanen, J. 2004. Complexity of planning with partial observability. In *Proc. ICAPS-04*, 345–354.

Röger, G., and Nebel, B. 2007. Expressiveness of ADL and Golog: Functions make a difference. In *Proc. AAAI-07*, 1051–1056.

Röger, G., and Nebel, B. 2008. On the relative expressiveness of ADL and Golog. Technical Report 237, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *AIJ* 168(1–2):38–69.