Towards an Integration of Golog and Planning

Jens Claßen^{+} and **Patrick Eyerich*** and **Gerhard Lakemeyer**$^{+}$ and **Bernhard Nebel***
$^{+}$ Department of Computer Science, RWTH Aachen, 52056 Aachen, Germany
$\langle$classen$|$gerhard$\rangle$@cs.rwth-aachen.de
* Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany
$\langle$eyerich$|$nebel$\rangle$@informatik.uni-freiburg.de

Abstract

The action language Golog has been applied successfully to the control of robots, among other things. Perhaps its greatest advantage is that a user can write programs which constrain the search for an executable plan in a flexible manner. However, when general planning is needed, Golog supports this only in principle, but does not measure up with state-of-the-art planners. In this paper we propose an integration of Golog and planning in the sense that planning problems, formulated as part of a Golog program, are solved by a modern planner during the execution of the program. Here we focus on the ADL subset of the plan language PDDL. First we show that the semantics of ADL can be understood as progression in the situation calculus, which underlies Golog, thus providing us with a correct embedding of ADL within Golog. We then show how Golog can be integrated with an existing ADL planner for closed-world initial databases and compare the performance of the resulting system with the original Golog.

1 Introduction

The action language Golog [Levesque *et al.*, 1997] has been applied successfully to the control of robots [Burgard *et al.*, 1999], among other things. It is based on Reiter's version of the situation calculus [Reiter, 2001], where so-called basic action theories (BATs), which define action preconditions and effects together with an initial situation, are combined with a programming language, which offers imperative programming constructs as well as non-deterministic action choices. Perhaps the greatest advantage of Golog is that a user can write programs which constrain the search for an executable plan in a flexible manner. However, when general planning is needed, Golog supports this only in principle through the use of non-deterministic actions, but does not measure up with state-of-the-art planners.

As a running example, consider a taxi-driver agent in a grid world, whose task it is to deliver passengers to their destinations. Its actions are pickup, dropoff, and moving up, down, left or right one square in the grid. In Golog, the agent's high-level control program can then be specified like this:

While there are passengers to be served do the following: choose a passenger, plan a route to her location, follow that route, pick up the passenger, plan a route to her destination, follow that route and drop her off.

Planning can easily be specified in Golog as trying out actions nondeterministically in a forward-search manner until a given goal is satisfied [Reiter, 2001]. While possible in principle, this quickly becomes infeasible, in our example even for small grid worlds. Modern planners like FF [Hoffmann and Nebel, 2001] or HSP2 [Bonet and Geffner, 2001], on the other hand, are capable of handling much larger planning problems. So it seems natural to try to combine Golog with a modern planner. Here we focus on planners for the ADL [Pednault, 1989] subset of the plan language PDDL [Fox and Long, 2003].

For the integration, at least two issues arise: differences in expressiveness and the correctness of the embedding. While BATs in the situation calculus can use the full power of first-order logic, ADL descriptions are more restricted. For example, they assume a finite domain and the initial state consists of literals only. As we will see, it is not very difficult to map ADL into BATs. For the purposes of this paper, it will suffice to simply use a BAT that is the result of such a mapping; a companion paper [Eyerich *et al.*, 2006] discusses the expressiveness issue in more detail.

As for correctness, we need to make sure that a plan returned by an ADL planner for a given goal and state description is also a legal plan in the situation calculus for the corresponding situation. The semantics of PDDL's ADL subset [Fox and Long, 2003] is given in terms of state transitions as an adaptation of Lifschitz' semantics of STRIPS [Lifschitz, 1986]. Roughly, a state consists of a set of literals, and a transition for a plan operator is obtained by adding and deleting literals. By extending earlier work by Lin and Reiter [1997], we are able to show that the ADL semantics is captured precisely by progression in the situation calculus, which refers to updating the description of an initial situation so that conclusions about the future that can be drawn using the updated theory are the same as those drawn from the theory before the update. As a nice side-effect, we obtain a purely declarative semantics of ADL within the situation calculus.[1]

[1]Pednault's [1989; 1994] original semantics for ADL defined operators similarly by additions and deletions, however of tuples in relations and functions of first-order structures. He also derived sit-

To obtain our results, we use a variant of the situation calculus called $\mathcal{ES}$ recently proposed by Lakemeyer and Levesque [2004]. Compared to Reiter's situation calculus, this has at least two advantages: for one, the translation of ADL into the new situation calculus is simpler as there are no situation terms to worry about (in $\mathcal{ES}$, situations occur only in the semantics); for another, semantic definitions like progression become simpler as it is no longer necessary to consider arbitrary first-order structures but only certain ones over a fixed universe of discourse. These simplifications do not lead to a loss of expressiveness. In fact, Lakemeyer and Levesque [2005] showed that $\mathcal{ES}$ captures precisely the non-epistemic fragment of the situation calculus and Golog.

After establishing that ADL plans are correct for the corresponding BAT, we turn to the embedding of an ADL planner into Golog. In particular, we use the latest version of FF [Hoffmann and Nebel, 2001] together with an implementation of IndiGolog [Sardina *et al.*, 2004], which incorporates on-line execution, Reiter's forward-search planner and progression. We present first experimental results showing the pay-off of using Golog with FF versus pure Golog.

The paper proceeds as follows. We first introduce $\mathcal{ES}$ and show how BATs are formulated in this logic. Next, we define ADL problem descriptions and provide a formal semantics by mapping them into basic action theories. In Section 4, we define progression and establish the correctness of updating an ADL state with respect to progression. In Section 5, we discuss the integration of Golog and FF. Then we conclude.

2 The logic $\mathcal{ES}$

The language is a first-order modal dialect with equality, predicate and function symbols. For simplicity we assume that all predicates are *fluent* and that all function symbols are *rigid*, i.e. predicates may be changed by actions while functions may not. The logical connectives are $\wedge$, $\neg$, $\forall$, together with the modal operators $\Box$ and $[r]$ where r may be any term, including a variable.[2] Other connectives like $\vee$, $\supset$, $\equiv$, and $\exists$ are used as the usual abbreviations.

Terms and formulas are built from these primitives in the usual way. We let R denote the set of all ground terms. For simplicity, instead of having variables of the *action* sort distinct from those of the *object* sort as in the situation calculus, we lump both of these together and allow to use any term as an action or as an object. We read $[r]\alpha$ as "α holds after action r" and $\Box\alpha$ as "α holds after any sequence of actions". We call a formula without free variables a *sentence* and a formula *fluent*, when it does not contain $\Box$ and $[r]$ operators and does not mention the special predicate *Poss* (which is used to denote executable actions). In addition, we introduce the following special notation: A *type* τ is a unary predicate; we write $\forall x{:}\tau.\,\phi$ instead of $\forall x.\,\tau(x) \supset \phi$.

The semantics: Intuitively, a world w will determine which fluents are true, but not just initially, also after any sequence of actions. Formally, let P denote the set of all pairs $\sigma{:}\rho$ where $\sigma \in R^*$ is considered a sequence of ground actions, and $\rho = F(r_1, \ldots, r_k)$ is a k-ary ground fluent atom. A *world* then is a mapping from P to $\{0, 1\}$.

First-order variables are interpreted substitutionally over the rigid terms R, i.e. R is treated as being isomorphic to a fixed universe of discourse. This is similar to $\mathcal{L}$ [Levesque and Lakemeyer, 2001], where standard names are used as the domain. Given a world w, for any sentence α, we define $w \models \alpha$ as $w, \langle\rangle \models \alpha$ where

$w, \sigma \models F(r_1, \ldots, r_k)$ iff $w[\sigma{:}F(r_1, \ldots, r_k)] = 1$;
$w, \sigma \models (r_1 = r_2)$ iff r_1 and r_2 are identical;
$w, \sigma \models (\alpha \wedge \beta)$ iff $w, \sigma \models \alpha$ and $w, \sigma \models \beta$;
$w, \sigma \models \neg\alpha$ iff $w, \sigma \not\models \alpha$;
$w, \sigma \models \forall x.\,\alpha$ iff $w, \sigma \models \alpha^x_r$, for every $r \in R$;
$w, \sigma \models [r]\alpha$ iff $w, \sigma \cdot r \models \alpha$;
$w, \sigma \models \Box\alpha$ iff $w, \sigma \cdot \sigma' \models \alpha$, for every $\sigma' \in R^*$.

The notation α^x_t means the result of simultaneously replacing all free occurrences of the variable x by the term t.

When Σ is a set of sentences and α is a sentence, we write $\Sigma \models \alpha$ (read: Σ logically entails α) to mean that for every w, if $w \models \alpha'$ for every $\alpha' \in \Sigma$, then $w \models \alpha$. Further $\models \alpha$ (read: α is valid) abbreviates $\{\} \models \alpha$.

Basic Action Theories: Basic action theories can be defined similar to Reiter's: A set of sentences Σ is a *basic action theory* iff it only mentions the fluents in a given set $\mathcal{F}$ and is of the form $\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, where

- Σ_0 is a finite set of fluent sentences,
- Σ_{pre} is a singleton of the form[3] $\Box \mathit{Poss}(a) \equiv \pi$, where π is fluent with a being the only free variable;
- Σ_{post} is a finite set of successor state axioms of the form[4] $\Box[a]F(\vec{x}) \equiv \gamma_F$, one for each fluent $F \in \mathcal{F} \setminus \{\mathit{Poss}\}$, where γ_F is a fluent formula whose free variables are among $\vec{x}$ and a.

The idea is that Σ_0 represents the initial database, Σ_{pre} is one large precondition axiom and Σ_{post} the set of successor state axioms for all fluents in $\mathcal{F}$ (incorporating Reiter's solution [Reiter, 2001] to the frame problem).

3 ADL

PDDL's ADL subset goes beyond STRIPS by supporting equality, conditional effects and typing. Also, preconditions may contain negation, disjunction and quantifiers.

ADL Problems: These are defined by

1. a finite list of types $\tau_1, \ldots, \tau_l$ and *Object*; along with this a finite list of statements of the form
$$\tau_i{:}(\text{either } \tau_{i_1} \cdots \tau_{i_{k_i}}) \qquad (1)$$
defining τ_i as the union of the τ_{i_j}; a *primitive type* is one that does not appear on the left-hand side of such a definition and is distinct from *Object*;

uation calculus formulas from ADL operators, but did not show the semantic correspondence between the two.

[2]The original $\mathcal{ES}$ also considers epistemic operators which we do not need here and are left out.

[3]Free variables are understood as universally quantified from the outside; $\Box$ has lower syntactic precedence than the logical connectives, i.e. $\Box \mathit{Poss}(a) \equiv \pi$ stands for $\forall a.\Box(\mathit{Poss}(a) \equiv \pi)$.

[4]The $[t]$ construct has higher precedence than the logical connectives. So $\Box[a]F(\vec{x}) \equiv \gamma_F$ abbreviates $\forall a.\Box([a]F(\vec{x}) \equiv \gamma_F)$.

2. a finite list of fluent predicates $F_1, \ldots, F_n$; associated with each F_j a list of types $\tau_{j_1}, \ldots, \tau_{j_{k_j}}$ (one for each argument of F_j, abbreviated as $\vec{\tau_{F_j}}$)
3. a finite list of objects with associated primitive types $o_1{:}\tau_{o_1}, \ldots, o_k{:}\tau_{o_k}$ (object symbols are constants);
4. a finite list of ADL operators $A_1, \ldots, A_m$ (see below);
5. an initial state I (see below) and
6. a goal description G in form of a precondition formula.

I and G may only contain the symbols from items 1 to 3. We further require that all the symbols are distinct. In particular, this forbids using a type also as an F_j and using an object also as an A_i. All objects are implicitly of type *Object*, which is a supertype of all other types. In the case of closed-world planning, the initial state I is simply given by a finite set of ground fluent atoms $F(\vec{r})$, otherwise it is a set of ground literals.

An *ADL operator* A is a triple $(\vec{y}{:}\vec{\tau}, \pi_A, \epsilon_A)$, where $\vec{y}{:}\vec{\tau}$ is a list of variables with associated types,[5] π_A is a precondition formula and ϵ_A an effect formula, both with free variables among $\vec{y}$. Both π_A and ϵ_A have to be constructed using only the $\vec{y}$ and the symbols from items 1 to 3 of the problem description. The name of the operator A has to be a p-ary function symbol, where p is the number of parameters of A.

Precondition formulas are the following: Every atomic formula $F(\vec{t})$ and every equality atom $(t_1 = t_2)$, where each of the t_i is either a variable or a constant, is a precondition formula. If ϕ_1 and ϕ_2 are precondition formulas, then so are $\phi_1 \wedge \phi_2$, $\neg\phi_1$ and $\forall x{:}\tau.\phi_1$. Without loss of generality, we assume that an *effect formula* is a conjunction of effects of the following forms, at most one of each kind for each fluent F_j:

$$\begin{array}{l}\forall \vec{x_j}{:}\vec{\tau_{F_j}}.\ (\gamma^+_{F_j,A}(\vec{x_j}, \vec{y}) \Rightarrow F_j(\vec{x_j}))\ , \\ \forall \vec{x_j}{:}\vec{\tau_{F_j}}.\ (\gamma^-_{F_j,A}(\vec{x_j}, \vec{y}) \Rightarrow \neg F_j(\vec{x_j}))\end{array} \quad (2)$$

The notation $\gamma \Rightarrow \psi$ stands for a conditional effect: if the precondition formula γ holds *before* performing the action, then ψ will become true afterwards.

An Example: Let us consider the taxi domain. For reasons of space we only consider the operators, initial state, and goal description:

1. Operators:
   $move = (\langle y_1{:}\mathit{Taxi},\ y_2{:}\mathit{Direction}\rangle,\ \text{TRUE},$
   $\forall x_1{:}\mathit{Object}.\ \forall x_2{:}\mathit{Xcoord}.\ \forall x_3{:}\mathit{Ycoord}.$
   $((\exists z_1{:}\mathit{Xcoord}.\ \exists z_2{:}\mathit{Ycoord}.$
   $\mathit{At}(y_1, z_1, z_2) \wedge \mathit{Connects}(z_1, z_2, y_2, x_2, x_3) \wedge$
   $(x_1 = y_1 \vee \mathit{In}(y_1, x_1))) \Rightarrow\ \mathit{At}(x_1, x_2, x_3)) \wedge$
   $\forall x_1{:}\mathit{Object}.\ \forall x_2{:}\mathit{Xcoord}.\ \forall x_3{:}\mathit{Ycoord}.$
   $((\exists z_1{:}\mathit{Xcoord}.\ \exists z_2{:}\mathit{Ycoord}.$
   $\mathit{At}(y_1, z_1, z_2) \wedge \mathit{Connects}(x_2, x_3, y_2, z_1, z_2) \wedge$
   $(x_1 = y_1 \vee \mathit{In}(y_1, x_1))) \Rightarrow\ \neg\mathit{At}(x_1, x_2, x_3)))$

   It is always possible to move a taxi into one of the four main cardinal directions; a move that would lead out of the grid has no effect. After moving, the location of the taxi and every passenger in it has changed accordingly.

   $pickup = (\langle y_1{:}\mathit{Taxi},\ y_2{:}\mathit{Passenger}\rangle,$
   $\exists z_1{:}\mathit{Xcoord}.\ \exists z_2{:}\mathit{Ycoord}.\ \mathit{At}(y_1, z_1, z_2) \wedge \mathit{At}(y_2, z_1, z_2)$
   $\wedge \neg\exists z_3{:}\mathit{Passenger}.\ \mathit{In}(y_1, z_3),$
   $\mathit{In}(y_1, y_2))$

   Picking someone up is only possible when the taxi is at the same location as the passenger and there is noone already in the taxi (we assume it can only hold one person at a time).

   $drop\_passenger = (\langle y_1{:}\mathit{Taxi}\rangle,\ \text{TRUE},$
   $\forall x_1{:}\mathit{Taxi}.\ \forall x_2{:}\mathit{Passenger}.$
   $(x_1 = y_1 \wedge \mathit{In}(x_1, x_2)) \Rightarrow \neg\mathit{In}(x_1, x_2))$

   Finally, it is always possible to drop the taxi's occupant (if any) at its current location.
2. Initial State (in a closed world):
   $I\ =\ \{\mathit{At}(\mathit{taxi1}, 5, 7), \mathit{At}(\mathit{passenger1}, 3, 1),$
   $\mathit{Destination}(\mathit{passenger1}, 1, 1),$
   $\mathit{Connects}(1, 1, \mathit{east}, 2, 1), \ldots$
   $\mathit{Connects}(10, 10, \mathit{north}, 10, 9)\}$
3. Goal description:
   $G\ = \forall p{:}\mathit{Passenger}.\ \forall x{:}\mathit{Xcoord}.\ \forall y{:}\mathit{Ycoord}.$
   $\mathit{Destination}(p, x, y) \supset \mathit{At}(p, x, y)$

In [Fox and Long, 2003] a state-transition semantics is defined for applying ADL operators in the closed-world case. The idea is roughly this: given any closed state I, represented as a set of ground atoms, and an operator $A(\vec{p})$ whose precondition is satisfied in I, then the application of $A(\vec{p})$ results in a new state I' which is obtained from I by deleting all those $F_j(\vec{o})$ where $\gamma^-_{F_j,A}(\vec{o}, \vec{p})$ holds in I and adding all those where $\gamma^+_{F_j,A}(\vec{o}, \vec{p})$ holds.

4 Mapping ADL to $\mathcal{ES}$

In this section, we generalize the approach of [Lin and Reiter, 1997] and show that applying ADL operators corresponds to a certain form of first-order progression in $\mathcal{ES}$. We begin by constructing, given an ADL problem description, a corresponding basic action theory Σ.

The Successor State Axioms Σ_{post}: It is not a coincidence that the normal form (2) resembles the effect axioms used by Reiter to construct his successor state axioms as a solution to the frame problem. Generalizing his approach (also applied in [Pednault, 1994]), we transform a set of operator descriptions to a set of successor state axioms as follows. Let

$$\gamma^+_{F_j} \overset{def}{=} \bigvee_{\gamma^+_{F_j,A_i} \in NF(A_i)} \exists \vec{y_i}.a = A_i(\vec{y_i}) \wedge \gamma^+_{F_j,A_i} \quad (3)$$

By "$\gamma^+_{F_j,A_i} \in NF(A_i)$" we mean that there only is a disjunct for those A_i for which[6] there is some $\gamma^+_{F_j,A_i}$ in the effect of A_i. Using a similar definition for $\gamma^-_{F_j}$, we get the successor state axiom for F_j:

$$\Box[a]F_j(\vec{x_j}) \equiv \gamma^+_{F_j} \wedge \vec{\tau_{F_j}}(\vec{x_j})\ \vee\ F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j} \quad (4)$$

The additional conjunct[7] $\vec{\tau_{F_j}}(\vec{x_j})$ ensures that F_j can only become true for instantiations of the $\vec{x_j}$ that are consistent with the type definitions for F_j's arguments.

[5] $\vec{y}{:}\vec{\tau}$ is to be understood as a list of pairs $y_i{:}\tau_i$.

[6] Recall that (2) does not require a conjunct for each F_j and A_i.

[7] $\vec{\tau}(\vec{t})$ abbreviates $\tau_1(t_1) \wedge \cdots \wedge \tau_k(t_k)$.

Since our semantics defines predicates to be fluent and types have to be situation-independent, we additionally include for each type τ_i a successor state axiom

$$\Box[a]\tau_i(x) \equiv \tau_i(x). \tag{5}$$

Applying all of the above to the example, we get

$$\begin{aligned}
\gamma^+_{At} &= \exists y_1.\exists y_2.\ a = \mathit{move}(y_1, y_2) \land \\
&\quad (\exists z_1{:}\mathit{Xcoord}.\ \exists z_2{:}\mathit{Ycoord}. \\
&\qquad \mathit{At}(y_1, z_1, z_2) \land \mathit{Connects}(z_1, z_2, y_2, x_2, x_3) \land \\
&\qquad\quad (x_1 = y_1 \lor \mathit{In}(y_1, x_1))), \\
\gamma^-_{At} &= \exists y_1.\exists y_2.\ a = \mathit{move}(y_1, y_2) \land \\
&\quad (\exists z_1{:}\mathit{Xcoord}.\ \exists z_2{:}\mathit{Ycoord}. \\
&\qquad \mathit{At}(y_1, z_1, z_2) \land \mathit{Connects}(x_2, x_3, y_2, z_1, z_2) \land \\
&\qquad\quad (x_1 = y_1 \lor \mathit{In}(y_1, x_1))), \\
\gamma^+_{In} &= \exists y_1.\exists y_2.\ a = \mathit{pickup}(y_1, y_2), \\
\gamma^-_{In} &= \exists y_1.\ a = \mathit{drop\_passenger}(y_1) \land \\
&\quad \forall x_1{:}\mathit{Taxi}.\ \forall x_2{:}\mathit{Passenger}.\ (x_1 = y_1 \land \mathit{In}(x_1, x_2))
\end{aligned}$$

Notice that, as stated above, not all operators are mentioned in γ^+_{At}, but only those that possibly cause a positive truth value for *At*. Therefore, the construction presented here still incorporates a solution to the frame problem. Our Σ_{post} now consists of the following sentences (among others[8]):

$$\begin{aligned}
&\Box[a]\mathit{At}(x_1, x_2, x_3) \ \equiv\ \gamma^+_{At} \land \mathit{Object}(x_1) \land \mathit{Xcoord}(x_2) \\
&\quad \land \mathit{Ycoord}(x_3) \ \lor\ \mathit{At}(x_1, x_2, x_3) \land \neg\gamma^-_{At}, \\
&\Box[a]\mathit{In}(x_1, x_2) \ \equiv \\
&\quad \gamma^+_{In} \land \mathit{Taxi}(x_1) \land \mathit{Passenger}(x_2) \lor\ \mathit{In}(x_1, x_2) \land \neg\gamma^-_{In}, \\
&\Box[a]\mathit{Taxi}(x_1) \ \equiv \mathit{Taxi}(x_1), \\
&\Box[a]\mathit{Passenger}(x_1) \ \equiv \mathit{Passenger}(x_1)
\end{aligned}$$

The Precondition Axiom Σ_{pre}: A precondition axiom can be obtained by a similar case distinction; in the example:

$$\begin{aligned}
\pi = \ & \exists y_1{:}\mathit{Taxi}.\exists y_2{:}\mathit{Direction}.\ a = \mathit{move}(y_1, y_2) \lor \\
& \exists y_1{:}\mathit{Taxi}.\exists y_2{:}\mathit{Passenger}.\ a = \mathit{pickup}(y_1, y_2) \\
& \quad \land \exists z_1{:}\mathit{Xcoord}.\exists z_2{:}\mathit{Ycoord}.\ \mathit{At}(y_1, z_1, z_2) \\
& \qquad \land \mathit{At}(y_2, z_1, z_2) \land \neg\exists z_3{:}\mathit{Passenger}.\ \mathit{In}(y_1, z_3) \lor \\
& \exists y_1{:}\mathit{Taxi}.\ a = \mathit{drop\_passenger}(y_1)
\end{aligned} \tag{6}$$

The Initial Description Σ_0: We finally not only have to encode the information about the initial world state, but also about the typing of objects:

$$\tau_i(x) \equiv (\tau_{i_1}(x) \lor \cdots \lor \tau_{i_{k_i}}(x)) \tag{7}$$

$$F_j(x_{j_1}, \ldots, x_{j_{k_j}}) \supset (\tau_{j_1}(x_{j_1}) \land \cdots \land \tau_{j_{k_j}}(x_{j_{k_j}})) \tag{8}$$

$$\tau_i(x) \equiv (x = o_{j_1} \lor \cdots \lor x = o_{j_{k_i}}) \tag{9}$$

$$\mathit{Object}(x) \equiv (\tau_1(x) \lor \cdots \lor \tau_l(x)) \tag{10}$$

Σ_0 contains one sentence (7) for each "either" statement (1). We further have one sentence of the form (8) for each type declaration of predicate arguments. For each primitive type τ_i such that $o_{j_1}, \ldots, o_{j_{k_i}}$ are all objects declared to be of that type, we include a sentence of the form (9). (10) finally declares *Object* to be the union of all other types. Although the above sentences in themselves only establish type consistency in the initial situation (there are no $\Box$ operators here), the special form of Σ_{post} defined earlier ensures that these facts will remain true in successor situations.

To encode the actual initial state, we include for each F_j[9]

$$F_j(\vec{x_j}) \equiv (\vec{x_j} = \vec{o_1} \lor \cdots \lor \vec{x_j} = \vec{o_{k_o}}) \tag{11}$$

in case of a closed world, assuming that $F_j(\vec{o_1}), \ldots, F_j(\vec{o_{k_o}})$ are all atoms in I mentioning F_j. In an open-world problem, we instead include

$$\begin{aligned}
&(\vec{x_j} = \vec{o_1} \lor \cdots \lor \vec{x_j} = \vec{o_{k_o}}) \supset F_j(\vec{x_j}) \land \\
&F_j(\vec{x_j}) \supset \neg(\vec{x_j} = \vec{p_1} \lor \cdots \lor \vec{x_j} = \vec{p_{k_p}})
\end{aligned} \tag{12}$$

if the $\vec{o_i}$ are all the objects in positive literals $F_j(\vec{o_i})$ and the $\vec{p_i}$ are all the objects in negative literals $\neg F_j(\vec{p_i})$ in I. In our closed-world example, we end up with a Σ_0 consisting of:

$$\begin{aligned}
&\mathit{At}(x_1, x_2, x_3) && \supset (\mathit{Object}(x_1) \land \mathit{Xcoord}(x_2) \land \mathit{Ycoord}(x_3)), \\
&\mathit{In}(x_1, x_2) && \supset (\mathit{Taxi}(x_1) \land \mathit{Passenger}(x_2)), \\
&\mathit{Taxi}(x) && \equiv (x = \mathit{taxi1}), \\
&\mathit{Passenger}(x) && \equiv (x = \mathit{passenger1}), \\
&\mathit{Object}(x) && \equiv (\mathit{Taxi}(x) \lor \cdots \lor \mathit{Passenger}(x)), \\
&\mathit{At}(x_1, x_2, x_3) && \equiv ((x_1 = \mathit{taxi1} \land x_2 = 5 \land x_3 = 7) \lor \\
& && \quad (x_1 = \mathit{passenger1} \land x_2 = 3 \land x_3 = 1)), \\
&\mathit{In}(x_1) && \equiv \text{FALSE}
\end{aligned}$$

Correctness: First we note some simple consequences of the above construction:

Lemma 1 *Let $A(\vec{p})$ be an action (i.e. an operator and object symbols as instantiations for A's parameters) and $\vec{o}$ be object parameters (i.e. constants) for the fluent F_j. Then*

1. *$\Sigma_0 \land \tau_{\vec{F_j}}(\vec{o})$ is satisfiable iff the $\vec{o}$ are of the correct types (according to the ADL problem description).*
2. *$\Sigma_0 \models \gamma^{*}{}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{A(\vec{p})}$ iff $\gamma^*_{F_j,A}(\vec{o}, \vec{p})$ is satisfied in the original ADL state I, where $* \in \{+, -\}$.*
3. *$\Sigma_0 \models \mathit{Poss}(A(\vec{p}))$ iff $\pi_A(\vec{p})$ is satisfied in the original ADL state I and the $\vec{p}$ are of the correct types.*

We are now ready to show the correspondence between ADL's state-transition semantics of adding and deleting literals and first-order progression in $\mathcal{ES}$. The definition below is derived from Lin and Reiter's original proposal of progression, but it is simpler due to the fact that we need not consider arbitrary first-order structures.

A set of sentences Σ_r is a *progression* of Σ_0 through a ground term r (wrt Σ_{pre} and Σ_{post}) iff:

1. all sentences in Σ_r are fluent in $\langle r \rangle$
   (i.e. equivalent to $[r]\phi$ for some fluent formula ϕ);
2. $\Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \models \Sigma_r$;
3. for every world w_r with $w_r \models \Sigma_r \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, there is a world w with $w \models \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$ such that:
   $$w_r, r \cdot \sigma \models F(\vec{t}) \quad \text{iff} \quad w, r \cdot \sigma \models F(\vec{t})$$
   for all $\sigma \in R^*$ and all primitive formulas $F(\vec{t})$ such that $F \in \mathcal{F}$ (including *Poss*).

[8]Here and in the following we omit axioms concerning the non-changing predicates *Connects* and *Destination* as well as the types *Xcoord*, *Ycoord* and *Direction* which encode the grid structure.

[9]If $\vec{r}$ abbreviates $r_1, \ldots, r_k$ and $\vec{t}$ stands for $t_1, \ldots, t_k$, then $(\vec{r} = \vec{t})$ means $(r_1 = t_1) \land \cdots \land (r_k = t_k)$.

Intuitively, for an observer in the situation after r was performed (and only looking "forward" in time), it is impossible to distinguish between a world satisfying the original theory Σ and one that satisfies $\Sigma_r \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$.

For a basic action theory that is the translation from an ADL problem, it is quite easy to obtain such a progression given an action $A(\vec{p})$ and an ADL state description I. Provided that $\Sigma_0 \cup \Sigma_{\text{pre}} \models \mathit{Poss}(A(\vec{p}))$, we do the following for all (finitely many) $\vec{o}$ and F_j such that $F_j(\vec{o})$ is type-consistent:[10]

1. If $\Sigma_0 \models \gamma^{+}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{A(\vec{p})}$: add $F_j(\vec{o})$.
2. If $\Sigma_0 \models \gamma^{-}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{A(\vec{p})}$: delete $F_j(\vec{o})$.

For open worlds, additionally delete $\neg F_j(\vec{o})$ in the first case and add $\neg F_j(\vec{o})$ in the second case. If we then denote the set of literals to be added by $Adds$ and the ones to be deleted by $Dels$, the new state description is

$$I' = (I \setminus Dels) \cup Adds.$$

In the closed-world case, I' corresponds precisely to the ADL state which results from applying the $A(\vec{p})$ to I according to the state-transition semantics of [Fox and Long, 2003].

Theorem 2 *Let I' be obtained as described above, given an ADL problem and a ground action $r = A(\vec{p})$. Further let $\Sigma_r = \{[r]\psi \mid \psi \in \Sigma_0(I')\}$, where $\Sigma_0(I')$ is the result of applying the constructions in (7)-(12) to I' instead of I. For all F_j, let the consistency condition $\models \neg(\gamma^{+}_{F_j} \wedge \gamma^{-}_{F_j})^{a}_{r}$ hold. Then Σ_r is a progression of Σ_0 through r in the closed-world case.*

In the open-world case, this holds under the additional condition that whenever for some $\gamma^{*}_{F_j}$ (with $* \in \{+,-\}$) it is the case that $\Sigma_0 \cup \{\gamma^{*}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{r}\}$ is satisfiable, then $\Sigma_0 \models \gamma^{*}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{r}$.

Let us consider progression in our closed-world example. Suppose we want to progress through $\mathit{move}(\mathit{taxi1}, \mathit{south})$ (m, for short). First note that $\Sigma_0 \cup \Sigma_{\text{pre}} \models \mathit{Poss}(m)$. The reader may verify (assuming an appropriate axiomatization of *Connects*) that

$$\Sigma_0 \models \gamma^{+}_{\mathit{At}}{}^{a}_{m}{}^{x_1}_{\mathit{taxi1}}{}^{x_2}_{5}{}^{x_3}_{8}, \quad \Sigma_0 \models \gamma^{-}_{\mathit{At}}{}^{a}_{m}{}^{x_1}_{\mathit{taxi1}}{}^{x_2}_{5}{}^{x_3}_{7}$$

and that these are all type-consistent instantiations for x_1, x_2, x_3 such that $\gamma^{+}_{\mathit{At}}{}^{a}_{m}$ respectively $\gamma^{-}_{\mathit{At}}{}^{a}_{m}$ are entailed by Σ_0. Because there are no disjuncts for *move* in γ^{+}_{In} and γ^{-}_{In}, $\gamma^{+}_{\mathit{In}}{}^{a}_{m}$ and $\gamma^{-}_{\mathit{In}}{}^{a}_{m}$ are not entailed for any instantiation of x_1, x_2. The new state description then is

$$\begin{aligned} I' = \{&\mathit{At}(\mathit{taxi1}, 5, 8), \mathit{At}(\mathit{passenger1}, 3, 1), \\ &\mathit{Destination}(\mathit{passenger1}, 1, 1), \\ &\mathit{Connects}(1, 1, \mathit{east}, 2, 1), \ldots \\ &\mathit{Connects}(10, 10, \mathit{north}, 10, 9)\} \end{aligned}$$

We obtain the progression Σ_m consisting of

$$\begin{aligned} &[m](\mathit{At}(x_1, x_2, x_3) \supset (\mathit{Object}(x_1) \wedge \mathit{Xcoord}(x_2) \\ &\qquad\qquad\qquad\qquad \wedge \mathit{Ycoord}(x_3))), \\ &[m](\mathit{In}(x_1, x_2) \supset (\mathit{Taxi}(x_1) \wedge \mathit{Passenger}(x_2))), \\ &[m](\mathit{Taxi}(x) \equiv (x = \mathit{taxi1})), \\ &[m](\mathit{Passenger}(x) \equiv (x = \mathit{passenger1})), \\ &[m](\mathit{Object}(x) \equiv (\mathit{Taxi}(x) \vee \cdots \vee \mathit{Passenger}(x))), \\ &[m](\mathit{At}(x_1, x_2, x_3) \equiv ((x_1 = \mathit{taxi1} \wedge x_2 = 5 \wedge x_3 = 8) \\ &\qquad \vee (x_1 = \mathit{passenger1} \wedge x_2 = 3 \wedge x_3 = 1))), \\ &[m](\mathit{In}(x_1) \equiv \text{FALSE}) \end{aligned}$$

[10]i.e. those $F_j(\vec{o})$ such that $\Sigma_0 \wedge \tau_{\vec{F_j}}(\vec{o})$ is satisfiable

Theorem 2 tells us that, for a closed initial state, the application of an ADL operator under the state-transition semantics is the same as progressing the initial situation of the corresponding BAT. In the open-world case, it prescribes what the result of applying the ADL operator should be. It is easy to show that the theorem extends to arbitrary sequences of actions and hence that legal plans under ADL are also legal plans in the BAT. [11]

5 Golog with FF

Golog is a programming language completely defined within the situation calculus. As shown in [Lakemeyer and Levesque, 2005], it can be as well defined directly within $\mathcal{ES}$. At its core, Golog uses a basic action theory to define the meaning of its (primitive) actions and the initial situation. Based on these primitives, complex actions can be formed using constructs from imperative programming such as *sequence* (;), *if-then-else*, *while-loops* and *procedures*. In addition, non-deterministic action choices and the non-deterministic choice of arguments (πx) offer some flexibility when executing a program, including full planning over the primitive actions.

For our experimental results, we are using an Eclipse Prolog implementation of IndiGolog, courtesy of Hector Levesque; see [Sardina *et al.*, 2004] for details of the semantics. IndiGolog features on-line execution, where a program is executed step-by-step, and progression in the sense of the previous section, that is, the initial state is updated after each execution of a primitive action. IndiGolog also includes an implementation of Reiter's forward-search planner (specified as a Golog procedure), which can be called as **achieve**(G) for a given goal formula G, and which returns as a plan a sequence of actions P, if one is found. P then takes the place of **achieve**(G) in the program, and step-wise execution resumes, starting with the execution of P.

The underlying BAT is the translation of the ADL problem description for the taxi domain. The following program is used throughout our tests:

```
while $\exists p{:}\mathit{Passenger}.\ \neg \mathit{atDest}(p)$ do
  $(\pi p{:}\mathit{Passenger})$
    $\neg \mathit{atDest}(p)?$; achieve($\mathit{atSamePos}(\mathit{taxi1}, p)$);
    $\mathit{pickup}(\mathit{taxi1}, p)$; achieve($\mathit{atDest}(p)$);
    $\mathit{drop\_passenger}(\mathit{taxi1})$
endWhile
```

Here, $\mathit{atDest}(p)$ stands for

$$\exists x{:}\mathit{Xcoord}.\ \exists y{:}\mathit{Ycoord}.\ \mathit{At}(p, x, y) \wedge \mathit{Destination}(p, x, y)$$

and $\mathit{atSamePos}(t, p)$ abbreviates

$$\exists x{:}\mathit{Xcoord}.\ \exists y{:}\mathit{Ycoord}.\ \mathit{At}(p, x, y) \wedge \mathit{At}(t, x, y).$$

The following table summarizes, for varying grid sizes and number of passengers, the run-time behavior (in seconds) of the original IndiGolog versus IndiGolog where **achieve** is replaced by a call to the ADL planner FF [Hoffmann and

[11]Legal means that all actions are executable and that the goal is satisfied in the final state or progressed situation.

Nebel, 2001]; according to Theorem 2, the legal plans in both cases are the same. Each test was run on four instances (with a random distribution of passengers and destinations), and the average run-time is given. Experiments were carried out on a Pentium M with 1.8GHz and 1GByte of main memory.

Prob. size	Golog + FF	Golog + **achieve**
3x3, 1 Pass.	6.50	6.52
3x3, 5 Pass.	8.83	8.21
3x3, 10 Pass.	12.87	9.99
4x4, 1 Pass.	7.25	11.75
4x4, 5 Pass.	10.37	38.33
4x4, 10 Pass.	16.97	—
7x7, 1 Pass.	7.55	—
7x7, 5 Pass.	152.79	—
7x7, 10 Pass.	—	—

The table clearly shows that the built-in Golog planner cannot compete for all but the smallest problem instances. (Missing entries are tests which did not return a result after 10 minutes and were aborted.)

We want to emphasize that our approach does not aim at competing with existing planning systems on solving classical planning problems.[12] Instead, we envision that the combined system pays off in scenarios where each approach by itself would be either inefficient or fail completely. In particular, this is the case when the problem is very large and information has to be gathered at run-time. To illustrate the idea, instead of the toy example above, consider a robot constantly transporting items in a warehouse: The robot would execute thousands of actions, including sensing its environment and servicing new transportation requests. While it does not seem feasible to solve the whole problem by an offline planner, it is certainly possible to write an IndiGolog program to solve such a task. The role of the planner is then to solve certain sequential subtasks (e.g. finding an optimal schedule for currently pending requests) at which Golog by itself would be too slow. We believe that the results in this paper lay the necessary ground work for such scenarios.

6 Conclusion

In this paper we showed that it pays off to combine the action language Golog with modern planners. In doing so, we developed a declarative semantics for ADL as progression in the situation calculus. Currently we are also considering larger fragments of PDDL [Fox and Long, 2003] with features such as time and preferences. It would be interesting to also consider other action formalisms like Flux [Thielscher, 2002] or the family of $\mathcal{A}$ languages [Gelfond and Lifschitz, 1998]. Our hope is that this will lead to a convergence of the planning and action-language communities, which have been largely separated since the inception of STRIPS.

Acknowledgments

This work has partly been supported by the EC under contract number FP6-004250-CoSy and DFG grants La 747/13-2 and Ne 623/10-1.

[12]In additional experiments we gave FF the overall task of finding a plan to deliver all passengers to their destinations. It was usually faster than the combination of Golog and FF; e.g. finding a plan for the 7x7 grid with 5 passengers took 29.74 seconds.

References

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Heuristic Search Planner 2.0. *AI Magazine*, 22(3):77–80, 2001.

[Burgard *et al.*, 1999] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artif. Intell.*, 114(1-2):3–55, 1999.

[Eyerich *et al.*, 2006] P. Eyerich, B. Nebel, G. Lakemeyer, and J. Claßen. Golog and PDDL: What is the relative expressiveness? submitted for publication, 2006.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[Gelfond and Lifschitz, 1998] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3, 1998.

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

[Lakemeyer and Levesque, 2004] G. Lakemeyer and H. Levesque. Situations, si! situation terms, no! In *Proc. KR-04*. AAAI Press, 2004.

[Lakemeyer and Levesque, 2005] G. Lakemeyer and H. Levesque. Semantics for a useful fragment of the situation calculus. In *Proc. IJCAI-05*, 2005.

[Levesque and Lakemeyer, 2001] H. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, 2001.

[Levesque *et al.*, 1997] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. Log. Prog.*, 31:59–84, 1997.

[Lifschitz, 1986] V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proc. of the 1986 Workshop*, pages 1–9. Morgan Kaufmann, 1986.

[Lin and Reiter, 1997] F. Lin and R. Reiter. How to progress a database. *Artif. Intell.*, 92(1-2):131–167, 1997.

[Pednault, 1989] E. Pednault. ADL: Exploring the middle ground between STRIPS and the Situation Calculus. In *Proc. KR-89*, pages 324–332, 1989.

[Pednault, 1994] E. Pednault. ADL and the state-transition model of action. *J. Log. Comput.*, 4(5):467–512, 1994.

[Reiter, 2001] R. Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press, Cambridge, Mass., 2001.

[Sardina *et al.*, 2004] S. Sardina, G. De Giacomo, Y. Lespérance, and H. Levesque. On the semantics of deliberation in IndiGolog: From theory to implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2–4):259–299, 2004.

[Thielscher, 2002] M. Thielscher. Programming of reasoning and planning agents with FLUX. In *Proc. KR-02*. AAAI Press, 2002.