A Situation-Calculus Semantics for an Expressive Fragment of PDDL

Jens Claßen
Dept. of Computer Science
RWTH Aachen University
52056 Aachen
Germany
classen@cs.rwth-aachen.de

Yuxiao Hu
Dept. of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 3G4
yuxiao@cs.toronto.edu

Gerhard Lakemeyer
Dept. of Computer Science
RWTH Aachen University
52056 Aachen
Germany
gerhard@cs.rwth-aachen.de

Abstract

The Planning Domain Definition Language (PDDL) has become a common language to specify planning problems, facilitating the formulation of benchmarks and a direct comparison of planners. Over the years PDDL has been extended beyond STRIPS and ADL in various directions, for example, by adding time and concurrent actions. The current semantics of PDDL is purely meta-theoretic and quite complex, which makes an analysis difficult. Moreover, relating the language to other action formalisms is also nontrivial. We propose an alternative semantics for an expressive fragment of PDDL within the situation calculus. This yields at least two advantages. For one, the new semantics is purely declarative, making it amenable to an analysis in terms of logical entailments. For another, it facilitates the comparison with and mapping to other formalisms that are defined on top of the same logic, such as the agent control language Golog. In particular we obtain the semantical foundation for embedding efficient PDDL-based planners into the more expressive, yet computationally expensive Golog, thus combining the benefits of both. Other by-products of our investigations are a simpler account of durative actions in the situation calculus and a new notion of compulsory actions.

Introduction

The Planning Domain Definition Language (PDDL), introduced in (Ghallab *et al.* 1998), has become a quasi standard for the formulation of planning domains and problems. It is used to define benchmarks for the empirical evaluation and comparison of planning systems, such as those used at the International Planning Competitions. During the last decade, the language has been extended by numerous features beyond what can be expressed by STRIPS and ADL (Pednault 1989). Among the most important steps in this development are the introduction of numerics, durative actions and concurrency in version 2.1 (Fox & Long 2003), the (re-)integration of derived predicates and timed initial literals in PDDL 2.2 (Edelkamp & Hoffmann 2004) and the extension with quantitative preferences and trajectory constraints in PDDL 3.0 (Gerevini & Long 2005). Today, many fast planners exist that support all or at least some of these features, e.g. (Hsu *et al.* 2006).

The first formal semantics was provided by Fox and Long for PDDL 2.1. They extend Lifschitz' (1986) state-transitional semantics for STRIPS to cope with numerics, time, durative actions and concurrency. All later versions of PDDL use that semantics for defining the meaning of their newly introduced features.

Although that semantics has served its purpose, we see a major drawback in its purely meta-theoretic and complex definition. As an example, for assuring that in a given plan an invariant condition is not violated during the duration interval of an action, Fox and Long insert dummy actions between each two happenings (i.e. simple actions) in that interval. Those actions then take the invariant as their precondition. Similarly, an action with a conditional effect $(\psi \Rightarrow \varphi)$ is split up into two, where one has $\neg\psi$ as an additional precondition and the other requires ψ and has the additional effect φ. Apart from the fact that these kinds of reductions would mean an exponential blowup in a 1-to-1 implementation (Nebel 2000), the complexity of its 19-page definition makes the semantics also difficult to grasp. Further, the analysis and comparison to other action formalisms becomes thus difficult and tedious. In this paper we propose an alternative semantics for PDDL (or its temporal fragment, to be more precise). The cardinal difference is that ours is a *declarative* one, meaning that we define it by means of entailments of theories formulated in a certain logic whose properties are well understood.

In (Claßen *et al.* 2007), the first step in this direction was taken, where it was shown that the state updates in the ADL fragment of PDDL can be understood as progression steps for a certain class of situation calculus (Reiter 2001) action theories, extending work by Lin and Reiter (1997) who did the same for STRIPS. Claßen et al. used this result as the semantical basis for embedding efficient PDDL-based planners into an interpreter of the more expressive, yet computationally more demanding agent control language Golog (which is based on the situation calculus). They provided experimental results that supported that this approach is beneficial in terms of the system's computation time. In this paper, we will extend their work by providing a mapping of the temporal fragment of PDDL to the situation calculus, including numerics, durative actions with discrete and continuous effects and timed initial literals. In the process we obtain a simpler account of durative actions in the situation

calculus and a new notion of compulsory (natural) actions.

We proceed as follows. In the next section we introduce the logic $\mathcal{ES}$, a fragment of the situation calculus well suited for our approach. The next section shows how the usual $\mathcal{ES}$ theories can be extended to cope with numerics, durative actions, concurrency and coercive actions. After that we review the temporal fragment of PDDL before presenting the actual mapping to an $\mathcal{ES}$ action theory. Then we conclude.

The Logic $\mathcal{ES}$

$\mathcal{ES}$ was introduced by Lakemeyer and Levesque (2004; 2005) as an alternative logic for reasoning about an agent's knowledge, action and sensing. It captures precisely the non-epistemic fragment of the situation calculus and Golog. Due to its special syntax and semantics, using the logic in this paper significantly simplifies the translation from PDDL and the accompanying semantical proofs.

The language is a first-order modal dialect with equality and sorts of type *object*, *action* and *number*. It includes countably infinitely many standard names for each of the sorts, allowing for a substitutional interpretation of quantification. Also included are both fluent and rigid predicate and function symbols[1]. Fluents vary as the result of actions, but rigids do not. The logical connectives are $\wedge$, $\neg$, $\forall$, together with the modal operators $\Box$ and $[r]$ where r may be any term of sort action, including a variable. Other connectives like $\vee$, $\supset$, $\subset$, $\equiv$, and $\exists$ are used as the usual abbreviations.

Terms and formulas are built from these primitives in the usual way. We read $[r]\alpha$ as "α holds immediately 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 predicates *Poss*. In addition, we introduce the following special notation: a *type* τ is a rigid unary predicate; we write $\forall x{:}\tau.\,\phi$ instead of $\forall x.\,\tau(x) \supset \phi$; we further extend this definition to tuples of variables $\vec{x}$ and types $\vec{\tau}$ in the obvious way.

The Semantics

Intuitively, a world w will determine whether or not a sentence α is true after a sequence of actions z (we then write $w, z \models \alpha$.). It does so by assigning truth values to the primitive sentences and co-referring standard names to the primitive terms, given z. By a primitive sentence (term) we mean an expression of the form $h(n_1, \ldots, n_k)$, where h is a rigid or fluent predicate (function) symbol and all the n_i are standard names. More precisely, let $\mathcal{N}$ denote the set of all standard names and $\mathcal{Z}$ the set of all finite sequences of standard action names, including $\langle\,\rangle$, the empty sequence. Then a world $w \in W$ is any function from the primitive sentences and $\mathcal{Z}$ to $\{0, 1\}$, and from the primitive terms and $\mathcal{Z}$ to $\mathcal{N}$ (preserving sorts), and satisfying the rigidity constraint: if g is a rigid function or predicate symbol, then for all z and z' in $\mathcal{Z}$, $w[g(n_1, \ldots, n_k), z] = w[g(n_1, \ldots, n_k), z']$.

We extend the idea of co-referring standard names to arbitrary ground terms as follows. Given a variable-free term t, a world w, and an action sequence z, we define $|t|^z_w$ (read: the co-referring standard name for t given w and z) by:

1. If $t \in \mathcal{N}$, then $|t|^z_w = t$;
2. $|h(t_1, \ldots, t_k)|^z_w = w[h(n_1, \ldots, n_k), z]$, if $n_i = |t_i|^z_w$.

Here then is the semantic definition of truth. Given a sentence α and $w \in W$, we define $w \models \alpha$ (read: α is true) as $w, \langle\,\rangle \models \alpha$, where for any $z \in \mathcal{Z}$ we have:

$w, z \models h(t_1, \ldots, t_k)$ iff $w[h(n_1, \ldots, n_k), z] = 1$, where $n_i = |t_i|^z_w$;
$w, z \models (t_1 = t_2)$ iff n_1 and n_2 are identical, where $n_i = |t_i|^z_w$;
$w, z \models [t]\alpha$ iff $w, z \cdot n \models \alpha$, where $n = |t|^z_w$;
$w, z \models (\alpha \wedge \beta)$ iff $w, z \models \alpha$ and $w, z \models \beta$;
$w, z \models \neg\alpha$ iff $w, z \not\models \alpha$;
$w, z \models \forall x.\,\alpha$ iff $w, z \models \alpha^x_n$, for every std. name n of the same sort as x;
$w, z \models \Box\alpha$ iff $w, z \cdot z' \models \alpha$, for every $z' \in \mathcal{Z}$.

The notation $\alpha^{t_1}_{t_2}$ means the simultaneous substitution of all t_1 by t_2 in α. 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 $w \models \alpha$. Finally, we write $\models \alpha$ (read: α is valid) to mean $\{\} \models \alpha$.

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

$$\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}, \tag{1}$$

where Σ_0, the initial database, is a finite set of fluent sentences and Σ_{pre} is a precondition axiom of the form[2]

$$\Box \mathit{Poss}(a) \equiv \pi, \tag{2}$$

with π being a fluent formula whose only free variable is a. Σ_{post} is a finite set of successor state axioms (SSAs)[3]

$$\Box[a]F(\vec{x}) \equiv \gamma_F, \tag{3}$$

$$\Box[a]f(\vec{x}) = y \equiv \gamma_f \tag{4}$$

for each relational fluent $F \in \mathcal{F} \setminus \{\mathit{Poss}\}$ and each functional fluent $f \in \mathcal{F}$, incorporating Reiter's (2001) solution to the frame problem. γ_F has to be a fluent formula with free variables $\vec{x}$, and γ_f one with free variables among $\vec{x}$ and y.

Extensions to $\mathcal{ES}$

In this section we will show how the standard definition of basic action theories of the previous section has to be extended to deal with numerics, durative and coercive actions.

[1] For simplicity we however consider only rigid action symbols.

[2] 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 $\forall a.\Box(\mathit{Poss}(a) \equiv \pi)$.

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

Numerics and Time

Without going into much detail about numerics here, suffice it to say that we use the first-order subset of an axiomatization of the reals (Tarski 1951) that has models with countable domains, thus being also satisfiable in $\mathcal{ES}$. For the restricted usage of arithmetic in this paper, the resulting models are completely sufficient. We denote these axioms by Σ_{num}.

Now let us turn to the question of how to represent the flow of time in $\mathcal{ES}$. What is presented below is based on work by Pinto (1994) and Reiter (1996).

The idea is to extend each action term by an additional numeric argument denoting the happening time of that action; the action *pickup*(x) thus turns into *pickup*(x,t). For being able to refer to the happening time of arbitrary actions, we introduce a new fluent $time(a)$ defined by the axiom

$$\Box time(a) = t \equiv \bigvee_i \exists \vec{x_i}.a = A_i(\vec{x_i}, t), \tag{5}$$

where the A_i are all the action symbols of the application domain. In the above example, this will entail that $time(\textit{pickup}(block, 3)) = 3$.

Next, we use a functional fluent now of sort number whose value always represents the happening time of the last action (i.e. the time of the "current" situation):

$$\Box[a] now = time(a) \tag{6}$$

We fix now's initial value by letting Σ_0 contain $now = 0$ and assert[4] that actions only happen chronologically:

$$\Box \textit{Poss}(a) \supset now \leq time(a) \tag{7}$$

Concurrent Durative Actions Further adapting Pinto and Reiter's idea, we represent processes with a duration by a corresponding start and end happening. Unlike them[5], we only need two additional action symbols $start(a', t)$ and $end(a', t)$ taking an action term a' as first and the corresponding happening time as second argument. The fluent $Performing(a')$ then denotes whether the durative action a' is currently in progress. Its SSA is

$$\begin{aligned}\Box[a] Performing(a') \equiv \exists t.a = start(a', t) \vee \\ Performing(a') \wedge \neg\exists t'.a = end(a', t')\end{aligned} \tag{8}$$

If c is $chew(gum)$, this entails $[start(c, 3)]Performing(c)$ and $[start(c, 3)][end(c, 5)]\neg Performing(c)$.

Additionally, a start event is only allowed when the corresponding action is currently not running; similarly the end event is only allowed if that action is currently in progress:

$$\Box \textit{Poss}(start(a', t)) \supset \neg Performing(a') \tag{9}$$

$$\Box \textit{Poss}(end(a', t)) \supset Performing(a') \tag{10}$$

Sometimes we have to refer to the starting time of a running process. The fluent $since$ "records" this value:

$$\begin{aligned}\Box[a] since(a') = t \equiv a = start(a', t) \vee \\ since(a') = t \wedge \neg\exists t'.a = end(a', t')\end{aligned} \tag{11}$$

[4]We will state more assertions of the form $\Box \textit{Poss}(a) \supset \psi_i$ in the following. We get a precondition axiom of the form (2) by making a completeness assumption, yielding $\Box \textit{Poss}(a) \equiv \bigwedge_i \psi_i$.

[5]They introduce a new fluent and new action symbols for each action type (e.g. $Chewing(x, s)$, $startChew(x, t)$ and $endChew(x, t)$) yielding correspondingly many SSAs. Our approach allows a more compact representation.

Continuous Change So far, our extensions to $\mathcal{ES}$ only allow for discrete changes applied at the (start or end) happening times of actions. This is however not sufficient for modelling continuous change, for instance when the $drive$ action of an electric car constantly drains the car's power. Our target language PDDL on the other hand is expressive enough for formulating such facts (but restricted to linear changes over time). We therefore include continuous changes by following the approach of Grosskreutz and Lakemeyer (2000). The main idea is that we do not let fluents take on numerical values directly, but instead we assign them terms of the form $linear(x, v, t)$. Intuitively, when $power(car) = linear(5, -1, 3)$ holds in the current situation then it means that the car's power level is linearly decreasing by 1 per time unit, starting at an initial value of 5 at time 3.

We define appropriate axioms (left out for space reasons) for $+$ and $=$ wrt $linear$ functions and constant values. We further identify constant numerical values c with $linear(c, 0, t)$. Next, we need a way to obtain the value of a continuously changing fluent at a given time point. We can extract this information from the $linear$ term by defining

$$\begin{aligned}&\Box eval(x, t) = y \equiv \\ &\exists x', v', t'.x = linear(x', v', t') \wedge y = x' + v'(t - t') \vee \\ &\forall x', v', t'.x \neq linear(x', v', t') \wedge y = x\end{aligned} \tag{12}$$

Let then $Eval[\phi, t]$ be an operation that replaces all terms r in a formula ϕ by $eval(r, t)$. We can thus evaluate a formula containing continuously changing fluents against a specified time point t. In a domain with such fluents, instead of (2), the precondition axioms needs then to take the form

$$\Box \textit{Poss}(a) \equiv Eval[\pi, time(a)]. \tag{13}$$

The reason is that when, for example, $power(car) > 0$ is a precondition of an action $drive(car)$, we have to ensure that $eval(power(car), time(drive(car))) > 0$, i.e. the car has to have power *at the time we start to drive* for the action to be possible.

Let Σ_{time} be all additional axioms for temporal properties.

Obligatory Actions

Our notion of coerciveness includes both the one forced by natural laws as well as predetermined exogenous actions. For that matter, we introduce another special fluent $Obli$, for which we have an axiom Σ_{obli} similar to the one for *Poss*:

$$\Box Obli(a) \equiv \Omega \tag{14}$$

where Ω is a fluent formula that describes all necessary and sufficient conditions under which a is an action that must occur in the current situation. For example

$$\begin{aligned}\Box Obli(a) \equiv \exists t.a = boil(t) \wedge Performing(heat) \\ \wedge t - since(heat) > 5 \vee a = closeShop(8pm)\end{aligned}$$

states that a pot of water heating more than five minutes will boil and the shop closes at 8 p.m.

The execution of coercive actions is enforced by

$$\begin{aligned}&\Box \textit{Poss}(a) \supset Obli(a) \vee \\ &\neg(\exists a'.Obli(a') \wedge now \leq time(a') \leq time(a))\end{aligned} \tag{15}$$

which intuitively says that a is not allowed to happen when there are currently pending coercive actions *unless* a itself is an obligatory action that has to happen now. The latter condition is because we do not want to have two obligatory actions scheduled for the same time blocking one another.

Executability

Finally we define a fluent $Executable$ which will hold in all situations reachable by valid (according to *Poss*) actions. For this purpose let Σ_0 contain (16) and Σ_{post} contain (17):

$$Executable \equiv \text{TRUE} \tag{16}$$

$$\Box[a]Executable \equiv Executable \wedge Poss(a). \tag{17}$$

With these extensions, the complete action theory now is

$$\Sigma = \Sigma_{\text{num}} \cup \Sigma_{\text{time}} \cup \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \cup \Sigma_{\text{obli}}. \tag{18}$$

Temporal PDDL

Instead of using the somewhat awkward LISP-based PDDL syntax directly, we will resort to the following more logic-like representation. It can easily be verified (Gerevini & Long 2005) that it corresponds exactly to the fragment of PDDL we obtain by only allowing the requirement flags `:adl`, `:fluents`, `:durative-actions` and `:timed-initial-literals`. We therefore assume that a PDDL problem instance consists of these parts:

1. a finite list of types $\tau_1, \ldots, \tau_l$;
2. finitely many fluent predicates F_j with types $\vec{\tau_j}$ associated to their arguments;
3. finitely many numeric functions f_j with types $\vec{\tau_j}$ associated to their arguments;
4. finitely many object standard names with associated types $o_1{:}\tau_{o_1}, \ldots, o_k{:}\tau_{o_k}$;
5. finitely many operators (described below);
6. a finite list of timed initial literals $\langle t_1, L_1\rangle, \ldots \langle t_r, L_r\rangle$, where each t_i is a number and each L_i is a relational atom or the negation of a relational atom;
7. the initial state description, consisting of a finite collection of functional and relational atoms, for which the closed-world assumption is made;
8. a goal description ψ in form of a precondition formula.

The non-logical symbols appearing in operators, timed initial literals, initial state and goal description have to be those from items 1-4. A relational atom $F_j(\vec{o})$ is a primitive sentence, a functional one has the form $f_j(\vec{o}) = c$, where $f_j(\vec{o})$ is a primitive term of sort number and c a number.

Precondition formulas are the following: A formula $F(\vec{t})$ and every equality atom $(t_1 = t_2)$, where each of the t_i is either a variable or an object constant, is a precondition formula. Further, a comparison of the form $exp_1\, op\, exp_2$ is a precondition formula, when *op* is one of $<$,$=$,$>$ and exp_1 and exp_2 are arithmetic expressions built from operators $+$, $-$, $\times$, $/$, numeric literals and terms $f(\vec{t})$ where each t_i is an object constant or a variable. If ϕ_1 and ϕ_2 are precondition formulas, then so are $\phi_1 \wedge \phi_2$, $\neg\phi_1$ and $\forall x{:}\tau.\phi_1$.

We distinguish simple from durative actions. A *simple action* is given by a triple $A = \langle \vec{z}{:}\vec{\tau}, \pi_A, \epsilon_A\rangle$, where $\vec{z}$ are A's arguments with associated types $\vec{\tau}$, π_A is its precondition (a precondition formula) and ϵ_A its effect, the latter being a conjunction of conditional effects of the forms

$$\begin{array}{lll} \forall \vec{x_j}{:}\vec{\tau_j}.\ \gamma^{+}_{F_j,A}(\vec{x_j}, \vec{z}) & \Rightarrow & F_j(\vec{x_j}), \\ \forall \vec{x_j}{:}\vec{\tau_j}.\ \gamma^{-}_{F_j,A}(\vec{x_j}, \vec{z}) & \Rightarrow & \neg F_j(\vec{x_j}), \\ \forall \vec{x_j}{:}\vec{\tau_j}.\ \gamma^{v}_{f_j,A}(\vec{x_j}, y_j, \vec{z}) & \Rightarrow & f_j(\vec{x_j}) = y_j. \end{array} \tag{19}$$

The meaning of $\gamma \Rightarrow \psi$ is that when γ holds before doing action A, ψ will hold afterwards. γ has to be a precondition formula in each of the cases. Without loss of generality we can assume that there is at most one single effect of each form for any given F_j or f_j in ϵ_A. Further $\gamma^{v}_{f_j,A}$ is required to ensure a unique value for the number variable y_j given appropriate instances for $\vec{x_j}$ and $\vec{z}$.

A *durative action* $A = \langle \vec{z}{:}\vec{\tau}, \delta_A, \pi_A, \epsilon_A\rangle$ now is given by

- $\delta_A = \langle \delta^s_A, \delta^e_A\rangle$, the start and end duration constraints, each of which a conjunction of expressions of the form $duration\ op\ expr$, where $op \in \{\leq, \geq, =\}$ and $expr$ is a numerical expression constructed from numbers, functional fluents, $+$ and $\times$. δ^s_A relates the special symbol $duration$ to the values of numeric fluents when *starting* the action and δ^e_A to their values when *ending* A.
- $\pi_A = \langle \pi^s_A, \pi^o_A, \pi^e_A\rangle$, the start, overall and end conditions of A, each one a precondition formula. The intended meaning is that π^s_A has to hold at the starting time of the action, π^o_A during the open interval between start and end and π^e_A at the ending time.
- $\epsilon_A = \langle \epsilon^s_A, \epsilon^o_A, \epsilon^e_A\rangle$, where
  - the start effect ϵ^s_A is a conjunction of conditional effects of the form (19) taking place at the starting time of A;
  - the overall effect ϵ^o_A is a conjunction of continuous effects $\langle op, f(\vec{t}), expr\rangle$, where op is $+$ or $-$, f is a numeric fluent, each of the t_i is either one of A's parameters or an object name and exp is a numeric expression; (The intended meaning is that $f(\vec{t})$ is linearly increased ($op = +$) or decreased ($op = -$) over the action's duration by $expr$.)
  - the end effect ϵ^e_A is a conjunction of effects of the form $\forall \vec{x_i}{:}\vec{\tau_i}.\langle \varphi^s_i, \varphi^o_i, \varphi^e_i\rangle \Rightarrow \psi_i$, stating that ψ_i (which is either some $F_j(\vec{x_j})$, some $\neg F_j(\vec{x_j})$ or some $f_j(\vec{x_j}) = y_j$) will be true at the ending time of the action when the precondition formula φ^s_i was true at the starting time, φ^o_i did hold during the open interval between start and end and φ^e_i holds at the ending time.

A PDDL *plan* P then is a finite set of simple action instances $(t_i{:}A_i(\vec{o_i}))$ and durative action instances $(t_i{:}A_i(\vec{o_i})[d_i])$, where A_i is the name of the operator, the o_i are the actual parameters, t_i is the happening of the simple action respectively the starting time of the durative action and d_i is the duration. Let the maximum of all t_i and $t_i + d_i$ be the *ending time* of P. For the details of PDDL's semantics, the reader may consult (Fox & Long 2003).

The Mapping

In the course of this section we will put together the parts of an $\mathcal{ES}$ action theory Σ as in (18), given a PDDL problem as defined in the previous section. We conclude with our main result, a theorem stating the correctness of this mapping.

The Initial State

The encoding of the initial state description is similar to the one in (Claßen *et al.* 2007). In addition to the corresponding axioms for the relational fluents, we let Σ_0 contain a complete description of each numeric function's initial values:

$$f_j(\vec{x_j}) = y_j \equiv \quad \vec{x_j} = \vec{o_{j_1}} \wedge y_j = c_{j_1} \vee \cdots \vee \vec{x_j} = \vec{o_{j_{k_j}}} \wedge y_j = c_{j_{k_j}} \tag{20}$$

where $f_j(\vec{o_{j_1}}) = c_{j_1}, \ldots, f_j(\vec{o_{j_{k_j}}}) = c_{j_{k_j}}$ are all the functional atoms in the initial state mentioning f_j. Note that Σ_0 further contains axioms encoding all typing information, among them for each type τ_j an axiom

$$\tau_j(x) \equiv x = o_1 \vee \cdots \vee x = o_k, \tag{21}$$

where the o_i were all the objects defined to be of type τ_j. This, together with the fact that all quantifiers in the PDDL problem are typed, ensures that PDDL's domain closure is also present in the resulting $\mathcal{ES}$ action theory.

Simple Actions

We handle simple actions again similar to Claßen et al. For each simple A_i, we include the axiom

$$\Box \mathit{Poss}(A_i(\vec{z_i}, t)) \supset \vec{\tau_i}(\vec{z_i}) \wedge \pi_{A_i}. \tag{22}$$

Like those for the relational ones, we additionally construct SSAs for the functional fluents:

$$\gamma^v_{f_j} \stackrel{def}{=} \bigvee_{f_j(\vec{x_j}) \text{ effect in } \epsilon_{A_i}} \exists \vec{z_i}, t.a = A_i(\vec{z_i}, t) \wedge \gamma^v_{f_j, A_i} \tag{23}$$

$$\gamma_{f_j} \stackrel{def}{=} \gamma^v_{f_j} \wedge \tau_{f_j}(\vec{x_j}) \ \vee \ f_j(\vec{x_j}) = y_j \wedge \neg\exists y'.(\gamma^v_{f_j})^{y_j}_{y'} \tag{24}$$

In (23), all the actions' effects that change the value of an f_j are collected. (24) then expresses that the value of f_j will be y_j iff one of the actions changes it to y_j or it has been already equal to y_j before and is not changed by any action. Remember from (19) that y_j is a free variable in $\gamma^v_{f_j, A_i}$.

Durative Actions

For representing a durative PDDL action $A(\vec{z})$ with duration d and starting time t, it seems natural to split it into two happenings $start(A(\vec{z}), t)$ and $end(A(\vec{z}), t + d)$. First consider the case when there are no continuous effects, no inter-temporal effects (i.e. end effects have the form $\forall \vec{x_j}{:}\vec{\tau_j}.\langle \text{TRUE}, \text{TRUE}, \varphi^e_i\rangle \Rightarrow \psi_i$), no invariants and no start duration constraints: we can just treat $start(A(\vec{z}), t)$ like a simple action with precondition π^s_A and effects ϵ^s_A and $end(A(\vec{z}), t)$ like a simple action with precondition π^e_A and effects $\forall \vec{x_j}{:}\vec{\tau_j}.\varphi^e_i \Rightarrow \psi_i$. The end duration constraint can be enforced by making it a precondition of $end(A(\vec{z}), t))$:

$$\Box \mathit{Poss}(end(A(\vec{z}), t)) \ \supset \ (\delta^e_A)^{duration}_{t-since(A(\vec{z}))} \tag{25}$$

Invariant Conditions To protect the invariant condition $\pi^o_{A_j}$ during the open duration interval of A_j, we disallow any action a that would violate it. A violation happens when $\pi^o_{A_j}$ does not hold *after* doing a (and A_j is still running):

$$\Box \mathit{Poss}(a) \supset \bigwedge_j \mathcal{R}[a, Performing(A_j(\vec{z_j})) \supset \pi^o_{A_j}] \tag{26}$$

Here, $\mathcal{R}[a, \phi]$ denotes the regression (Lakemeyer & Levesque 2004) of ϕ through a: $\mathcal{R}[a, \phi]$ is entailed by Σ iff $[a]\phi$ is entailed. The former however does not contain any $[a]$ operators which are disallowed in precondition axioms.

Inter-Temporal Effects These are present in a durative action $A_j(\vec{z_j})$ with an end effect $\forall \vec{x_i}{:}\vec{\tau_i}.\langle \varphi^s_i, \varphi^o_i, \varphi^e_i\rangle \Rightarrow \psi_i$ which also depends on the truth of conditions at the starting time (φ^s_i) and during the open interval of the duration (φ^o_i) of the action. At the situation where $end(A(\vec{z}), t)$ is to be applied, this information is however no longer accessible. We therefore introduce new fluents C^s_i and C^o_i which will "remember" the truth of those conditions. Their SSAs are

$$\Box[a]C^s_i(\vec{z_j}, \vec{x_i}) \equiv \exists t.a = start(A_j(\vec{z_j}), t) \wedge \varphi^s_i \vee C^s_i(\vec{z_j}, \vec{x_i}) \wedge \neg\exists t'.a = end(A_j(\vec{z_j}), t') \tag{27}$$

$$\Box[a]C^o_i(\vec{z_j}, \vec{x_i}) \equiv \exists t.a = start(A_j(\vec{z_j}), t) \wedge \mathcal{R}[a, \varphi^o_i] \vee C^o_i(\vec{z_j}, \vec{x_i}) \wedge \neg\exists t'.a = end(A_j(\vec{z_j}), t') \wedge \mathcal{R}[a, \varphi^o_i]. \tag{28}$$

We then simply have to let $end(A_j(\vec{z_j}), t)$ have the effect

$$\forall \vec{x_i}{:}\vec{\tau_i}.\ C^s_i(\vec{z_j}, \vec{x_i}) \wedge C^o_i(\vec{z_j}, \vec{x_i}) \wedge \varphi^e_i \Rightarrow \psi_i.$$

Start Duration Constraints Again, the information about the actual duration of an action A_j is not yet available in the situation where the start happening is executed, so it is not possible to just test its start duration constraint there directly. However by introducing a new functional fluent $f^s_{j,i}$ that remembers the values of each fluent f_i appearing in $\delta^s_{A_j}$ by

$$\Box[a]f^s_{j,i}(\vec{z_j}, \vec{x_i}) = y \equiv \exists t.a = start(A_j(\vec{z_j}), t) \wedge y = f_i(\vec{x_i}) \vee f^s_{j,i}(\vec{z_j}, \vec{x_i}) = y \wedge \neg\exists t.a = start(A_j(\vec{z_j}), t), \tag{29}$$

we can test $\delta^s_{A_j}$ at the *end* happening by replacing (25) with[6]

$$\Box \mathit{Poss}(end(A_j(\vec{z_j}), t)) \supset (\delta^s_{A_j})^{f_i(\vec{x_i})}_{f^s_{j,i}(\vec{z_j}, \vec{x_i})} {}^{duration}_{t-since(A_j(\vec{z_j}))} \wedge (\delta^e_{A_j})^{duration}_{t-since(A_j(\vec{z_j}))}. \tag{30}$$

Continuous Effects So far we ignored the fact that durative PDDL actions may contain continuous effects. Remember that ϵ^o_A consists of effects $\langle op, f(\vec{t}), expr\rangle$, where op is $+$ or $-$, meaning that $f(\vec{t})$ increases or decreases linearly by $expr$ over A's duration. Representing this in the $\mathcal{ES}$ action theory is actually straightforward: We just let $start(A(\vec{z}), t)$ have the effect that it adds $linear(0, op{\times}expr, t)$ to $f(\vec{t})$ and $end(A(\vec{z}), t)$ the effect of subtracting $linear(0, op{\times}expr, t)$ again. This approach covers even the case where multiple actions concurrently change the value of the same numeric fluent: The changing rates then simply add up.

However, now it might be the case that an invariant precondition $\pi^o_{A_j}$ is violated *during the execution interval* of A_j. Suppose the $drive$ action of an electric car has the invariant condition that always $power(car) > 0$ and the continuous effect that the power level decreases by 1 per time

[6] By $\phi^{f_i(\vec{x_i})}_{f^s_{j,i}(\vec{z_j}, \vec{x_i})}$ we mean the result of replacing any occurrence of f_i in ϕ by the corresponding $f^s_{j,i}$ with additional parameters $\vec{z_j}$.

unit. If we start driving at time 3, the initial power level is 4 and we want to end driving at 10, then the action should not be possible since the power level already reaches zero at time point 7. To handle this scenario, our proposal is to schedule a force stop of the action at the time point where an invariant would get violated. The invariant then still holds in the *open* interval ending before that particular time point, just as required in PDDL. The idea is implemented by asserting

$$\Box Obli(end(A_j(\vec{z_j}), t)) \subset Performing(A_j(\vec{z_j})) \wedge \neg Eval[\pi^o_{A_j}, t]. \quad (31)$$

Timed Initial Literals

Finally, we can easily implement the timed initial literals by introducing for each $\langle t_i, L_i \rangle$ a new simple action $A_{\langle t_i, L_i \rangle}(t)$ whose only effect is to make L_i true and for which we have:

$$\Box Poss(A_{\langle t_i, L_i \rangle}(t)) \supset t = t_i \quad (32)$$

$$\Box Obli(A_{\langle t_i, L_i \rangle}(t)) \subset t = t_i \quad (33)$$

Correctness

Before we can formulate our main result, we will need the following definition: A *linearization* of a PDDL plan P is a smallest sequence of actions $\langle r_1, \ldots, r_k \rangle$ such that:

- $A_i(\vec{o_i}, t_i)$ is one r_j, if $(t_i{:}A_i(\vec{o_i}))$ is a simple action in P.
- $start(A_i(\vec{o_i}), t_i)$ and $end(A_i(\vec{o_i}), t_i + d_i)$ are among the r_j, if $(t_i{:}A_i(\vec{o_i})[d_i])$ is a durative action in P.
- $A_{\langle t_i, L_i \rangle}$ is one of the r_j, if $\langle t_i, L_i \rangle$ is a timed initial literal whose t_i is less or equal to the ending time of P.
- For all $1 \leq j \leq k-1$, $\Sigma_{\text{time}} \models time(r_j) \leq time(r_{j+1})$.

Since we are using an interleaved model of concurrency, there is in general more than one linearization of a plan.

Let $Final$ abbreviate $\neg\exists a.Performing(a)$. The theorem below now relates the validity of PDDL plans to the entailments of $\mathcal{ES}$ basic action theories, thus drawing the connection between the two semantics definitions.

Theorem 1 *Let Σ be the result of applying the above mapping to a PDDL problem with goal formula ψ. Let P be a plan with no concurrent mutex actions. Then P is valid according to (Fox & Long 2003) and (Edelkamp & Hoffmann 2004) iff there is a linearization $\langle r_1, \ldots, r_k \rangle$ of P such that*

$$\Sigma \models [r_1] \cdots [r_k](Executable \wedge Final \wedge Eval[\psi, now]).$$

Conclusion

We presented an alternative, declarative semantics for the temporal fragment of the planning language PDDL. The new semantics is defined in terms of entailments of action theories in a variant of the situation calculus. Among other things, this allows to more easily relate PDDL to other situation calculus based formalisms such as the agent control language Golog and provides the semantical foundation for an integration of the two. Moreover, it may offer an alternative view on temporal planning in general, thus helping in constructing planners that are complete for temporally expressive domains (Cushing *et al.* 2007). In the future we want to extend the results to also include the yet missing features of PDDL 3.0, namely preferences and plan constraints.

Acknowledgements

This work was supported by DFG grant La747/13-2.

References

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

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proc. IJCAI-07*. AAAI Press.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Universität Freiburg.

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

Gerevini, A., and Long, D. 2005. BNF description of PDDL3.0.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language.

Grosskreutz, H., and Lakemeyer, G. 2000. cc-Golog: Towards more realistic logic-based robot controllers. In *AAAI-00*.

Hsu, C.; Wah, B.; Huang, R.; and Chen, Y. 2006. New features in SGPlan for handling soft constraints and goals preferences in PDDL3.0. In *Proc. of IPC-5 at ICAPS'06*.

Lakemeyer, G., and Levesque, H. J. 2004. Situations, si! situation terms, no! In *Proc. KR2004*. AAAI Press.

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

Lifschitz, V. 1986. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proc. of the 1986 Workshop*. Morgan Kaufmann.

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

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

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the Situation Calculus. In *Proc. KR-1989*. Morgan Kaufmann.

Pinto, J. 1994. *Temporal Reasoning in the Situation Calculus*. Ph.D. Dissertation, Department of Computer Science, University of Toronto, Toronto, Canada.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In *Proc. KR-96*. Morgan Kaufmann.

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

Tarski, A. 1951. *A Decision Method for Elementary Algebra and Geometry*. University of California Press.