A Fluent Calculus Semantics for ADL with Plan Constraints

Conrad Drescher and Michael Thielscher

Department of Computer Science,
Dresden University of Technology
Nöthnitzer Str. 46, 01187 Dresden, Germany

Abstract. Plan constraints are the most recent addition to the ever growing Planning Domain Definition Language (PDDL). In this work we consider the PDDL fragment consisting of basic ADL extended by plan constraints. We provide a purely declarative semantics for this fragment by interpreting it in the basic Fluent Calculus. We thus obtain a logical semantics for this fragment of PDDL instead of the usual meta-theoretical state transition semantics.

1 Introduction

Research in specialized planning languages originates with STRIPS [1]. Over the years this basic language has seen numerous extensions: first to the language ADL [2] and then to PDDL [3], the ever growing language that underlies the annual planning competitions. The new feature in the most recent version PDDL 3.0 [4] are plan constraints that allow to express both requirements and preferences with regard to plan quality.

Traditionally the semantics of planning languages is given in terms of state transitions. There also is a parallel line of research that seeks to provide a logical semantics for planning languages. Such complementary semantics exist for STRIPS [1, 5] and ADL [6, 7]. The recent works [8, 9] aim at successively covering all of the semantics of PDDL 2.1 [10].

In this work we provide a purely declarative semantics for the fragment of PDDL 3.0 consisting of basic ADL and plan constraints. We do so by interpreting this fragment in the basic Fluent Calculus. The resulting system is both natural and expressive.

2 Preliminaries

In this section we recall the theoretical basis upon which our work rests. We start by recalling the basics of Fluent Calculus. Then we identify the fragment of PDDL under consideration — ADL with plan constraints.

2.1 Fluent Calculus

The Fluent Calculus [11] can be seen as a modern extension of the classical Situation Calculus [12]. One of the major differences between Fluent and Situation Calculus is

that the former is action-centered while the latter is fluent-centered — at least in the popular version based on Reiter's successor state axioms [12]. That is to say, in Fluent Calculus we specify for each *action* the effects it has while a successor state axiom specifies for a *fluent* by which actions it is affected. Thus Fluent Calculus arguably is closer to current planning languages, which are also action-centered.

Our work uses a reformulation of the basic Fluent Calculus in a recently proposed unifying action calculus (UAC) [13] that allows us to keep the technical overhead to a minimum. A comprehensive treatment of the classical Fluent Calculus — which also captures the notion of state, i.e. of collections of fluents — can be found in [11].

Unifying Action Calculus The UAC has been introduced with the stated goal of bundling research efforts in the reasoning about action community; it has been shown to encompass the Event, Fluent, and Situation Calculus, as well as planning languages such as ADL.

Formally, the UAC is based on many-sorted first order logic with equality and the four sorts TIME, FLUENT, OBJECT, and ACTION.[1] Fluents are reified, i.e. modeled as terms, and the predicate Holds : FLUENT $\times$ TIME is used to indicate whether a particular fluent evaluates to true at a particular time. For axiomatizing action preconditions the predicate Poss : ACTION $\times$ TIME $\times$ TIME is used.[2] There are only finitely many function symbols into sorts FLUENT and ACTION, respectively.

The UAC abstracts from a particular time structure. It can be instantiated, e.g., by the natural numbers that serve as the linear time structure of the Event Calculus, or by situations that provide the branching time structure of the Fluent and Situation Calculus.

Fluent Calculus Domains Fluent Calculus domains are axiomatized in the UAC with the help of the following formula types:

Definition 1 (Basic Formulas).
For $\bar{\mathbf{s}}$*, a sequence of variables of sort* TIME*, a* state formula $\Phi[\bar{\mathbf{s}}\,]$ in $\bar{\mathbf{s}}$ *is a first-order formula with free variables* $\bar{\mathbf{s}}$ *and where*

- *for each occurrence of* $\mathit{Holds}(f, s)$ *we have* $s \in \bar{\mathbf{s}}$*;*
- *predicate Poss does not occur.*

Let A *be a function into sort* ACTION.

- *A* domain constraint *is a state formula in* s*:*

$$(\forall s)\delta[s].$$

- *A* precondition axiom *is of the form*

$$(\forall)\mathit{Poss}(A(\bar{\mathbf{x}}), s_1, s_2) \;\equiv\; \pi_A[s_1] \wedge s_2 = \mathit{Do}(A(\bar{\mathbf{x}}), s_1),$$

where $\pi_A[s_1]$ *is a state formula in* s_1 *with free variables among* $s_1, \bar{\mathbf{x}}$[3]

[1] By convention variable symbols s, f, x, and a are used for terms of sort TIME, FLUENT, OBJECT, and ACTION, respectively.
[2] Having two arguments of sort TIME allows to model actions with duration or indirect effects.
[3] By $(\forall)\varphi$ we denote the universal closure of φ.

– *An* effect axiom *is of the form*

$$(\forall)\mathit{Poss}(A(\bar{\mathbf{x}}), s, t) \supset$$
$$\bigvee_k (\exists \bar{\mathbf{y}}_k)(\Phi_k[s] \wedge (\forall f)\,[\bigvee_i f = f_{ki} \vee \mathit{Holds}(f, s) \wedge \bigwedge_j f \neq g_{kj} \equiv \mathit{Holds}(f, t)])$$

The f_{ki} and g_{kj} are fluent terms with variables among $\bar{\mathbf{x}}, \bar{\mathbf{y}}_k$ and denote the positive and negative effects, respectively. The Φ_k are state formulas in s with free variables among $s, \bar{\mathbf{x}}, \bar{\mathbf{y}}$ and represent conditions under which the effects materialize. Positive and negative action effects are subject to a natural consistency assumption, namely, we require that

$$\bigwedge_i \bigwedge_j f_{ki} \neq g_{kj}$$

holds for all $k = 1, \ldots, n$.

– *An* initial state axiom *is a state formula in the least element S_0 of sort* TIME.

– *Foundational axioms Σ_{aux} contain a first order axiomatization of situations (the underlying time structure). It is based on two functions into sort* TIME*; the constant S_0 denotes the initial situation and the function Do of sort* ACTION $\times$ TIME *is used to construct successor situations:*

$$(\forall)\mathit{Do}(a_1, s_1) = \mathit{Do}(a_2, s_2) \equiv a_1 = a_2 \wedge s_1 = s_2$$
$$(\forall)\neg s \sqsubset S_0$$
$$(\forall)s \sqsubset \mathit{Do}(a, s') \equiv s \sqsubseteq s'$$
$$\phi[S_0] \wedge (\forall s, a)(\phi[s] \supset \phi[\mathit{Do}(a, s)]) \supset (\forall s')\phi[s']$$

where in the axiom scheme on the last line ϕ ranges over all state formulas in s, with only s free. Foundational axioms Σ_{aux} also contain unique name axioms for sorts ACTION *and* FLUENT*; that is, an axiom of the form*

$$(\forall \bar{\mathbf{x}} \forall \bar{\mathbf{y}}) \bigwedge_{i=1..n-1} \bigwedge_{j=i+1..n} T_i(\bar{\mathbf{x}}) \neq T_j(\bar{\mathbf{y}}) \wedge \bigwedge_{i=1..n} T_i(\bar{\mathbf{x}}) = T_i(\bar{\mathbf{y}}) \supset \bar{\mathbf{x}} = \bar{\mathbf{y}},$$

where the T_i range over all function symbols of the respective sorts. For dealing with arithmetic later on we introduce the sort NUMBER *and include an axiomatization of Presburger arithmetic.*

Definition 2 (Domain Axiomatizations). *A* domain axiomatization *Σ consists of a set Σ_{Poss} of precondition-, and a set $\Sigma_{Effects}$ of effect axioms, each containing one axiom for every function into sort* ACTION*, along with a finite set of domain constraints Σ_{dc}, a finite set of initial state axioms Σ_{Init}, and foundational axioms Σ_{aux}.*

Let us illustrate all the introduced notions by an axiomatization of the familiar blocks world domain:

Example 1 (Blocks World Axiomatization). The *precondition* of moving a block from some location x to location y is expressed by the following axiom:

$$\begin{aligned}(\forall)\text{Poss}(\text{Move}(\text{block}_1, x, y), s_1, s_2) \equiv \\ \text{Holds}(\text{On}(\text{block}_1, x), s_1) \wedge x \neq y \wedge \\ (\neg\exists\text{block}_2)\text{Holds}(\text{On}(\text{block}_2, \text{block}_1), s_1) \wedge \\ (\neg\exists\text{block}_3)(\text{Holds}(\text{On}(\text{block}_3, y), s_1) \vee y = \text{Table}) \wedge \\ s_2 = \text{Do}(\text{Move}(\text{block}_1, x, y), s_1).\end{aligned}$$

The *effects* of moving a block are axiomatized as follows:

$$\begin{aligned}&(\forall)\text{Poss}(\text{Move}(\text{block}_1, x, y), s_1, s_2) \supset \\ &\quad[(\forall f)(f = \text{On}(\text{block}_1, y) \vee (\text{Holds}(f, s_1) \wedge f \neq \text{On}(\text{block}_1, x))) \equiv \text{Holds}(f, s_2)].\end{aligned}$$

The following *domain constraint* expresses the fact that every block is situated at exactly one location:[4]

$$(\exists!y)\text{Holds}(\text{On}(x, y), s).$$

Finally, suppose that the following axiom describes what is known about the initial situation:

$$(\forall f)\text{Holds}(f, S_0) \equiv f = \text{On}(\text{Block}_1, \text{Table}) \vee f = \text{On}(\text{Block}_2, \text{Table}).$$

It can be easily verified that this axiomatization, together with the unique name axioms for the blocks and the table, entails

$$\text{Holds}(\text{On}(\text{Block}_2, \text{Block}_1), \text{Do}(\text{Move}(\text{Block}_2, \text{Table}, \text{Block}_1), S_0)).$$

3 ADL with Plan Constraints

In this section we introduce the fragment of PDDL 3.0 that we consider in this work — ADL with plan constraints. For a general introduction to action languages based on state transition semantics the reader is referred to [14].

3.1 ADL

ADL has originally been introduced to cover the expressive middle-ground between STRIPS and the Situation Calculus. It still plays an important role in the sequential, deterministic part of the international planning competitions.

Definition 3 (ADL Signature). *An ADL signature is based on a finite set of types, where types may also be defined as unions of other types. The basic type* OBJECT *is always included. The signature then contains a finite set of typed constants $\mathcal{C}$ and typed variables $\mathcal{V}$. It also includes a finite set of typed fluents $\mathcal{F}$ of arity ≥ 0 and likewise a finite set of typed operator names $\mathcal{A}$ with associated arity.*

[4] By $\exists!x\phi[x]$ we abbreviate the first order formula expressing that there is exactly one x such that $\phi[x]$.

Planning problems are expressed in ADL with the help of the following constructs:

Definition 4 (Basic ADL Constructs).

- *A state formula $\phi[\bar{\mathbf{x}}]$ is a first order formula with free variables among $\bar{\mathbf{x}}$ containing as atoms only fluents $F(\bar{\mathbf{t}})$ and equalities $\bar{\mathbf{t}}_1 = \bar{\mathbf{t}}_2$.*
- *An effect formula is the universal closure of a first order conjunction built from the following inductively defined admissible components:*
  - *fluent literals $F(\bar{\mathbf{t}})$ and $\neg F(\bar{\mathbf{t}})$ are admissible;*
  - *if ϕ and ψ are admissible then the conjunction $\phi \wedge \psi$ and the universally quantified $(\forall \bar{\mathbf{x}})\phi$ are;*
  - *if ϕ is a state formula and ψ is admissible with no occurrence of $\Rightarrow$ or $\forall$ then $\phi \Rightarrow \psi$ is.*
- *For an operator name $A \in \mathcal{A}$ the* ADL operator *A is a triple $\langle \bar{\mathbf{x}}, \pi_A, \epsilon_A \rangle$, where*
  - *the variables $\bar{\mathbf{x}}$ denote the operator's typed* parameters *(possibly zero);*
  - *the* state formula *$\pi_A[\bar{\mathbf{x}}]$ denotes the operator's precondition; and*
  - *the* effect formula *$\epsilon_A[\bar{\mathbf{x}}]$ denotes the operator's effects.*

The $\Rightarrow$ construct is not to be confused with implication; its purpose is to relate states and successor states. The definition ensures that to the right of the $\Rightarrow$ construct there is always a conjunction of fluent literals. We proceed by defining a normal form for ADL operators; the informed reader should note that this definition deviates from the one used in [9, 13].

Definition 5 (Operator Normal Form). *An ADL operator A is in* normal form *if its effect formula has the following syntactic form:*

$$\bigvee_k (\forall \bar{\mathbf{x}}_k)\phi_k[\bar{\mathbf{x}}_k] \Rightarrow \delta_k[\bar{\mathbf{x}}_k]$$

where $\phi_k[\bar{\mathbf{x}}_k]$ is a state formula with free variables among $\bar{\mathbf{x}}_k$ and $\delta_k[\bar{\mathbf{x}}_k]$ is a conjunction of fluent literals with free variables among $\bar{\mathbf{x}}_k$. Further, we require that all ϕ_k are mutually exclusive.

The following proposition states that we lose nothing by making this operator normal form mandatory:

Proposition 1 (Operator Normal Form). *For every effect formula there exists an equivalent effect formula in normal form.*

Proof (Sketch). The key observation is that we can always replace e.g. $\top \Rightarrow \delta_1 \wedge \phi \Rightarrow \delta_2$ by the formula $(\phi \Rightarrow \delta_1 \wedge \delta_2) \vee (\neg\phi \Rightarrow \delta_1)$. □

Observe that rewriting an operator to normal form may introduce an exponential blowup.

Example 2 (Blocks World ADL Operator). The following is an ADL operator in normal form for the action $\mathrm{Move}(\mathrm{block}, x, y)$ in the blocks world:

$$\begin{aligned} \text{Precondition: } & \mathrm{On}(\mathrm{block}_1, x) \wedge x \neq y \wedge \\ & (\neg\exists \mathrm{block}_2)\mathrm{On}(\mathrm{block}_2, \mathrm{block}_1) \wedge \\ & (\neg\exists \mathrm{block}_3)(\mathrm{On}(\mathrm{block}_3, y) \vee y = \mathrm{Table}) \\ \text{Effects: } & \top \Rightarrow \mathrm{On}(\mathrm{block}_1, x) \wedge \neg\mathrm{On}(\mathrm{block}_1, y) \end{aligned}$$

Definition 6 (ADL Problem Descriptions). *An ADL planning problem consists of:*

- *an ADL operator in normal form for each operator name;*
- *an initial state specification in the form of a conjunction of ground fluent literals; and*
- *a goal description in the form of a closed state formula.*

The following is an ADL problem description analogous to the Fluent Calculus domain from example 1:

Example 3 (ADL Blocks World Description). The only operator shall be as given in example 2 above. Let the initial state be specified by $\mathrm{On}(\mathrm{Block}_1, \mathrm{Table}) \wedge \mathrm{On}(\mathrm{Block}_1, \mathrm{Table})$ and the goal consist of stacking up all blocks, axiomatized as $(\exists!\mathrm{block})\mathrm{On}(\mathrm{block}, \mathrm{Table})$.

ADL admits both open and closed world reasoning – in the open world case the truth-value of fluent literals may be unknown. The existing state transition semantics for ADL from [10], however, is based on the closed world assumption. In this setting the initial state specification is a conjunction of ground fluent atoms. This specification is completed by adding the negation of every ground fluent atom that does not yet occur in the initial state specification, so that eventually every ground fluent atom of the language or its negation occurs in the initial state specification.

The semantics of ADL also makes strong assumptions about the meaning of the constants $\mathcal{C}$: no two constants denote the same object (uniqueness of names) and all existing objects are named by some constant. This latter requirement allows for substitutional quantification: e.g. a subformula $(\forall x)P(x)$ can equivalently be written as $\bigwedge_{C_i} P(C_i)$ where the C_i are all the constants of the domain. Thus, although ADL uses the language of first order logic it does not employ first order semantics.

A plan for an ADL planning problem is a ground sequence $\langle A_1(\bar{\mathbf{t}}_1), \ldots, A_n(\bar{\mathbf{t}}_n)\rangle$ of operators A_i with constants $\bar{\mathbf{t}}_i$ substituted for the parameters $\bar{\mathbf{x}}_i$. A plan is a solution for the planning problem iff the state obtained by sequentially applying the operators $A_i(\bar{\mathbf{t}}_i)$ to the initial state yields a state satisfying the goal description.

3.2 Plan Constraints

Plan constraints allow to express both hard and soft constraints on the computed plans: "hard" means that a constraint *has* to be satisfied while "soft" means that it *should*, if possible.

State Trajectory Constraints State trajectory constraints are the hard constraints. They allow to express that some property has to hold throughout/at some point/etc. in the plan.

Formally, state trajectory constraints are handled by introducing modalities that can be used in goal descriptions. The available modalities are `at end`, `always`, `sometime`, `at-most-once`, `sometime-after`, and `sometime-before`. We omit the modalities `within` and `always-within` since these require an explicit notion of time that is not supported by the ADL subset of PDDL. The modalities may

not be nested. They can be combined by logical conjunction and be universally quantified from the outside. Universally quantified constraints only serve as shorthand for the equivalent ground formula.[5] For example we can enforce that a property ϕ holds throughout a plan that achieves the goal ψ by writing $\psi \wedge \texttt{always}\ \phi$.

The original semantics for state trajectory constraints in PDDL has been defined in terms of sequences of state-timepoint pairs $\langle (S_0, 0), (S_1, t_1), \ldots, (S_n, t_n) \rangle$, where the S_i denote all the states that occur during plan execution in chronological order. In the case of ADL this can be simplified to sequences of states $\langle S_0, \ldots, S_n \rangle$.

Definition 7 (Semantics of Temporal Modalities). *The semantics of the temporal modalities is then as follows:*

$\langle S_0, \ldots, S_n \rangle \vDash \phi$	*iff*	$S_n \vDash \phi$
$\langle S_0, \ldots, S_n \rangle \vDash \texttt{at end}\ \phi$	*iff*	$S_n \vDash \phi$
$\langle S_0, \ldots, S_n \rangle \vDash \texttt{always}\ \phi$	*iff*	$\forall i : 0 \leq i \leq n : S_i \vDash \phi$
$\langle S_0, \ldots, S_n \rangle \vDash \texttt{sometime}\ \phi$	*iff*	$\exists i : 0 \leq i \leq n : S_i \vDash \phi$
$\langle S_0, \ldots, S_n \rangle \vDash \texttt{at-most-once}\ \phi$	*iff*	$\forall i : 0 \leq i \leq n :$ *if* $S_i \vDash \phi$ *then* $\neg\exists j, k : i < j < k \leq n :$ $S_j \vDash \neg\phi$ *and* $S_k \vDash \phi$
$\langle S_0, \ldots, S_n \rangle \vDash \texttt{sometime-after}\ \phi\ \psi$	*iff*	$\exists i : 0 \leq i \leq n : S_i \vDash \phi$ *implies* $\exists j : i < j \leq n : S_j \vDash \psi$
$\langle S_0, \ldots, S_n \rangle \vDash \texttt{sometime-before}\ \phi\ \psi$	*iff*	$\exists i : 0 \leq i \leq n : S_i \vDash \phi$ *implies* $\exists j : 0 \leq j < i : S_j \vDash \psi$

The expression $\texttt{at-most-once}\ \phi$ prohibits that ϕ changes its truth-value to false and back to true in the course of a plan. A constraint using the `always` modality might conflict with the initial state specification; we tacitly assume that initial state specifications ϕ do not violate any constraints. State trajectory constraints can appear both in the planning problem file and in the action domain file [4].

Preferences Preferences are the soft constraints, i.e. properties that are desired but not required to hold. Instead of an elaborate qualitative model of preferences PDDL adopts a quantitative model.

The syntax for ADL preferences is

$$\texttt{preference}\ \phi,$$

where ϕ denotes a state formula possibly containing state trajectory constraints.[6] As in the case of state trajectory constraints preferences may not be nested, and only be combined by logical conjunction. Again, formulas of the form

$$(\forall \bar{\mathbf{x}})\texttt{preference}\ \phi(\bar{\mathbf{x}})$$

may be used as shorthand for the logical equivalent conjunction

$$\bigwedge_{\bar{\mathbf{t}}} \texttt{preference}\ \phi(\bar{\mathbf{t}})$$

[5] Recall that ADL admits substitutional quantification.

[6] But note that state trajectory constraints may not contain preferences.

where $\bar{\mathbf{t}}$ denotes all possible ground substitutions for the variables $\bar{\mathbf{x}}$. ADL preferences may occur in goals and in operator preconditions. In the latter case they must not contain the state trajectory modalities.

The semantics of preferences is simple and intuitive. A preference simply always evaluates to true. However, a preference can be satisfied or violated. Let $\langle S_0, \ldots, S_n \rangle$ denote the sequence of states corresponding to a plan. If the precondition of the operator applied to state S_i contains a preference `preference` ϕ, then the preference is violated if $\langle S_i \rangle \vDash \neg\phi$. Likewise a preference occurring in the planning goal is violated if $\langle S_0, \ldots, S_n \rangle \vDash \neg\phi$. An overall penalty is assigned to the plan and equals the sum of

- the number of `preference` ϕ expressions from operator preconditions that have been violated; and
- the number of `preference` ϕ expressions from the goal description that have been violated.

The optimal plan in the setting of ADL with plan constraints is the plan with the minimal number of preferences violated. It is worth pointing out that the notion of optimality crucially depends on complete information — in the case of open world ADL it may be impossible to identify whether a plan is optimal.

4 The Fluent Calculus Semantics for ADL with Plan Constraints

The Fluent Calculus semantics for ADL with plan constraints is obtained by correctly embedding the latter into the former.

4.1 Scope of the ADL Constraints

First off we have to decide whether the constraints from the action domain file and the planning problem file should be treated alike or not. Quoting from [4], "constraints (...) specified in the action domain file (...) might be seen as safety conditions (...) that must alway be respected in any valid plan for the domain (...)." This seems to suggest that these constraints are intended to serve a purpose similar to that of domain constraints in the Fluent Calculus. However, quoting from [15] constraints from the planning problem file "(...) are added to those (if any) in the domain file and together they represent a collection of goals that must be satisfied by any valid plan.". So we adopt the viewpoint that the constraints apply only to the planning goal and not to the action domain as a whole. Let us illustrate the issue at hand by a small example.

Example 4 (Scope of ADL Constraints). Assume that in the ADL blocks world domain from example 3 the constraint `always` $\mathrm{On}(\mathrm{Block}_1, \mathrm{Table})$ is part of the action domain file. Let Σ denote the Fluent Calculus axiomatization of this domain from example 1. If we extend Σ by the domain constraint $(\forall s)\mathrm{Holds}(\mathrm{On}(\mathrm{Block}_1, \mathrm{Table}), s)$ the resulting theory is inconsistent. Instead we extend the reasoning problem $\Sigma \vDash (\exists s)\phi(s)$ to the additionally qualified $\Sigma \vDash (\exists s)\phi(s) \wedge \neg(\exists s')s' \sqsubseteq s \wedge \neg\mathrm{Holds}(\mathrm{On}(\mathrm{Block}_1, \mathrm{Table}), s')$ — where ϕ denotes the goal description.

4.2 The Mapping

We are finally ready to define the mapping:

A Corresponding Language We start by defining a Fluent Calculus signature based on the ADL signature. In order to simplify the presentation we will assume that ADL problems only include the single type OBJECT. Our results can easily be reformulated in an appropriately sorted version of the Fluent Calculus, so that this is without loss of generality. Given an ADL problem based on a signature with constants $\mathcal{C}$, operator names $\mathcal{A}$, and fluent predicates $\mathcal{F}$ we create a Fluent Calculus signature with corresponding constants $\mathcal{C}'$, functions into sort ACTION $\mathcal{A}'$, and functions into sort FLUENT $\mathcal{F}'$. In order to deal with preferences we introduce the additional sort NUMBER and extend the foundational axioms Σ_{aux} by an axiomatization of the natural numbers.

Based on this signature we include unique-name-axioms for sort OBJECT. Likewise we include a domain closure axiom for sort OBJECT, that is an axiom of the form:

$$(\forall x) \bigvee_{i=1..n} x = c_i,$$

where x is a variable of sort OBJECT and the c_i denote all object constants of the signature. For dealing with preferences we introduce a special fluent, Penalty$/1$, that takes a natural number as argument.

The Initial State Mapping ADL initial state specifications ϕ to Fluent Calculus initial state axioms $\phi'[S_0]$ is done in the obvious way: replace every fluent $F(\bar{\mathbf{x}})$ in ϕ by $\text{Holds}(F'(\bar{\mathbf{x}}), S_0)$. Below for an ADL state formula ψ by $\psi'[s]$ we will denote the corresponding Fluent Calculus state formula obtained in this fashion for an arbitrary situation s. Finally we include $\text{Holds}(\text{Penalty}(0), S_0)$ into $\phi[S_0]$. The purpose of this fluent will be to accumulate the number of preferences violated.

The Operators Mapping ADL operators to Fluent Calculus consists of creating corresponding precondition and effect axioms. First we introduce a bit of notation. Let A be an operator with operator precondition π_A. Without loss of generality we assume that π_A is of the form $\pi_{A_1} \wedge \pi_{A_2}$ where π_{A_1} is an ordinary ADL precondition and π_{A_2} is a conjunction of preferences. Then denote

- by $\Pi_{A_{\text{pref}}}$ the set consisting of the logical parts ψ_i of the preferences `preference` ψ_i from π_{A_2}; and
- by $\Pi_{A_{\text{pref-cases}}}$ the set of pairs $\langle \bigwedge_i (\neg)\psi_i, n_j \rangle$ where each $\psi_i \in \Pi_{A_{\text{pref}}}$ and $n_j \in \mathbb{N}$ equals the number of $\neg\psi_i$ that occur in $\bigwedge_i (\neg)\psi_i$ where $\psi_i \in \Pi_{A_{\text{pref}}}$.

That is to say, n_j denotes the number of preferences that are violated if $\bigwedge_i (\neg)\psi_i$ holds. There are 2^i such pairs in $\Pi_{A_{\text{pref-cases}}}$ and by construction these pairs are mutually exclusive.

For every ADL operator $A = \langle \bar{\mathbf{x}}, \pi_A, \epsilon_A \rangle$ from the planning problem we define the action precondition axiom

$$(\forall \bar{\mathbf{x}}, s)\text{Poss}(A(\bar{\mathbf{x}}), s_1, s_2) \equiv \pi'_{A_1} \wedge s_2 = \text{Do}(A(\bar{\mathbf{x}}), s).$$

Recall that we assume ϵ_A to be of the form $\bigvee_k (\forall \bar{\mathbf{x}}_k)\phi[\bar{\mathbf{x}}_k] \Rightarrow \delta[\bar{\mathbf{x}}_k]$. By Δ^+ (Δ^-) denote the set of positive (negative) literals from $\delta[\bar{\mathbf{x}}_k]$. Define the corresponding Fluent Calculus effect axiom as:

$$
\begin{aligned}
&(\forall)\text{Poss}(A(\bar{\mathbf{x}}), s_1, s_2) \supset \\
&\quad \bigvee_{k_i} \phi'[\bar{\mathbf{x}}_k, s_1] \wedge \gamma_i'[s_1] \wedge \text{Holds}(\text{Penalty}(n_1), s_1) \wedge n_2 = n_1 + n_i \wedge \\
&\qquad [\![(\forall f)[f = \text{Penalty}(n_2) \vee \bigvee_{F(\bar{\mathbf{x}}) \in \Delta^+} f = F(\bar{\mathbf{y}})] \vee \\
&\qquad\quad [\text{Holds}(f, s_1) \wedge f \neq \text{Penalty}(n_1) \wedge \bigwedge_{F(\bar{\mathbf{x}}) \in \Delta^-} f \neq F(\bar{\mathbf{x}})] \\
&\qquad \equiv \text{Holds}(f, s_2)]\!],
\end{aligned}
$$

where $\langle \gamma_i, n_i \rangle \in \Pi_{A_{\text{pref-cases}}}$. Each of the k_i disjuncts states that, if prior to action application

- the accumulated penalty equates n_1; and
- case k of the ADL operator applies,

then after action application a fluent f holds if-and-only if

- f is equal to Penalty$(n2)$ where $n2$ is the new accumulated penalty; or
- f is a positive effect of the ADL operator; or
- f does not equal Penalty$(n1)$; or
- f held prior to action application and is not a negative effect of the ADL operator.

Let us stress that all the k_i disjuncts are mutually exclusive. This completes the definition of a Fluent Calculus domain Σ corresponding to an ADL planning problem.

The Goal Descriptions We now turn to goal descriptions. These will be mapped to Fluent Calculus queries that will be evaluated with regard to the domain axiomatization Σ. If the ADL goal description ϕ does not contain any constraints our task is easy: we simply ask whether

$$
\begin{aligned}
\Sigma \vDash\ & (\exists s, n)\phi'[s] \wedge \text{Holds}(\text{Penalty}(n), s) \wedge \\
& (\neg\exists s', n')\phi'[s'] \wedge \text{Holds}(\text{Penalty}(n'), s') \wedge n' < n.
\end{aligned} \tag{1}
$$

We proceed by extending this mapping to goal descriptions containing constraints. Without loss of generality we can assume that the goal description ϕ is of the form $\phi_1 \wedge \phi_2 \wedge \phi_3$, where

- ϕ_1 is an ordinary ADL goal;
- ϕ_2 is a conjunction of state trajectory constraints; and
- ϕ_3 is a conjunction of preferences.

The corresponding Fluent Calculus query is of the form

$$(\exists s, n, n_{\text{final}})\psi_1[s, n] \wedge \psi_2[s] \wedge \psi_3[s, n, n_{\text{final}}],$$

where $\psi_1[s, n]$

– is the formula from (1) if there are no preferences ϕ_3 in the goal description;
– is $(\exists s, n)\phi'[s] \wedge \text{Holds}(\text{Penalty}(n), s)$ otherwise.

The mapping from the hard state trajectory constraints ϕ_2 to $\psi_2[s]$ can be obtained from the base cases depicted in figure 1.

ADL constraint	Fluent Calculus subquery
`at end` ψ	$\psi'[s]$
`always` ψ	$(\forall s_1)s_1 \sqsubseteq s \supset \psi'[s_1]$
`sometime` ψ	$(\exists s_1)s_1 \sqsubseteq s \wedge \psi'[s_1]$
`at-most-once` ψ	$(\exists s_1)s_1 \sqsubseteq s \wedge \psi'[s_1] \supset$ $\neg(\exists s_2, s_3)s_1 \sqsubseteq s_2 \wedge s_2 \sqsubseteq s_3 \wedge s_3 \sqsubseteq s \wedge \neg\psi'[s_2] \wedge \psi'[s_3]$
`sometime-after` ψ_1 ψ_2	$(\exists s_1)s_1 \sqsubseteq s \wedge \psi_1'[s_1] \supset (\exists s_2)s_1 \sqsubseteq s_2 \wedge s_2 \sqsubseteq s \wedge \psi_2'[s_2]$
`sometime-before` ψ_1 ψ_2	$(\exists s_1)s_1 \sqsubseteq s \wedge \psi_1'[s_1] \supset (\exists s_2)s_2 \sqsubseteq s_1 \wedge \psi_2'[s_2]$

Fig. 1. Mapping State Trajectory Constraints to Fluent Calculus.

For the mapping from the preferences $\phi_3 = \bigwedge_k$ `preference` φ_k in the goal description to the Fluent Calculus subquery $\psi_3[s, n, n_{\text{final}}]$ we introduce again some notation: denote by Φ_{cases} the set of pairs $\langle \bigwedge_k (\neg)\varphi_k, n_k \rangle$ where $n_k \in \mathbb{N}$ equals the number of $\neg\varphi_k$ that occur in $\bigwedge_k (\neg)\varphi_k$. Without loss of generality we assume that φ_k is of the form $\varphi_{k_1} \wedge \varphi_{k_2}$, where φ_{k_1} is an ADL state formula and φ_{k_2} is a conjunction of state trajectory constraints. We map φ_{k_1} to $\varphi'_{k_1}[s]$ and φ_{k_2} to $\varphi^*_{k_2}[s]$ — where $\varphi^*_{k_2}[s]$ is obtained analogously to the mapping from ϕ_2 to $\psi_2[s]$. With a little abuse of notation we denote $\varphi'_{k_1}[s] \wedge \varphi^*_{k_2}[s]$ by $\varphi'_k[s]$.

We now define the Fluent Calculus subquery $\psi_3[s, n, n']$ corresponding to ϕ_3 to be

$$\bigwedge_i \bigwedge_k (\neg)\varphi'_k[s] \supset n_{\text{final}} = n + n_i) \wedge$$
$$(\neg\exists s', n', n'_{\text{final}})\psi_1[s', n'] \wedge \psi_2[s'] \wedge$$
$$\bigwedge_i \bigwedge_k (\neg)\varphi'_k[s'] \supset n'_{\text{final}} = n' + n_i) \wedge n'_{\text{final}} < n_{\text{final}},$$

where $\langle \bigwedge_k (\neg)\varphi_k, n_i \rangle \in \Phi_{\text{cases}}$. This subquery ensures plan optimality by requiring that

– n_{final} is the sum of
  • the penalty n that stems from violated preferences in action preconditions and
  • the number n_i of preferences φ_k from the goal description violated by s; and
– there does not exists a situation s' satisfying the goal description ψ_1 and the hard plan constraints ψ_2 with a smaller final penalty.

4.3 Correctness of the Translation

We have defined a mapping from ADL planning problems with plan constraints to Fluent Calculus domain axiomatizations Σ and Fluent Calculus queries $(\exists s)\phi[s]$. We are now ready to state our main result:

Theorem 1 (Correctness of the Translation). *Let the Fluent Calculus domain Σ and query $(\exists s)\phi[s]$ be obtained from an ADL planning problem via our mapping. A sequence $\langle A_1(\bar{\mathbf{t}}_1), \ldots, A_n(\bar{\mathbf{t}}_n)\rangle$ of ground ADL operators $A_i(\bar{\mathbf{t}}_i)$ is an optimal solution for the planning problem if and only if $\Sigma \models \phi[Do(A_n(\bar{\mathbf{t}}_n), Do(A_{n-1}(\bar{\mathbf{t}}_{n-1}), \ldots S_0)\ldots)]$.*

Proof (Sketch). The full proof of this theorem is quite tedious and therefore omitted. However, in order to provide some evidence for the correctness of the theorem, we point out that our embedding is very generic, in the sense that the Fluent Calculus domain axiomatization Σ and the ADL planning problem correspond axiom-by-axiom. □

5 Summary

5.1 Related Work

The series of works [7–9] successively provided logical semantics for more and more expressive fragments of PDDL by interpreting these in a recently proposed first order modal variant of the Situation Calculus [16]. None of these works covers plan constraints yet. Many ontological features of PDDL like e.g. concurrent actions and actions with duration are not present in the basic Situation Calculus, however. Thus, in order to obtain a mapping from PDDL fragments to Situation Calculus the underlying logic had to be considerably extended. For Fluent Calculus such extensions have first been introduced in [17]. Most likely these ideas can be adapted in order to obtain Fluent Calculus semantics for equally expressive fragments of PDDL as those covered in [7–9].

Using a mapping defined in [13] we can obtain Situation Calculus axiomatizations corresponding to their Fluent Calculus counterparts. This immediately yields a Situation Calculus semantics for ADL with plan constraints.

Instead of plan constraints costs associated to actions have been introduced for the sequential deterministic part of this year's planning competition at ICAPS-08.[7] This system can also very naturally be interpreted in the Fluent Calculus; plan costs can be computed by summing over situation terms.

Our result identifies a fragment of the Fluent Calculus for which reasoning can be based on efficient specialized planning software instead of the more general constraint logic programming implementation Flux [11]. A tight integration may be achieved by adopting ideas from [18], where planning problems have efficiently been encoded and solved in CLP(FD). Note that reversing our mapping does not introduce an additional blowup as opposed to compiling operators to normal form.

5.2 Conclusion

We have given a purely declarative semantics for ADL with plan constraints by interpreting it in the basic Fluent Calculus. Our semantics is logical, as opposed to the only previously available semantics, which was based on state transitions. Along the way we have clarified the role played by state trajectory constraints by determining their scope. The resulting system is expressive and — since both PDDL and the Fluent Calculus are action-centered formalisms — very natural.

[7] See `http://ipc.informatik.uni-freiburg.de/`.

References

1. Fikes, R.E., Nilsson, N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence **2** (1971) 189–208
2. Pednault, E.P.D.: ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In: Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR 89), San Mateo, California, Morgan Kaufmann (1989) 324–332
3. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL—The Planning Domain Definition Language (1998) `ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`.
4. Gerevini, A., Long, D.: Preferences and Soft Constraints in PDDL 3.0. In: Proceedings of the ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning, Lake District of the UK (2006) 46–53
5. Lifschitz, V.: On the Semantics of STRIPS. In: Reasoning about Actions and Plans, Temberline, Oregon, Morgan Kaufmann (1986) 1–9
6. Pednault, E.P.D.: ADL and the State-Transition Model of Action. Journal of Logic and Computation **4** (1994) 467–512
7. Claßen, J., Lakemeyer, G.: A Semantics for ADL as Progression in the Situation Calculus. In: Proceedings of the 11th International Workshop on Non-Monotonic Reasoning (NMR06), Lake District, UK (2006)
8. Claßen, J., Hu, Y., Lakemeyer, G.: A Situation-Calculus Semantics for an Expressive Fragment of PDDL. In: Proceedings of the Twenty-second National Conference on Artificial Intelligence (AAAI 2007), Menlo Park, CA (2007) 956–961
9. Claßen, J., Eyerich, P., Lakemeyer, G., Nebel, B.: Towards an Integration of Golog and Planning. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 07), Hyderabad, India (2007) 1846–1851
10. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research **20** (2003) 61–124
11. Thielscher, M.: Reasoning Robots: The Art and Science of Programming Robotic Agents. Springer, Dordrecht, NL (2005)
12. Reiter, R.: Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. MIT Press, Cambridge, MA (2001)
13. Thielscher, M.: A Unifying Action Calculus. Artificial Intelligence (submitted) (2007) `www.fluxagent.org/publications.htm`.
14. Gelfond, M., Lifschitz, V.: Action Languages. Electronic Transactions on Artificial Intelligence **3** (1998) `http://www.ep.liu.se/ea/cis/1998/016/`.
15. Gerevini, A., Long, D.: BNF Description of PDDL3.0. `http://zeus.ing.unibs.it/ipc-5/` (2005)
16. Lakemeyer, G., Levesque, H.J.: Situations, Si! Situation Terms, No! In: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 04), Whistler, Canada (2004) 516–526
17. Thielscher, M.: The Concurrent, Continuous Fluent Calculus. Studia Logica **67** (2001) 315–331
18. Dovier, A., Formisano, A., Pontelli, E.: Multivalued Action Languages with Constraints in CLP(FD). In: Proceedings of the Twenty-third International Conference on Logic Programming (ICLP 2007), Porto, Portugal (2007) 255–270