A Situation-Calculus Semantics for an Expressive Fragment of PDDL

Jens Claßen[1] Yuxiao Hu[2] Gerhard Lakemeyer[1]

[1]RWTH Aachen University, 52056 Aachen, Germany

[2]University of Toronto, Toronto, Ontario M5S3G4, Canada

July 26, 2007

Outline

Background and Motivation

A state-transitional semantics for PDDL exists (Fox & Long 03)

- Meta-theoretic, *e.g.*
  - Invariant conditions protected by dummy actions
  - Conditional effects handled by splitting an action into two
- Complexity (19-page definition)

Goal: A declarative semantics for PDDL

- Based on a well-understood logic
- Analyze planning problems with logical entailments
- Bridge planning and reasoning-about-actions communities
  - *e.g.* Embedding planners in temporal Golog

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1,l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1,l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

```
(:action unplug (:parameters ?v - vehicle)
    (:effect (not (power ?v))))
```

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.
A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1,l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

```
(:durative-action drive
    (:parameters ?v - vehicle ?l1 ?l2 - location)
```

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1,l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

```
(:durative-action drive
    (:duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v))))
```

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1,l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

```
(:durative-action drive
    (:condition ... (over all (power ?v)) ...)
```

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1,l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

```
(:durative-action drive
    (:effect (when (at start (not (engine ?v)))
        (at start (engine ?v))))
```

The Planning Domain Definition Language

We cover PDDL 2.1 & 2.2, excluding derived predicates.

A running example: The Electro-Car domain

- Predicates: $At(v, l)$, $Engine(v)$, $Power(v)$;
- Functions: $miles(v)$, $velocity(v)$, $distance(l_1, l_2)$.
- Actions
  - $unplug(v)$: simple action that removes power of v
  - $drive(v, l_1, l_2)$: durative action with duration $\frac{distance(l_1, l_2)}{velocity(v)}$ and $Power(v)$ as invariant condition. If engine was off before the start of driving, turn it on at start and off again at end.

```
(:durative-action drive
    (:effect (when (at start (not (engine ?v)))
        (at end (not (engine ?v)))))
```

The Logic $\mathcal{ES}$

$\mathcal{ES}$ (Lakemeyer & Levesque 04) is a modal logic capturing SitCalc

- $[a]\alpha$: formula α holds after action a
- $\Box\alpha$: formula α holds after any sequence of actions

The Logic $\mathcal{ES}$

$\mathcal{ES}$ (Lakemeyer & Levesque 04) is a modal logic capturing SitCalc

- $[a]\alpha$: formula α holds after action a
- $\Box\alpha$: formula α holds after any sequence of actions

The Basic Action Theory $\Sigma = \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0$

- Σ_{pre}: precondition axiom $\Box Poss(a) \equiv \Pi$,

  $$e.g.\ \Box Poss(a) \equiv a = move(x,y) \wedge Clear(x) \wedge Clear(y) \vee a = moveToTable(x) \wedge Clear(x)$$

- Σ_{post}: successor state axioms $\Box[a]F(\vec{x}) \equiv \Phi_F(\vec{x},a)$

  $$e.g.\ \Box[a]On(x,y) \equiv a = move(x,y) \vee On(x,y) \wedge \neg(a = moveToTable(x) \vee \exists z.a = move(x,z))$$

- Σ_0: initial database,

  $$e.g.\ On(x,y) \equiv (x = a \wedge y = b) \vee (x = b \wedge y = c)$$

The Logic $\mathcal{ES}$

$\mathcal{ES}$ (Lakemeyer & Levesque 04) is a modal logic capturing SitCalc

- $[a]\alpha$: formula α holds after action a
- $\Box\alpha$: formula α holds after any sequence of actions

The Basic Action Theory $\Sigma = \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0$

- Σ_{pre}: precondition axiom $\Box Poss(a) \equiv \Pi$,
  e.g. $\Box Poss(a) \equiv a = move(x, y) \land Clear(x) \land Clear(y) \lor$
  $a = moveToTable(x) \land Clear(x)$
- Σ_{post}: successor state axioms $\Box[a]F(\vec{x}) \equiv \Phi_F(\vec{x}, a)$
  e.g. $\Box[a]On(x, y) \equiv a = move(x, y) \lor$
  $On(x, y) \land \neg(a = moveToTable(x) \lor \exists z.a = move(x, z))$
- Σ_0: initial database,
  e.g. $On(x, y) \equiv (x = a \land y = b) \lor (x = b \land y = c)$

The regression operator $\mathcal{R}$ transforms a formula to an equivalent one without $[a]$ operator, *e.g.* $\Sigma \models [a]\alpha$ iff $\Sigma_0 \models \mathcal{R}[a, \alpha]$

Extensions to $\mathcal{ES}$

Temporal extension similar to (Pinto & Reiter) in SitCalc

- Time:
    - $A(\vec{x})$ is extended to $A(\vec{x}, t)$, with $time(A(\vec{x}, t)) = t$
    - The start time of current situation: $\Box[a](now = time(a))$
    - Ensure correct temporal ordering: $\Box Poss(a) \supset now \leq time(a)$
- Durative actions modeled by instantaneous actions + fluents
  $start(walk(x, y), t)$, $end(walk(x, y), t)$,
  $Performing(walk(x, y))$, $since(walk(x, y))$

Simple Actions

Mapping ADL problems to BAT in $\mathcal{ES}$ follows (Claßen *et al.* 07)

- Initial Database Σ_0: Initial world + Typing
- Precondition axiom Σ_{pre}: Case disjunction over all operators
- Successor state axioms Σ_{post}
  - For each fluent predicate $F_j(\vec{x_j})$, extract the positive condition $\gamma^+_{F_j}$ and the negative condition $\gamma^-_{F_j}$ from the effect definitions of all actions, and obtain the SSA
    $\square[a]F_j(\vec{x_j}) \equiv \gamma^+_{F_i} \wedge \vec{\tau_j}(\vec{x_j}) \vee F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j}$;

Simple Actions

Mapping ADL problems to BAT in $\mathcal{ES}$ follows (Claßen *et al.* 07), (numerical) functional fluents are handled similarly.

- Initial Database Σ_0: Initial world + Typing
- Precondition axiom Σ_{pre}: Case disjunction over all operators
- Successor state axioms Σ_{post}
  - For each fluent predicate $F_j(\vec{x_j})$, extract the positive condition $\gamma^+_{F_j}$ and the negative condition $\gamma^-_{F_j}$ from the effect definitions of all actions, and obtain the SSA
    $\Box[a]F_j(\vec{x_j}) \equiv \gamma^+_{F_i} \wedge \vec{\tau_j}(\vec{x_j}) \vee F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j};$
  - For each fluent function $f_j(\vec{x_j})$, extract the update condition $\gamma^v_{f_j}$ from effect definitions of all actions, and obtain the SSA
    $\Box[a]f_j(\vec{x_j}) = y_j \equiv \gamma^v_{f_j} \wedge \vec{\tau_j}(\vec{x_j}) \vee f_j(\vec{x_j}) = y_j \wedge \neg\exists y'.\,(\gamma^v_{f_j})^{y_j}_{y'}$

Durative Actions

For each PDDL durative action $\widetilde{A}(\vec{x})$, map "`at start`" conditions and effects to $start(\widetilde{A}(\vec{x}), t)$, and "`at end`" ones to $end(\widetilde{A}(\vec{x}), t)$.

```
(:durative-action drive
 :parameters (?v - vehicle ?l1 ?l2 - location)
 :duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
 :precondition (and (at start (at ?v ?l1))
                    (over all (power ?v)))
 :effect (and (at start (not (at ?v ?l1)))
              (when (at start (not (engine ?v)))
                    (and (at start (engine ?v))
                         (at end (not (engine ?v)))))
              (at end (at ?v ?l2))
              (at end (increase (miles ?v) (distance ?l1 ?l2)))))
```

$start(drive(v, l_1, l_2), t)$
$end(drive(v, l_1, l_2), t)$

Durative Actions

For each PDDL durative action $\widetilde{A}(\vec{x})$, map "at start" conditions and effects to $start(\widetilde{A}(\vec{x}), t)$, and "at end" ones to $end(\widetilde{A}(\vec{x}), t)$.

```
(:durative-action drive
 :parameters (?v - vehicle ?l1 ?l2 - location)
 :duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
 :precondition (and (at start (at ?v ?l1))
                    (over all (power ?v)))
 :effect (and (at start (not (at ?v ?l1)))
              (when (at start (not (engine ?v)))
                    (and (at start (engine ?v))
                         (at end (not (engine ?v)))))
              (at end (at ?v ?l2))
              (at end (increase (miles ?v) (distance ?l1 ?l2)))))
```

$start(drive(v, l_1, l_2), t)$

$end(drive(v, l_1, l_2), t)$

Durative Actions

For each PDDL durative action $\widetilde{A}(\vec{x})$, map "`at start`" conditions and effects to $start(\widetilde{A}(\vec{x}), t)$, and "`at end`" ones to $end(\widetilde{A}(\vec{x}), t)$.

```
(:durative-action drive
  :parameters (?v - vehicle ?l1 ?l2 - location)
  :duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
  :precondition (and (at start (at ?v ?l1))
                     (over all (power ?v)))
  :effect (and (at start (not (at ?v ?l1)))
               (when (at start (not (engine ?v)))
                     (and (at start (engine ?v))
                          (at end (not (engine ?v)))))
               (at end (at ?v ?l2))
               (at end (increase (miles ?v) (distance ?l1 ?l2)))))
```

$start(drive(v, l_1, l_2), t)$

$end(drive(v, l_1, l_2), t)$

Durative Actions

For each PDDL durative action $\widetilde{A}(\vec{x})$, map "at start" conditions and effects to $start(\widetilde{A}(\vec{x}), t)$, and "at end" ones to $end(\widetilde{A}(\vec{x}), t)$.

```
(:durative-action drive
  :parameters (?v - vehicle ?l1 ?l2 - location)
  :duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
  :precondition (and (at start (at ?v ?l1))
                     (over all (power ?v)))
  :effect (and (at start (not (at ?v ?l1)))
               (when (at start (not (engine ?v)))
                     (and (at start (engine ?v))
                          (at end (not (engine ?v)))))
               (at end (at ?v ?l2))
               (at end (increase (miles ?v) (distance ?l1 ?l2)))))
```

$start(drive(v, l_1, l_2), t)$
$end(drive(v, l_1, l_2), t)$

Problems: Duration constraint, invariant condition and inter-temporal conditional effect are ignored.

Durative Actions

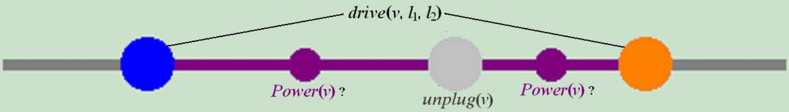
For each PDDL durative action $\widetilde{A}(\vec{x})$, map "at start" conditions and effects to $start(\widetilde{A}(\vec{x}), t)$, and "at end" ones to $end(\widetilde{A}(\vec{x}), t)$.

```
(:durative-action drive
 :parameters (?v - vehicle ?l1 ?l2 - location)
 :duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
 :precondition (and (at start (at ?v ?l1))
                    (over all (power ?v)))
 :effect (and (at start (not (at ?v ?l1)))
              (when (at start (not (engine ?v)))
                    (and (at start (engine ?v))
                         (at end (not (engine ?v)))))
              (at end (at ?v ?l2))
              (at end (increase (miles ?v) (distance ?l1 ?l2)))))
```

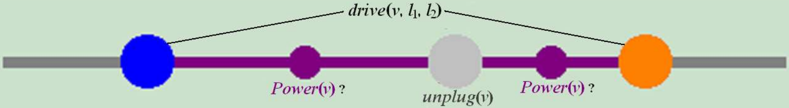
$start(drive(v, l_1, l_2), t)$
$end(drive(v, l_1, l_2), t)$

Problems: Duration constraint, invariant condition and inter-temporal conditional effect are ignored.

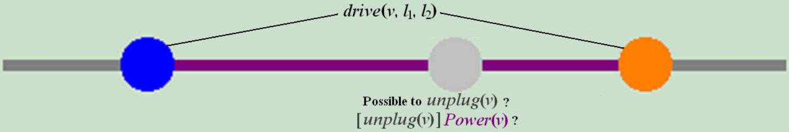
Invariant Condition



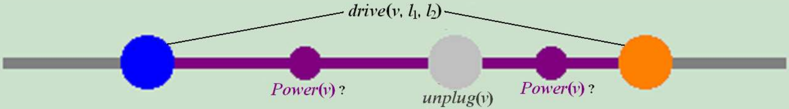
Invariant Condition



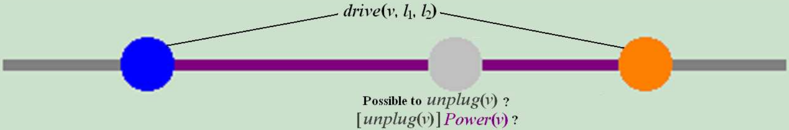
Protect the invariant condition of $drive(v, l_1, l_2)$ by not allowing actions that violate it (*e.g.* $unplug(v)$) to happen.



Invariant Condition



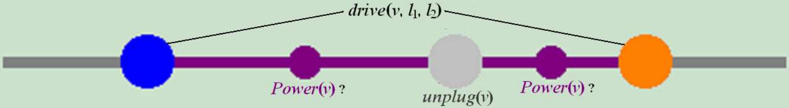
Protect the invariant condition of $drive(v, l_1, l_2)$ by not allowing actions that violate it (*e.g.* $unplug(v)$) to happen.



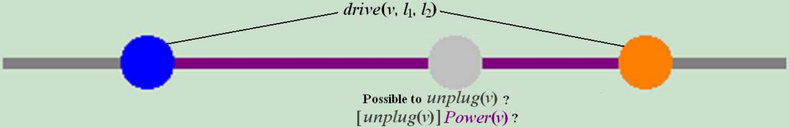
Formally, assert

$$\Box Poss(a) \supset [a]\big(Performing(drive(v, l_1, l_2)) \supset Power(v)\big)$$

Invariant Condition



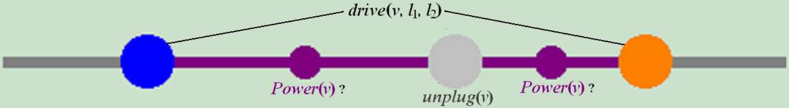
Protect the invariant condition of $drive(v, l_1, l_2)$ by not allowing actions that violate it (*e.g.* $unplug(v)$) to happen.



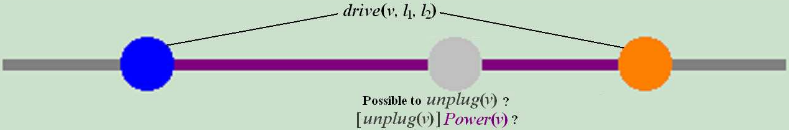
Formally, assert

$$\Box Poss(a) \supset \mathcal{R}[a, Performing(drive(v, l_1, l_2)) \supset Power(v)]$$

Invariant Condition



Protect the invariant condition of $drive(v, l_1, l_2)$ by not allowing actions that violate it (*e.g.* $unplug(v)$) to happen.

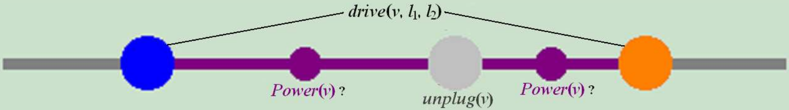


Formally, assert

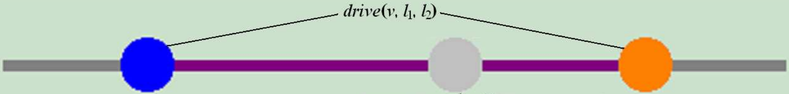
$$\Box Poss(a) \supset \mathcal{R}[a, Performing(drive(v, l_1, l_2)) \supset Power(v)] \iff$$

$$[\exists t.\, a = start(drive(v, l_1, l_2), t) \vee Performing(drive(v, l_1, l_2)) \wedge \neg\exists t.\, a = end(drive(v, l_1, l_2), t)] \supset [\textsc{False} \vee Power(v) \wedge \neg\exists t.\, a = unplug(v, t)]$$

Invariant Condition



Protect the invariant condition of $drive(v, l_1, l_2)$ by not allowing actions that violate it (*e.g.* $unplug(v)$) to happen.



Formally, assert

$$\Box Poss(a) \supset \{$$

$$[\exists t.\, a = start(drive(v, l_1, l_2), t) \vee Performing(drive(v, l_1, l_2)) \wedge$$
$$\neg\exists t.\, a = end(drive(v, l_1, l_2), t)] \supset [\textsc{False} \vee Power(v) \wedge \neg\exists t.\, a = unplug(v, t)]$$

$$\}$$

Other Features

- Concurrency

  Interleaved concurrency

  e.g. $[unplug(car, 5)][unplug(truck, 5)]\neg Power(car)$

- Continuous effects

  Introduce linear functions to BAT (Grosskreutz & Lakemeyer 00)

- Timed initial literals

  e.g. `(at 2 (engine truck))`

  introduce a new and unique action *turn_on_engine* with $Poss(turn\_on\_engine(2)) \wedge Obli(turn\_on\_engine(2))$ and with the single effect to make $Engine(truck)$ true.

- Start duration constraint

  "Remember" the involved function values at the start event, and assert the constraint at the end.

- Invariant condition with continuous effect

  Force execution of *end* action.

Correctness

Theorem

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 03) iff there is a linearization $\langle r_1, \cdots, r_k \rangle$ of P such that

$$\Sigma \models [r_1] \cdots [r_k] \big(\mathit{Executable} \wedge \psi \wedge \neg \exists a. \mathit{Performing}(a) \big)$$

Conclusion and Future Work

Contribution

- We present a situation-calculus semantics for the temporal fragment of PDDL;
- Theoretical ground for relating PDDL to other situation calculus based formalisms, *e.g.* Golog;
- Offer an alternative view on temporal planning,
  (*e.g.* Cushing *et al.* (2007) observe that most *state-of-the-art* planners are temporally simple.)

Ongoing and future work

- Temporally-expressive planner based on the semantics;
- Extend the result to include preferences and constraints on plan trajectory in PDDL 3.0.