A Declarative Semantics for a Subset of PDDL with Time and Concurrency

Master Thesis Project in RWTH Aachen, Germany

Yuxiao (Toby) Hu

PT263
yuxiao@cs.toronto.edu

April 20, 2007

Outline

Background and Motivation

- Planning and action languages have developed independently
  - Planning
    - Given an initial state, a set of action operators and a goal state, find a sequence of actions to achieve the goal.
    - Efficient but less expressive
  - Action formalisms
    - Concentrate more on the underlying logic
    - Expressive but less efficient
- Goal: Bring the two together, *e.g.* planners in GOLOG
  1. GOLOG generates a sub-goal
  2. PDDL as interface
  3. Solve the sub-goal with an external planner
  4. Return the plan to GOLOG
- Needed: A declarative semantics of PDDL

The Situation Calculus

- A dialect of FOL for reasoning about dynamic world.
- First introduced by McCarthy and later refined by Reiter.
- Syntax:

$$\neg Holding(Obj_5, S_0) \wedge Holding\Big(Obj_5, do\big(pickup(Obj_5), S_0\big)\Big)$$

- Basic action theory $\Sigma = \mathcal{FA} \cup \Sigma_{una} \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0$
  - $\mathcal{FA}$: foundational axioms for situations
  - Σ_{una}: unique names axioms
  - Σ_{pre}: precondition axiom $Poss\big(A(\vec{x}), s\big) \equiv \pi_A(\vec{x}, s)$
  - Σ_{post}: successor state axioms $F\big(\vec{x}, do(a, s)\big) \equiv \Phi_F(\vec{x}, a, s)$ or $f\big(\vec{x}, do(a, s)\big) = y \equiv \Phi_f(\vec{x}, y, a, s)$
  - Σ_0: initial database
- Theoretical foundation of action language GOLOG

The Logic $\mathcal{ES}$

- Introduced by Lakemeyer and Levesque 2004
- An extension of the logic $\mathcal{OL}$ with useful fragment of the situation calculus
- Simplifies definitions and proofs in situation calculus
  - Fixed universe of discourse: Countably infinite standard names (objects, actions)
  - No explicit situation terms
    - $[a]\alpha$: α holds after action a
    - $\Box\alpha$: α holds after any sequence of actions
  - Basic action theory $\Sigma = \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0$
    - Σ_{pre}: precondition axiom $\Box Poss(a) \equiv \Pi$
    - Σ_{post}: successor state axiom $\Box[a]F(\vec{x}) \equiv \Phi_F(\vec{x}, a)$
    - Σ_0: initial database
- Powerful enough to capture GOLOG

Regression and Progression

Two solutions to evaluate a formula mentioning non-initial situations (the projection problem):

- *Regression*:
  Transform the sentence to an equivalent one (wrt the BAT) mentioning only the initial situation, and evaluate the transformed formula, *e.g.*
  $$\mathcal{R}[z \cdot t, F(t_1, \cdots, t_k)] \equiv \mathcal{R}[z, (\gamma_F)^{a\ x_1 \cdots x_k}_{t\ t_1 \cdots t_k}]$$
- *Progression*:
  Update the initial database through actions, and evaluate the formula with the current database.
  Σ_r is the progression of Σ_0 through r iff
  1. Σ_r is in $\langle r \rangle$
  2. $\Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0 \models \Sigma_r$
  3. Observers in future situations cannot differentiate between models of Σ_0 and of Σ_r

The Development of PDDL

PDDL: The Planning Domain Definition Language

- Ancestors:
  - STRIPS (Fikes and Nilsson 1971)
    - State: a set of formulas in FOL
    - Operator: name, precondition, add- and delete list
  - ADL (Pednault 1989)
    - Typed objects, built-in equality, conditional effects
    - Negation, disjunction and quantification in conditions
- Version 1.2 (McDermott *et al.* 1998)
- Version 2.1 (Fox and Long 2003)
  - Numerics, plan metrics
  - Time, concurrency and durative actions
  - A formal state-transitional semantics
- Version 2.2 (Edelkamp and Hoffmann 2004)
  - Derived predicates and timed initial literals
- Version 3.0 (Gerevini and Long 2005)
  - Constraints and preferences on plans

An Example

The Electro-Vehicle Domain

- Actions:
  - $drive(v, l_1, l_2)$ is a durative action with duration $\frac{distance(l_1, l_2)}{velocity(v)}$ and overall precondition $power(v) > 0$. If engine initially off, turn it on at start and off again at end.
  - $unplug(v)$ is a simple action that makes $power(v) = 0$
- Predicates:
  - $At(v, l)$: vehicle v is at location l
  - $Engine(v)$: the engine of vehicle v is on
- Functions:
  - $miles(v)$: the covered distance of vehicle v
  - $power(v)$: the remaining power of vehicle v
  - $velocity(v)$: the velocity of vehicle v
  - $distance(l_1, l_2)$: the distance between locations l_1 and l_2

PDDL Domain Definition

```
(:define (domain electro-car)
  (:requirements :adl :durative-actions)
  (:types vehicle location)
  (:predicates (at ?v - vehicle ?l - location) (engine ?v - vehicle))
  (:functions (power ?v - vehicle) (miles ?v - vehicle) ···)
  (:action unplug
    :parameters (?v - vehicle)
    :effect (assign (power ?v) 0))
  (: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) 0)))
    :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)))))
)
```

PDDL Domain Definition

```
(:define (domain electro-car)
  (:requirements :adl :durative-actions)
  (:types vehicle location)
  (:predicates (at ?v - vehicle ?l - location) (engine ?v - vehicle))
  (:functions (power ?v - vehicle) (miles ?v - vehicle) ···)
  (:action unplug
    :parameters (?v - vehicle)
    :effect (assign (power ?v) 0))
  (: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) 0)))
    :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)))))
)
```

PDDL Domain Definition

```
(:define (domain electro-car)
  (:requirements :adl :durative-actions)
  (:types vehicle location)
  (:predicates (at ?v - vehicle ?l - location) (engine ?v - vehicle))
  (:functions (power ?v - vehicle) (miles ?v - vehicle) ···)
  (:action unplug
    :parameters (?v - vehicle)
    :effect (assign (power ?v) 0))
  (: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) 0)))
    :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)))))
)
```

PDDL Domain Definition

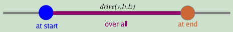
```
(:define (domain electro-car)
  (:requirements :adl :durative-actions)
  (:types vehicle location)
  (:predicates (at ?v - vehicle ?l - location) (engine ?v - vehicle))
  (:functions (power ?v - vehicle) (miles ?v - vehicle) ··· )
  (:action unplug
    :parameters (?v - vehicle)
    :effect (assign (power ?v) 0))
  (: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) 0)))
    :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)))))
)
```

PDDL Domain Definition

```
(:define (domain electro-car)
  (:requirements :adl :durative-actions)
  (:types vehicle location)
  (:predicates (at ?v - vehicle ?l - location) (engine ?v - vehicle))
  (:functions (power ?v - vehicle) (miles ?v - vehicle) ···)
  (:action unplug
    :parameters (?v - vehicle)
    :effect (assign (power ?v) 0))
  (: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) 0)))
    :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)))))
)
```

PDDL Domain Definition

```
(:define (domain electro-car)
  (:requirements :adl :durative-actions)
  (:types vehicle location)
  (:predicates (at ?v - vehicle ?l - location) (engine ?v - vehicle))
  (:functions (power ?v - vehicle) (miles ?v - vehicle) ···)
  (:action unplug
    :parameters (?v - vehicle)
    :effect (assign (power ?v) 0))
  (: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) 0)))
    :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)))))
)
```



PDDL Problem Definition

```
(:define (problem my-car)
  (:domain electro-car)
  (:objects car truck - vehicle
            home office factory - location)
  (:init (and (at car home)               (at truck factory)
              (= (miles car) 0)           (= (miles truck) 0)
              (= (power car) 1)           (= (power truck) 0)
              (= (velocity car) 10)       (= (velocity truck) 5)
              (= (distance home office) 5000)
              (= (distance factory office) 20000)
              (at 2 (engine truck))))
  (:goal (at car office))
  (:metric minimize (+ (miles car) (miles truck)))
)
```

Existing Work and Project Goal

Existing declarative semantics for subsets of PDDL

1. Relational STRIPS: a mechanism to (first-order) progress basic action theories with complete initial database and strongly context free successor state axioms (Lin and Reiter 1997)
2. ADL subset of PDDL: first-order progression in $\mathcal{ES}$ (Claßen *et al.* 2007)

Extend the results above with the advanced features in the larger subset of PDDL

- Numerics and Metrics
- Durative Actions
- Timed Initial Literals

The ADL Subset

Mapping ADL problems to BAT in $\mathcal{ES}$ (Claßen *et al.* 2007)

- Successor state axioms Σ_{post}
  For a fluent predicate, *e.g.* $At(v, l_1, l_2)$
  - γ^+_{At}: the condition to make $At(v, l)$ true
  - γ^-_{At}: the condition to make $At(v, l)$ false

  $\Box[a]At(v, l) \equiv \gamma^+_{At} \vee At(v, l) \wedge \neg\gamma^-_{At}$
- Precondition axiom Σ_{pre}
  Case disjunction over all operators
  $\Box Poss(a) \equiv a = A_1 \wedge \pi_{A_1} \vee \cdots \vee a = A_n \wedge \pi_{A_n}$
- Initial Database Σ_0
  - Initial world:
    $At(v, l) \equiv (v = car \wedge l = home) \vee (v = trck \wedge l = factory)$
  - Typing:
    $At(v, l) \supset Vehicle(v) \wedge Location(l)$
    $Vehicle(v) \equiv v = car \vee v = truck$

Numerical Expressions

Numerical functions are modeled by normal fluent functions in $\mathcal{ES}$, and the existing axiomatization of numbers is used.

- Successor state axioms Σ_{post}:

$$\begin{aligned}\gamma^{v}_{power} &= (\exists y_1.a = unplug(y_1) \wedge x_1 = y_1 \wedge y = 0)\\ \gamma_{power} &= (\exists y_1.a = unplug(y_1) \wedge x_1 = y_1)\end{aligned}$$

  So the axiom for *power* is

$$\Box[a]power(x_1) = y \equiv \gamma^{v}_{power} \wedge Car(x_1) \wedge Number(y) \vee power(x_1) = y \wedge \neg\gamma_{power}$$

- Precondition axiom Σ_{pre}: same as before, except that comparison between numerical expressions allowed
- Initial database Σ_0: has additionally the following for *power*

$$power(x_1) = y \equiv (x_1 = car_1 \wedge y = 0.6) \vee (x_1 = car_2 \wedge y = 0.8)$$

Temporal Extensions to $\mathcal{ES}$

Similar to Pinto and Reiter's work in the situation calculus.

- 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: $\Box Poss(a) \supset now \leq time(a)$
- Concurrency: Interleaved concurrency. For example:
  $$[unplug(car, 5)][unplug(truck, 5)]$$
- Durative Actions:
  - Situation calculus: $startWalk(x, y, t)$, $endWalk(t)$, $Walking(x, y, s)$
  - In $\mathcal{ES}$: $start(walk(x, y), t)$, $end(walk(x, y), t)$, $Performing(walk(x, y))$, $since(walk(x, y))$

A Simple Mapping

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

```
(: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) 0)))
  :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)))))
```

$\mathit{start}(\mathit{drive}(v, l_1, l_2), t)$
$\mathit{end}(\mathit{drive}(v, l_1, l_2), t)$

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

A Simple Mapping

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) 0)))
 :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.

A Simple Mapping

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) 0)))
  :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.

A Simple Mapping

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 $end(\widetilde{A}(\vec{x}), t)$.

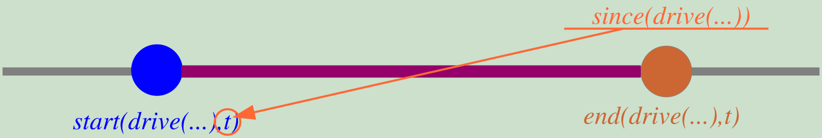
```
(:durative-action drive
  :parameters (?v - vehicle ?l1 ?l2 - location)
  :duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
  :precondition (and (at start (at ?v ?l1))
                     (over all (> (power ?v) 0)))
  :effect (and (at start (not (at ?v ?l1)))
               (when (at 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.

Duration Constraint

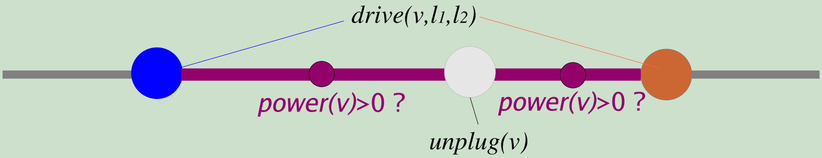
```
:duration (= ?duration (/ (distance ?l1 ?l2) (velocity ?v)))
```



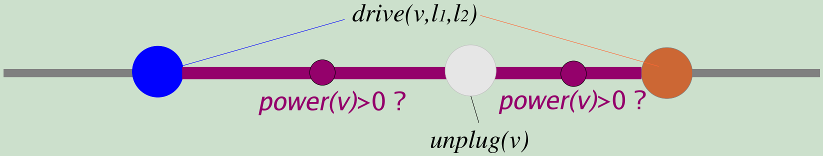
- The duration of $drive(v, l_1, l_2)$ can be obtained at its end event $end(drive(v, l_1, l_2), t)$ by $(t - since(drive(v, l_1, l_2)))$
- Assert in the precondition

$$\Box Poss(end(drive(v, l_1, l_2), t)) \supset$$
$$t - since(drive(v, l_1, l_2)) = distance(l_1, l_2)/velocity(v)$$

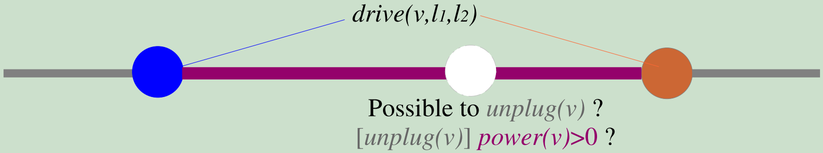
Invariant Condition



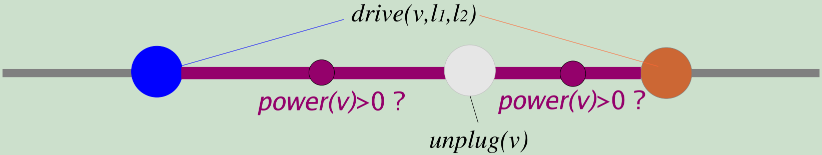
Invariant Condition



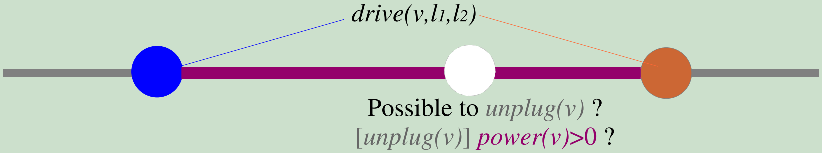
Protect the invariant condition of $drive(v, l_1, l_2)$ by not allowing actions that violate it (*e.g.* $unplug(v)$) to happen.



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.



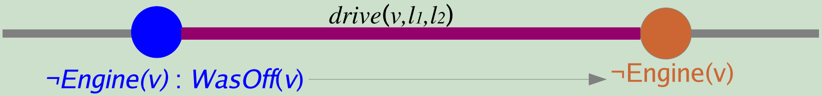
Formally, assert

$$\Box Poss(a) \supset \mathcal{R}[a, Performing(drive(v, l_1, l_2)) \supset power(v) > 0]$$

Inter-Temporal Conditional Effect

```
(when (at start (not (engine ?v)))
      (at end (not (engine ?v))))
```

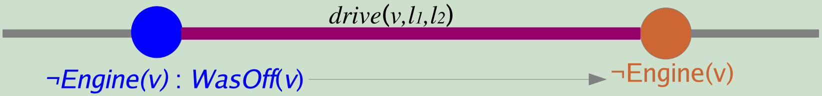
has "`at start`" premise but "`at end`" effect.
To "remember" the old state of *Engine*(v), introduce a new and unique fluent predicate *WasOff*(v)



Inter-Temporal Conditional Effect

```
(when (at start (not (engine ?v)))
      (at end (not (engine ?v))))
```

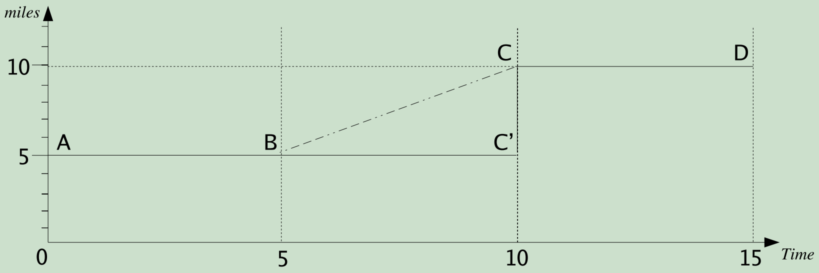
has "`at start`" premise but "`at end`" effect.
To "remember" the old state of $Engine(v)$, introduce a new and unique fluent predicate $WasOff(v)$



Then, the successor state axiom for $Engine$ is

$$\Box[a]Engine(v) \equiv$$
$$\exists l_1, l_2, t.a = start(drive(v, l_1, l_2), t) \wedge \neg Engine(v) \vee$$
$$Engine(v) \wedge \neg(\exists l_1, l_2, t.a = end(drive(v, l_1, l_2), t) \wedge WasOff(v))$$

Continuous Effects: The Necessity



- Discretized numerical effects: $(\overline{ABC'CD})$
  `(at end (increase (miles ?v) (distance ?l1 ?l2)))`
- Continuous numerical effects: $(\overline{ABCD})$
  `(increase (miles ?v) (* #t (velocity ?v)))`

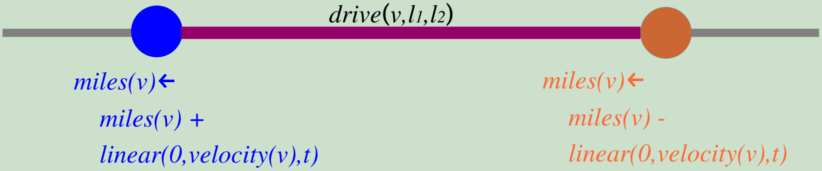
Need to map continuous effects to BAT in $\mathcal{ES}$.

Continuous Extensions to $\mathcal{ES}$ and the Solution

Like (Grosskreutz and Lakemeyer 2000), define

$$\begin{aligned} linear(x_0, v_0, v_0) &\triangleq x_0 + v_0 \cdot (t - t_0) \\ eval(x,t) = y &\equiv \exists x_0, v_0, t_0 . x = linear(x_0, v_0, t_0) \wedge y = x_0 + v_0 \cdot (t - t_0) \vee \\ &\quad \forall x_0, v_0, t_0 . x \neq linear(x_0, v_0, t_0) \wedge y = x \end{aligned}$$

and axiomatize operations with *linear*'s.


drive(v,l1,l2)
miles(v)← miles(v) + linear(0,velocity(v),t)
miles(v)← miles(v) - linear(0,velocity(v),t)

Then, we may have $\Sigma \models [start(drive(car_1, l_1, l_2), 5)]eval(miles(v), 8) = 8$.

Invariant Conditions with Continuous Effects

The problem

- Discrete case: protect invariants by not allowing violating actions to happen
- Continuous case: invariants may be violated without any action happening! *e.g.*
  `(decrease (power ?v) (* #t (consume-rate ?v)))`

Solution

- Introduce obligatory actions to $\mathcal{ES}$ BAT (Similar to "natural actions" in situation calculus)
    - $\Box Obli(a) \equiv \Omega$
    - $\Box Poss(a) \supset Obli(a) \vee$
      $\neg(\exists a'.Obli(a') \wedge now \leq time(a') \leq time(a))$
- Stop a durative action as soon as its invariant is violated
    - $Performing(a) \wedge \neg Eval[\pi_a^o, t] \supset Obli(end(a, t))$

Other Features

- Timed initial literals

  `(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.
- Plan metric

  An expression on normal fluent numerical functions
- Start duration constraint

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

More information at

http://www-users.rwth-aachen.de/Yuxiao.Hu/projects/thesis/final/thesis1122.pdf

Correctness

- Due to the process-related properties, mere progression is not enough.
- Extract these properties from the state and the plan, which corresponds to the auxiliary properties in the BAT, such as now, $Performing(drive(v, l_1, l_2))$, $WasOff(v)$.
- Single-step update is modeled by first-order progression in $\mathcal{ES}$, including for auxiliary properties.
- Correctness due to the fact that the semantics by Fox and Long and ours share the same set of valid plans.

Problems with Interleaved Concurrency

Interleaved concurrency is simple (no modification to $\mathcal{ES}$ semantics is necessary) and powerful enough to capture almost all the features in PDDL, except the following special cases

1. The invariant condition of a durative action is not satisfied before its start event, and is turned to true by a simultaneously happening action.

Integrating true concurrency in $\mathcal{ES}$ may solve these problems.

Conclusion and Future Work

- Contribution
  1. Explored the possibility to extend the BAT in $\mathcal{ES}$ to model realistic domains.
  2. Defined a declarative semantics for the subset of PDDL with time and concurrency.
- Ongoing and Future Work
  1. Study true concurrency
  2. Encode "temporally expressive" planning problems as CSP based on the semantics
  3. Integrate PDDL planners into temporal concurrent GOLOG.
  4. Consider larger subset of PDDL (derived predicates, constraints and preferences, *etc.*).