Temporally-Expressive Planning as Constraint Satisfaction Problems

Yuxiao Hu

Department of Computer Science
University of Toronto

yuxiao (a) cs toronto edu

September 25, 2007
As shown by Cushing *et al.* (2007), there are “temporally-expressive” planning problems that

- can be represented by PDDL 2.x
- cannot be solved by many *state-of-the-art* planners

This is due to their strong assumptions on

1. temporal annotation
   (Over-all preconditions, at end effects)
2. decision epochs
   (An action can happen only when another event is happening)
As shown by Cushing et al. (2007), there are “temporally-expressive” planning problems that

- can be represented by PDDL 2.x
- cannot be solved by many state-of-the-art planners

This is due to their strong assumptions on

1. temporal annotation
   (Over-all preconditions, at end effects)

2. decision epochs
   (An action can happen only when another event is happening)

Goal: Solve general PDDL planning problems with a unified approach
A Running Example

Fire

burnCandle [≤6]

Fire

(duration=6) → ¬Fire

¬Occupied ∧ Fire

makeWish [ ]

Occupied

¬Occupied

numWish += duration

¬Occupied ∧ Fire

blowCandle

Fire

(numWish ≥ 3) → Happy
A Running Example

- Required concurrency ("makeWish" be contained within "burnCandle")
- Duration inequality and duration-related effects
Our approach consists of two steps:

1. **PDDL $\mapsto$ BAT (Basic Action Theory)**
   Based on a concurrent extension to the situation-calculus semantics of PDDL (Claßen et al. 2007)

2. **BAT $\mapsto$ CSP**
   Encode the BAT into a CSP problem, and solve the CSP to obtain the plan.
Our approach consists of two steps:

1. PDDL $\implies$ BAT (Basic Action Theory)
   Based on a concurrent extension to the situation-calculus semantics of PDDL (Claßen et al. 2007)

2. BAT $\implies$ CSP
   Encode the BAT into a CSP problem, and solve the CSP to obtain the plan.

The intuition behind it is to

- model durative actions with simple (instantaneous) actions
- treat time as a numerical property, and advance it with constraints.
**Concurrent temporal situation calculus** (Reiter 2001) with the following syntax:

- $A(\vec{x}, t)$ denotes the happening of simple action $A(\vec{x})$ at time $t$.
- A durative action $A(\vec{x})$ is represented by:
  - Action $\text{start}(A(\vec{x}), t)$: the start event of $A(\vec{x})$ at time $t$.
  - Action $\text{end}(A(\vec{x}), t)$: the end event of $A(\vec{x})$ at time $t$.
  - Predicate $\text{Performing}(A(\vec{x}))$: whether $A(\vec{x})$ is in progress.
  - Function $\text{since}(A(\vec{x}))$: the last starting time of $A(\vec{x})$.

- $\{a_1, \cdots, a_n\}$ means the concurrent happening of $a_i$.
- $[c] \alpha$ means $\alpha$ holds after a list $c$ of concurrent actions.
- $\Box \alpha$ means $\alpha$ holds in any situation.
- *now* is a special functional fluent representing the current time.
The basic action theory $\Sigma$ consists of

- The initial database $\Sigma_0$ e.g.:
  $$\neg Fire, \ numWish = 0, \ \neg Performing(burnCandle), \ now = 0;$$

- The precondition axiom $\Sigma_{pre}$, obtained from, e.g.:
  $$\Box Poss(end(burnCandle, t)) \supset (t - since(burnCandle)) \leq 6;$$

- The successor state axioms $\Sigma_{post}$, e.g.:
  $$\Box[c]Fire \equiv \exists t.\ start(burnCandle, t) \in c \lor$$
  $$\neg Fire \land \neg (\exists t.\ end(burnCandle, t) \in c \land (t - since(burnCandle)) = 6) \lor$$
  $$\exists t.\ blowCandle(t) \in c);$$

- The unique names axioms $\Sigma_{una};$

- The foundational axioms $\mathcal{F}A$, e.g.:
  $$\Box Poss(c) \supset now < time(c)$$
Planning by encoding the basic action theory into a CSP

- Search for increasing plan length \( n = 1, 2, 3, \cdots \), where “length” means the number of concurrent happenings.
- When searching for a plan of length \( n \), create
  - \( n \) boolean variables for each ground action term \( A(\vec{o}) \):
    \[ A^{(0)}_{\vec{o}}, \ldots, A^{(n-1)}_{\vec{o}} \]
  - \( n + 1 \) boolean variables for each ground predicate \( P(\vec{o}) \):
    \[ P^{(0)}_{\vec{o}}, \ldots, P^{(n)}_{\vec{o}} \]
  - \( n + 1 \) numerical variables for each ground function \( f(\vec{o}) \):
    \[ f^{(0)}_{\vec{o}}, \ldots, f^{(n)}_{\vec{o}} \]
An example of searching for a plan of length 2:
The Variable Structure

An example of searching for a plan of length 2:
Constraints: Initial and Goal States

The 0th fact layer encodes the initial state
- Problem-specific fluents set according to the initial description
- Auxiliary fluents set to 0 (or `FALSE`)

The last fact layer encodes the goal condition
- Problem-specific goals
- All “Performing” variables must be false
**Constraints: Initial and Goal States**

The 0th fact layer encodes the initial state
- Problem-specific fluents set according to the initial description
- Auxiliary fluents set to 0 (or `FALSE`)

The last fact layer encodes the goal condition
- Problem-specific goals
- All “Performing” variables must be false
Constraints: Action Preconditions

For each formula of the form $\square \text{Poss}(A) \supset \pi_A$, construct an action precondition constraint, e.g.:

From (part of) precondition axiom in the BAT

$\square \text{Poss}(%539\text{end(burnCandle, t)}) \supset (t - \text{since(burnCandle)}) \leq 6$

we obtain the action precondition constraint

$e_{\text{end\,burn}}(i) \supset (\text{now}(i+1) - \text{since\,burn}(i)) \leq 6$
Constraints: Action Preconditions

For each formula of the form $\square Poss(A) \supset \pi_A$, construct an action precondition constraint, e.g.:

From (part of) precondition axiom in the BAT

$\square Poss(\text{end}(\text{burnCandle}, t)) \supset (t - \text{since}(\text{burnCandle})) \leq 6$

we obtain the action precondition constraint

$\text{end burn}^{(i)} \supset (\text{now}^{(i+1)} - \text{since burn}^{(i)}) \leq 6$
Constraints: Successor States

For each formula of the form $\Box[c] F \equiv \Phi_F$, construct a successor state constraint, e.g.:

From the successor state axiom in the BAT

$$\Box[c] Fire \equiv \exists t. start(burnCandle, t) \in c \lor$$
$$Fire \land \neg(\exists t. end(burnCandle, t) \in c \land (t - since(burnCandle) = 6) \lor$$
$$\exists t. blowCandle(t) \in c)$$

we obtain the successor state constraint

$$Fire^{(i+1)} \equiv start burn^{(i)} \lor$$
$$Fire^{(i)} \land \neg(end burn^{(i)} \land (now^{(i+1)} - since burn^{(i)} = 6) \lor blowC^{(i)})$$
\[
\text{Fire}^{(i+1)} \equiv s_{\text{start}} \text{burn}^{(i)} \lor \\
\text{Fire}^{(i)} \land \neg (e_{\text{end}} \text{burn}^{(i)} \land (\text{now}^{(i+1)} - s_{\text{ince}} \text{burn}^{(i)} = 6) \lor \text{blowC}^{(i)})
\]
To ensure chronological order of happenings, the foundational axiom in BAT

\[ \Box \text{Poss}(c) \supset \text{now} < \text{time}(c) \]

is encoded as the action happening time constraint

\[ \text{now}^{(i)} < \text{now}^{(i+1)} \]
To ensure chronological order of happenings, the foundational axiom in BAT

$$\Box Poss(c) \supset now < time(c)$$

is encoded as the action happening time constraint

$$now(i) < now(i+1)$$
In addition, we need the following constraints (details in paper):

- Invariant constraints;
- Non-null step constraints;
- Mutex constraints;
- Timed-initial literal enforcement constraints.
We hand-encoded several problem with constraint programming language *Choco* (choco.sourceforge.net), which supports

1. Higher-order constraints, *e.g.* `implies(A, and(B,C));`

Can solve problems with required concurrency, duration inequalities and duration-related effects.
Example Result

\[
\begin{align*}
\text{Fire}^{(0)} &= 0 \\
\text{Occupied}^{(0)} &= 0 \\
\text{Happy}^{(0)} &= 0 \\
\text{numWish}^{(0)} &= 0 \\
\text{Performingburn}^{(0)} &= 0 \\
\text{PerformingWish}^{(0)} &= 0 \\
\text{sinceburn}^{(0)} &= 0 \\
\text{sinceWish}^{(0)} &= 0 \\
\text{now}^{(0)} &= 0 \\
\text{startburn}^{(0)} &= 1 \\
\text{endburn}^{(0)} &= 0 \\
\text{startWish}^{(0)} &= 0 \\
\text{endWish}^{(0)} &= 0 \\
\text{blowC}^{(0)} &= 0 \\
\text{Fire}^{(1)} &= 1 \\
\text{Occupied}^{(1)} &= 0 \\
\text{Happy}^{(1)} &= 0 \\
\text{numWish}^{(1)} &= 0 \\
\text{Performingburn}^{(1)} &= 1 \\
\text{PerformingWish}^{(1)} &= 0 \\
\text{sinceburn}^{(1)} &= 1 \\
\text{sinceWish}^{(1)} &= 0 \\
\text{now}^{(1)} &= 1 \\
\text{startburn}^{(1)} &= 0 \\
\text{endburn}^{(1)} &= 0 \\
\text{startWish}^{(1)} &= 1 \\
\text{endWish}^{(1)} &= 0 \\
\text{blowC}^{(1)} &= 0 \\
\text{Fire}^{(2)} &= 1 \\
\text{Occupied}^{(2)} &= 1 \\
\text{Happy}^{(2)} &= 0 \\
\text{numWish}^{(2)} &= 0 \\
\text{Performingburn}^{(2)} &= 1 \\
\text{PerformingWish}^{(2)} &= 0 \\
\text{sinceburn}^{(2)} &= 1 \\
\text{sinceWish}^{(2)} &= 2 \\
\text{now}^{(2)} &= 2 \\
\text{startburn}^{(2)} &= 0 \\
\text{endburn}^{(2)} &= 0 \\
\text{startWish}^{(2)} &= 0 \\
\text{endWish}^{(2)} &= 1 \\
\text{blowC}^{(2)} &= 0 \\
\text{Fire}^{(3)} &= 1 \\
\text{Occupied}^{(3)} &= 0 \\
\text{Happy}^{(3)} &= 0 \\
\text{numWish}^{(3)} &= 3 \\
\text{Performingburn}^{(3)} &= 1 \\
\text{PerformingWish}^{(3)} &= 0 \\
\text{sinceburn}^{(3)} &= 1 \\
\text{sinceWish}^{(3)} &= 2 \\
\text{now}^{(3)} &= 5 \\
\text{startburn}^{(3)} &= 0 \\
\text{endburn}^{(3)} &= 1 \\
\text{startWish}^{(3)} &= 0 \\
\text{endWish}^{(3)} &= 0 \\
\text{blowC}^{(3)} &= 1 \\
\text{Fire}^{(4)} &= 0 \\
\text{Occupied}^{(4)} &= 0 \\
\text{Happy}^{(4)} &= 1 \\
\text{numWish}^{(4)} &= 3 \\
\text{Performingburn}^{(4)} &= 0 \\
\text{PerformingWish}^{(4)} &= 0 \\
\text{sinceburn}^{(4)} &= 1 \\
\text{sinceWish}^{(4)} &= 2 \\
\text{now}^{(4)} &= 6
\end{align*}
\]
Example Result

PDDL plan: \{ (1 : burnCandle[5]), (2 : makeWish[3]), (6 : blowCandle) \}
Other Experiments
Other Experiments

- No comparison with state-of-the-art planners available yet.
- Presumably slower on existing (temporally-simple) benchmarks, since we have only focused on generality, not yet efficiency.
Conclusion and Future Work

We have

1. extended a declarative semantics of PDDL with true concurrency
2. proposed a general solution to PDDL planning problems based on a CSP encoding of the declarative semantics

- Handles arbitrary PDDL temporal annotations
- Determines happening times by satisfying constraints
- Solves temporally-expressive PDDL problems (possibly with duration-related constraints and effects) in a unified search.
Conclusion and Future Work

We have

1. extended a declarative semantics of PDDL with true concurrency
2. proposed a general solution to PDDL planning problems based on a CSP encoding of the declarative semantics
   - Handles arbitrary PDDL temporal annotations
   - Determines happening times by satisfying constraints
   - Solves \textit{temporally-expressive} PDDL problems (possibly with duration-related constraints and effects) in a \textit{unified search}.

Future work:

1. Implementation and optimization of automatic translation
2. Constraints and preferences in PDDL 3.0