

# Optimality Properties of Planning via Petri Net Unfolding: A Formal Analysis

Sebastian Sardina and Sarah Hickmott

RMIT University  
Australia

ICAPS 2009

# Planning via Directed Unfolding

1. Forward search partial order planning
  - STRIPS, SAS+
  - Synthesizes a partially ordered plan  $\langle A, \prec \rangle$ 
    - E.g.  $\pi = \langle \{ o1, o2, o3 \}, \{ o1 < o3 \} \rangle$
    - True concurrency semantics

# Planning via Directed Unfolding

## 1. Forward search partial order planning

- STRIPS, SAS+
- Synthesizes a partially ordered plan  $\langle A, \prec \rangle$ 
  - E.g.  $\pi = \langle \{ o1, o2, o3 \}, \{ o1 < o3 \} \rangle$
  - True concurrency semantics

## 2. Advantages:

- Explicit concurrency and causal relations  $\implies$  decomposition
- Notion of a state  $\implies$  state-based heuristics to guide and prune
- Parallel plans  $\implies$  “faster and more flexible”

# Planning via Directed Unfolding

## 1. Forward search partial order planning

- STRIPS, SAS+
- Synthesizes a partially ordered plan  $\langle A, \prec \rangle$ 
  - E.g.  $\pi = \langle \{ o1, o2, o3 \}, \{ o1 < o3 \} \rangle$
  - True concurrency semantics

## 2. Advantages:

- Explicit concurrency and causal relations  $\implies$  decomposition
- Notion of a state  $\implies$  state-based heuristics to guide and prune
- Parallel plans  $\implies$  “faster and more flexible”

## 3. However:

- Lack understanding of the partial order semantics accounted by:
  - solution space;
  - plans generated.

**$\therefore$  How concurrent are the plans obtained by this approach?**

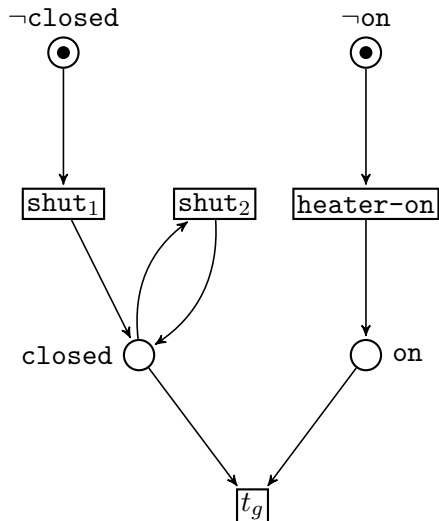
# Talk Overview

## Planning Via Unfolding

1. What it is
2. Concurrency Semantics
3. Optimality properties wrt flexibility and execution time

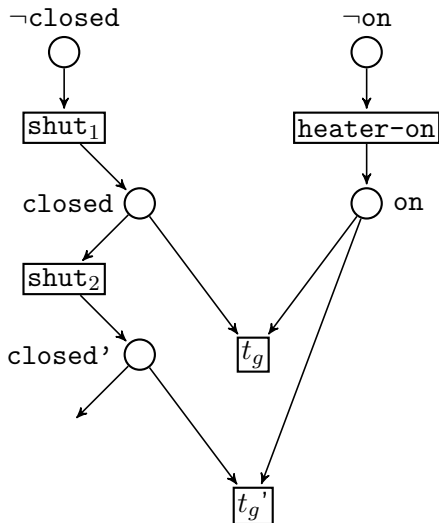
# 1. Cast to Petri net executability problem

- ▶ Variable-value assignments  $\rightarrow$  Places (circles)
- ▶ Operators  $\rightarrow$  Transitions (boxes)
- ▶ State  $\rightarrow$  Tokens (dots)
- ▶ Goal transition  $t_g$
- ▶ Example  
shut-door =  $\langle \{ \}, \{ \text{closed} \} \rangle$   
heater-on =  $\langle \{ \neg\text{on} \}, \{ \text{on} \} \rangle$



## 2. Unfold Petri net to find $t_g$

- ▶ Exact reachability analysis
- ▶ Preserves and exploits causal relations in the Petri net structure
- ▶ Unrolls the space of parallel plans captured by the Petri net
- ▶ **Directed Unfolding:**
  - Value function + planning heuristics
  - Prune and guide toward solution plan



# Concurrency Semantics

1. What is the concurrency semantics of plans synthesised using this approach?
  - What are the restrictions on two actions executing concurrently?
2. How does it compare to the standard notion of concurrency induced by Smith and Weld's [1999] definition of independent actions?

# Independent Actions

Two actions are **independent** iff

1. Their effects don't contradict
2. Their preconditions don't contradict
3. The preconditions for one aren't clobbered by the effect of the other.

## Example

have-cake =  $\langle \{ \text{cake} \}, \{ \} \rangle$       eat-cake =  $\langle \{ \}, \{ \neg \text{cake} \} \rangle$

By restriction 3, not independent actions

# Independent Actions

Two actions are **independent** iff

1. Their effects don't contradict
2. Their preconditions don't contradict
3. The preconditions for one aren't clobbered by the effect of the other.

## Example

have-cake =  $\langle \{ \text{cake} \}, \{ \} \rangle$       eat-cake =  $\langle \{ \}, \{ \neg \text{cake} \} \rangle$

By restriction 3, not independent actions

A plan **respects independence** iff for any two non-independent actions  $a$  and  $b$  the plan ensures that either  $a < b$  or  $b < a$ .

## Strongly Independent Actions

Two actions are **strongly independent** in state  $S$  iff

1. They are independent
2. Any *common postcondition* already holds true in state  $S$ .

### Example

`set-x` =  $\langle \{ \}, \{x\} \rangle$ ,      `set-all` =  $\langle \{ \}, \{x, y, z\} \rangle$

Strongly independent in state  $S = \langle x, \dots \rangle$

Not strongly independent in state  $S' = \langle \neg x, \dots \rangle$

## Strongly Independent Actions

Two actions are **strongly independent** in state  $S$  iff

1. They are independent
2. Any *common postcondition* already holds true in state  $S$ .

### Example

`set-x` =  $\langle \{ \}, \{x\} \rangle$ ,      `set-all` =  $\langle \{ \}, \{x, y, z\} \rangle$

Strongly independent in state  $S = \langle x, \dots \rangle$

Not strongly independent in state  $S' = \langle \neg x, \dots \rangle$

A plan **respects strong independence** iff at any possible state  $S$  during plan execution *all possible concurrent actions* are strongly independent in  $S$ .

Related to locks/monitors; read/write access [Hoare 1974]  
Reduces to independence if original operators are toggling

# Unfolding Generates Strongly Independent Plan

## Theorem

*A plan generated via unfolding respects strong independence for the initial state of the planning problem.*

- ▶ But any totally ordered plan will respect strong independence...

# Unfolding Generates Strongly Independent Plan

## Theorem

*A plan generated via unfolding respects strong independence for the initial state of the planning problem.*

- ▶ But any totally ordered plan will respect strong independence...

## Take-home message:

- ▶ Planning via unfolding “conforms” to the strong independence notion of concurrency
- ▶ If the original operators are toggling, then it “conforms” to independence notion of concurrency

# Plan Flexibility

Partially-ordered plans are in principle more **flexible** in that they may avoid over-committing to action orderings

- ▶ Scheduler can have alternative execution realizations to choose from
  - e.g. Need to post-process or adapt a plan for actions with deadlines and earliest release times
- ▶ Execution time may be reduced when actions can be executed in parallel

## Plan validity w.r.t. Strong Independence

A partially ordered plan  $\pi$  is  $\mathcal{P}$ -valid for planning problem  $\mathcal{P}$  iff

- ▶ All linearizations of  $\pi$  solve  $\mathcal{P}$ , and
- ▶  $\pi$  respects strong independence for the initial state of  $\mathcal{P}$ .

## Plan De/reordering

Can we remove (deorder) or change (reorder) the constraints from a plan synthesized via the unfolding approach?

catch-train < cook-dinner < eat-dinner < read-paper

⇓ **deorder** - remove constraints

catch-train < cook-dinner < {eat-dinner, read-paper}

## Plan De/reordering

Can we remove (deorder) or change (reorder) the constraints from a plan synthesized via the unfolding approach?

catch-train < cook-dinner < eat-dinner < read-paper

⇓ **deorder** - remove constraints

catch-train < cook-dinner < {eat-dinner, read-paper}

⇕ **reorder** - change constraints

{catch-train, read paper} < cook-dinner < eat-dinner

## Minimal De/re-ordering

Consider plan  $\pi$  which is  $\mathcal{P}$ -valid:

- ▶  $\pi$  is a **minimal de/re-ordering wrt flexibility** if you can't de/re-order it to reduce the number of constraints and retain  $\mathcal{P}$ -validity.
- ▶  $\pi$  is a **minimal de/re-ordering wrt execution time** if you can't de/re-order it to reduce the execution time and retain  $\mathcal{P}$ -validity.

[Backstrom 1998] gave similar definitions in the context of plans which respect independence.

## Optimality Guarantees (1/2)

### Theorem

*Any plan synthesized via the unfolding approach is a **minimal deordering wrt flexibility**.*

- ▶ i.e No constraint can be removed without rendering the plan invalid.
- ▶ A minimal deordering wrt flexibility  $\Rightarrow$  a minimal deordering wrt execution time.

## Optimality Guarantees (1/2)

### Theorem

*Any plan synthesized via the unfolding approach is a **minimal deordering wrt flexibility**.*

- ▶ i.e No constraint can be removed without rendering the plan invalid.
- ▶ A minimal deordering wrt flexibility  $\Rightarrow$  a minimal deordering wrt execution time.

### Theorem

*All solution plans which are minimally deordered wrt flexibility exist in the unfolding space.*

- ▶ These results extend to all plans in the unfolding space (not necessarily solutions)

i.e. planning via unfolding “conforms” to strong independence

## Optimality Guarantees (2/2)

### Theorem

*If the unfolding is directed to prefer faster plans, then the plan synthesized is a **minimal reordering wrt execution time**.*

- ▶ Reordering a plan to be optimal wrt execution time is (still) NP-hard in the context of strong independence requirements.

In fact the stronger result is proven:

- ▶ Plan which is optimal among all minimal reorderings wrt time.
- ▶ Can not make a faster plan by changing the actions

## In Summary

- ▶ If the original operators are **toggleing** then the unfolding space consists of plans which conform to **independence**.
- Plan(s) with minimum makespan, as defined by Smith and Weld [1999], exist in the unfolding space and can be obtained using an appropriate search procedure.

## In Summary

- ▶ If the original operators are **toggling** then the unfolding space consists of plans which conform to **independence**.
  - Plan(s) with minimum makespan, as defined by Smith and Weld [1999], exist in the unfolding space and can be obtained using an appropriate search procedure.
- ▶ If the original operators are **not toggling**, then the unfolding space consists of plans which conform to **strong independence**.
  - Stronger restrictions on concurrent execution than independence
  - Analogous to kind of concurrency captured by monitors for thread synchronisation.

## References

[Backstrom 1998 ]

Backstrom, C. 1998. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research* 9:99137.

[Hoare 1974 ]

Hoare, C. A. 1974. Monitors: an operating system structuring concept. *Communications of the ACM* 17(10):549557.