sfb transregio 62
Companion Technology

POP $\equiv$ POCL, right?

Complexity Results for Partial Order (Causal Link) Makespan Minimization

Pascal Bercher and Conny Olz

Institute of Artificial Intelligence, Ulm University, Germany

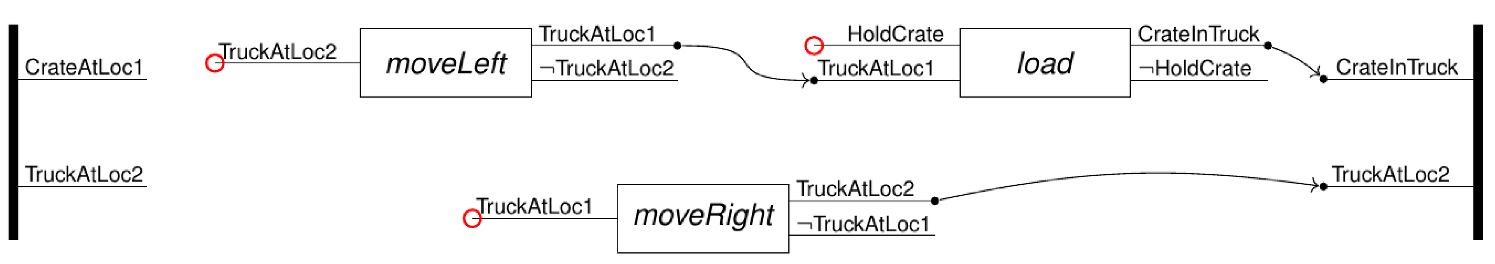
pascal.bercher@anu.edu.au and conny.olz@uni-ulm.de

What is POCL?

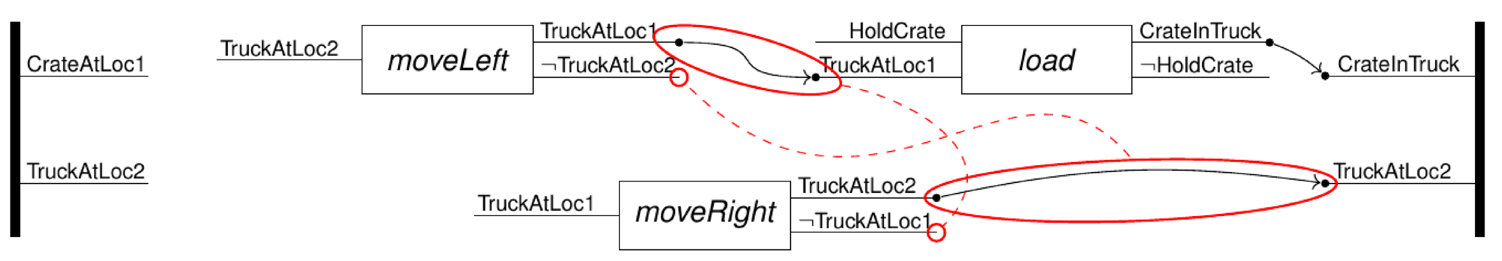
- POCL Planning = Partial Order Causal Link Planning
  - A planning technique that was prominent in the 80s to late 90s based on: *search in the space of partial plans*
  - The search algorithm performs a regression-like search
    → but this is not important for the paper! It focuses on the data structure *partial plans* rather than on search.
- So what is a POCL plan?
  - A partially ordered set of actions.
  - Preconditions and effects might be "connected" with causal links. The respective actions are then called *producer* and *consumer*, respectively.
  - Causal links *protect* their condition: no task may be ordered in between a link. (See graphical example)
- When is a POCL plan a solution?
  - If all preconditions are protected by a causal link
    → no *open precondition flaws*.
  - No causal link can be invalidated
    → no *causal threat flaws*.
  - → See the next example

A POCL plan that's not a solution yet:

... because of unprotected (open) preconditions:



... and because of causal threats:



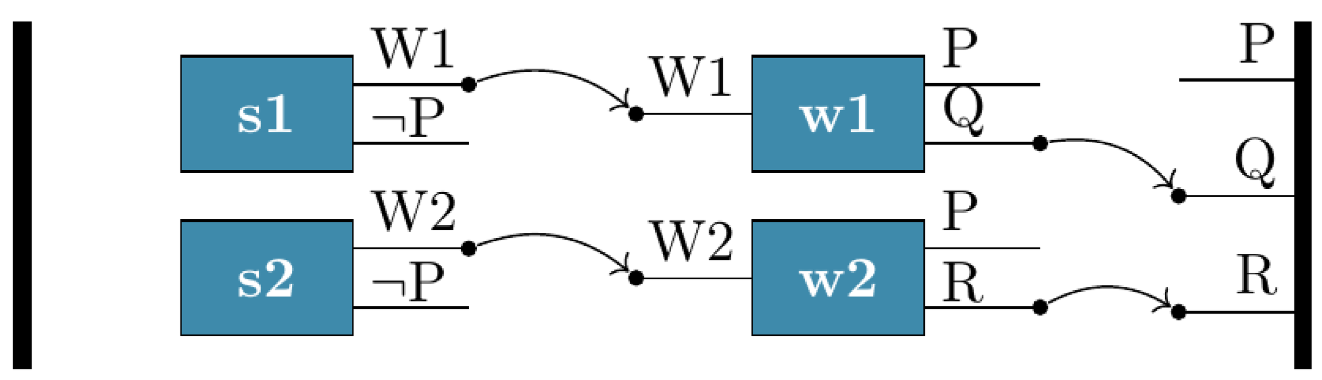
What is POP?

- POP = Partial Order Planning (i.e., synonym to POCL Planning)
- **Here, it means *Partial Order Plan!*** These PO plans are essentially POCL plans *without causal links.*
- So what is a PO plan?
  - A partially ordered set of actions.
  - That's it – no links exist.
- When is a PO plan a solution?
  - If every linearization is executable. (I.e. every linearization is a solution in the standard classical sense.)
  - This can be checked in polynomial time (see the paper, as this is quite complicated).

POP vs. POCL: Is there any *true* difference?

- Yes! (Which is important in theory and practice!)
  - Every POCL solution *is* a PO solution *(just delete the links)*
  - But to turn a PO solution into a POCL solution we might have to change the ordering constraints!
  - → Thus the resulting plans are not equivalent anymore! See the next example.

Example by Kambhampati:



Properties of this POCL plan:

- It has 6 linearizations.
- All linearizations are executable, so it is a PO solution (just delete the links).
- It's not yet a POCL solution: the precondition P is still open.
- Inserting a missing link will cause a causal threat! So we will have to insert orderings.

→ For this PO plan there does not exist a POCL plan with the same linearizations!

POP $\not\equiv$ POCL: So What?

- This has become forgotten, so there are many (sometimes "sloppy") formulations stating POCL plans to be equivalent to PO plans.
- Theorems or even algorithms might be wrong!
  - Clearly, all algorithms that optimize with regard to ordering constraints need to be aware of any difference.
  - What if an algorithm aims for makespan-optimal plans, but searches in the space of POCL plans? Can it be optimal? Do we *know* this yet?
  - See the paper for a deeper discussion.
- It raises an open question: Does for each PO plan exist a POCL plan with the same makespan? This question is practically relevant as only if the answer is *yes* we are allowed to search makespan-minimal plans in space of POCL plans!

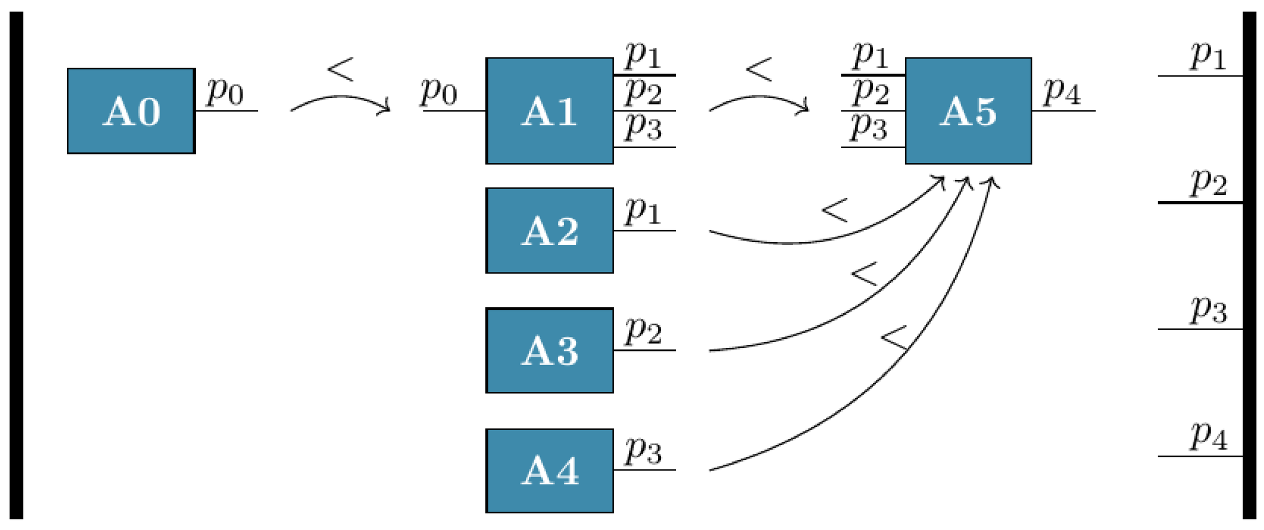
Makespan-minimal PO plan $\equiv$ Makespan-minimal POCL plan?

- Yes! We show that for every PO plan with makespan k there exists a POCL plan with makespan k.
- We also provide a polynomial-time procedure to *compute* such a POCL plan.

Optimizing Plans by *Deordering*

- *Deordering* is the process of (only!) deleting ordering constraints from a plan (i.e., without adding other ordering constraints) without violating executability.
- Using deordering, we can minimize the *number of ordering constraints* (thus increasing the number of linearizations) or we can minimize the makespan.

Example showing that minimizing the *number of ordering constraints* is not the same is minimizing the *makespan*:



- Delete 3 orderings from A2, A3, and A4 to A5 to minimize orderings.
- Delete the ordering from A1 to A5 to minimize makespan.

Computational Hardness of Deordering

- We are interested in the decision problem whether a deordering of an input plan exists that has a certain makespan.
- The computational hardness of *Reordering* was already known to be NP-complete. Reordering allows to change orderings (and links) arbitrarily.
- We show that *Deordering* is also *NP-complete.*
- Membership is trivial. For hardness we provide a reduction from 3-SAT:
  - We create a PO solution with makespan 4.
  - That plan can be deordered to a PO plan of makespan 3 if and only if the SAT formula is satisfiable.

Encoding:

We are given a 3-SAT formula in a form of a set of clauses $C = \{C_1, \ldots, C_m\}$ over the set of variables $X = \{x_1, \ldots, x_n\}$, such that for all $1 \le j \le m$, $C_j = \{l_{j,1}, l_{j,2}, l_{j,3}\}$ is a clause of three literals over X. We construct a plan PO plan $P = (PS, \prec)$ based on the STRIPS planning problem $\mathscr{P} = (V, A, s_I, g)$.

$$V = \bigcup_{1 \le i \le n} \{x_i^T, x_i^F, g_i\} \cup \bigcup_{1 \le j \le m} \{c_j\} \quad s_I = \emptyset \quad g = \{g_i \mid 1 \le i \le n\}$$

action	*prec*	*add*	*del*	action	*prec*	*add*	*del*
T_i	$\emptyset$	$\emptyset$	$\{g_i\}$	B_j^1	$\{l_{j,1}^*\}$	$\{c_j\}$	$\emptyset$
A_i^T	$\emptyset$	$\{g_i, x_i^T\}$	$\emptyset$	B_j^2	$\{l_{j,2}^*\}$	$\{c_j\}$	$\emptyset$
A_i^F	$\emptyset$	$\{g_i, x_i^F\}$	$\emptyset$	B_j^3	$\{l_{j,3}^*\}$	$\{c_j\}$	$\emptyset$
				D_j	$\{c_j\}$	$\emptyset$	$\emptyset$

Table 1: Actions for each atom x_i (left) and clause C_j (right). We define $l_{j,r}^* \leftarrow x_k^T$ if $l_{j,r} = x_k$ and $l_{j,r}^* \leftarrow x_k^F$ if $l_{j,r} = \neg x_k$.

$$T_i \prec \left\{ \begin{array}{c} A_i^T \\ A_i^F \end{array} \right\} \prec \left\{ \begin{array}{c} B_j^1 \\ B_j^2 \\ B_j^3 \end{array} \right\} \prec D_j.$$

ulm university universität
uulm

Deutsche
Forschungsgemeinschaft
DFG