Lecture *Hierarchical Planning*

Chapter: *Solving (Non-Hierarchical) Planning Problems via SAT*

Gregor Behnke

Institute of Artificial Intelligence,
Ulm University, Germany

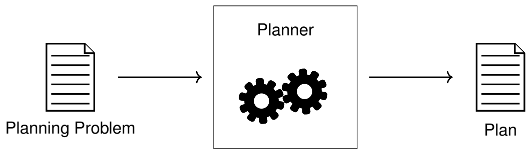
Winter Term 2018/2019

(Compiled on: November 30, 2023)

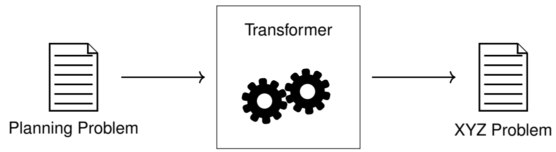
ulm university universität uulm

Overview:

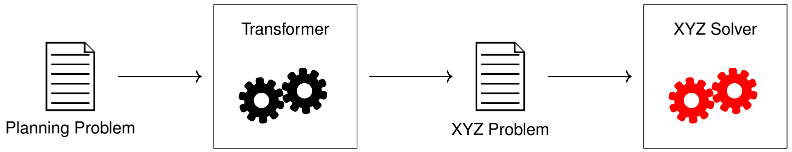
Idea: Problem Transformation



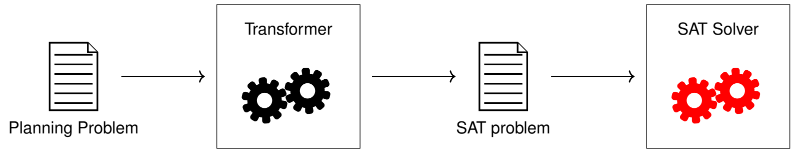
Idea: Problem Transformation



Idea: Problem Transformation



Idea: Problem Transformation



SAT

Definition (SAT)

Given a propositional formula $\mathcal{F}$, decide whether $\mathcal{F}$ has a satisfying valuation.

Definition (CNF-SAT)

Given a propositional formula $\mathcal{F}$ in conjunctive normal form, decide whether $\mathcal{F}$ has a satisfying valuation.

A valuation is an assignment of decision variables to $\{\top, \bot\}$.

CNF:

$$\mathcal{F} = \bigwedge_{C \in \mathfrak{C}} \bigvee_{\ell \in C} \ell$$

($\mathfrak{C}$ is the set of clauses; C is a clause, a set of literals.)

SAT Solvers

- SAT solvers are programs that determine whether a satisfying valuation exists and if so output it.
- A **lot** of research in recent years (annual competitions since 2002).
- Usable OSes have `minisat` in their package manager.
- Standardised input format DIMACS:

```
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

$\equiv$

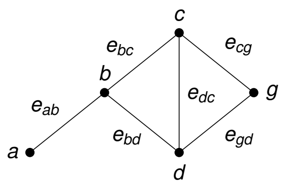
CNF with 5 vars and 3 clauses:

$(v_1 \vee \neg v_5 \vee v_4) \wedge$
$(\neg v_1 \vee v_5 \vee v_3 \vee v_4) \wedge$
$(\neg v_3 \vee \neg v_4)$

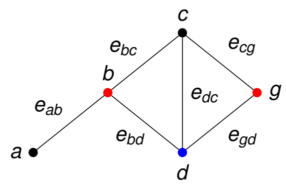
Colouring

Definition

Given a graph $G = (V, E)$ and a number k.
Is there an assignment of k colours to the vertices of G, such that all adjacent vertices have different colours?



SAT Modelling | Theoretical Background | Sequential Classical Planning in SAT | Invariants | ∀-step | ∃-step | Summary
Modelling Example
Colouring
Definition
Given a graph $G = (V, E)$ and a number k.
Is there an assignment of k colours to the vertices of G, such that all adjacent vertices have different colours?



Colouring

Variables for choosing the colour of each node

$$\texttt{colour}^i_v \text{ where } v \in V \text{ and } i \in \{1, \ldots, k\}$$

If a node has a colour, all adjacent nodes have a different colour

$$\texttt{colour}^i_v \rightarrow \neg\texttt{colour}^i_w \qquad \forall (v, w) \in E$$

$$\neg\texttt{colour}^i_v \vee \neg\texttt{colour}^i_w \qquad \forall (v, w) \in E$$

Every node has a colour

$$\bigvee_{i=1}^{k} \texttt{colour}^i_v \qquad \forall v \in V$$

Every node has at most one colour

$$\bigwedge_{i=1}^{k} \left[\texttt{colour}^i_v \rightarrow \bigwedge_{j=1, i \neq j}^{k} \neg\texttt{colour}^j_v \right] \qquad \forall v \in V$$

Computational Complexity

Definition (PLANEX)

Given a planning problem $\mathcal{P}$.
Is there a solution π of $\mathcal{P}$.

Theorem (Bylander'94)

PLANEX *is* $\mathbb{PSPACE}$*-complete.*

Theorem (Bylander'94)

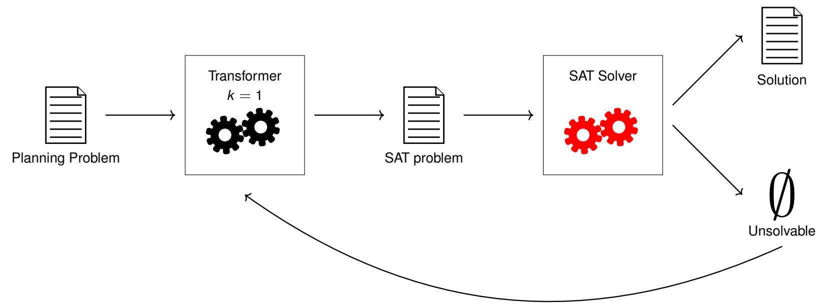
PLANEX *with bounded plan length* k *is* $\mathbb{PSPACE}$*-complete.*

$\mathbb{PSPACE}$ with $\mathbb{NP}$ calculus?

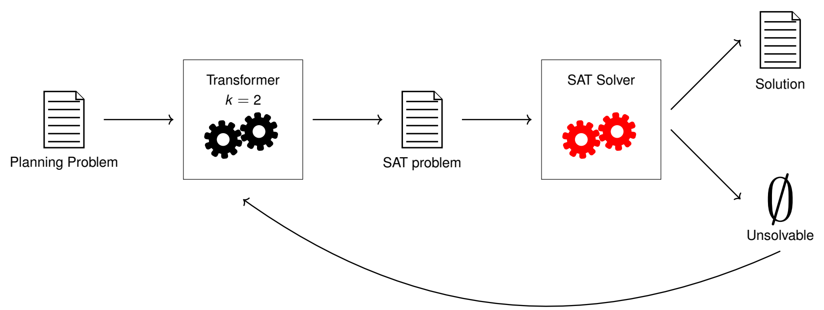
Transformation Idea

- Bounded plan length assumes binary encoding of k.
- What if we assume k in *unary* encoding?
- PLANEX "becomes" $\mathbb{NP}$-"complete".
- For full PLANEX: how to choose the plan length?
  - Theoretical limit: $2^{|V|}$.
  - Practical limit: usually smaller (sometimes polynomially bounded).
- Start with a small k and increase until a solution is found.

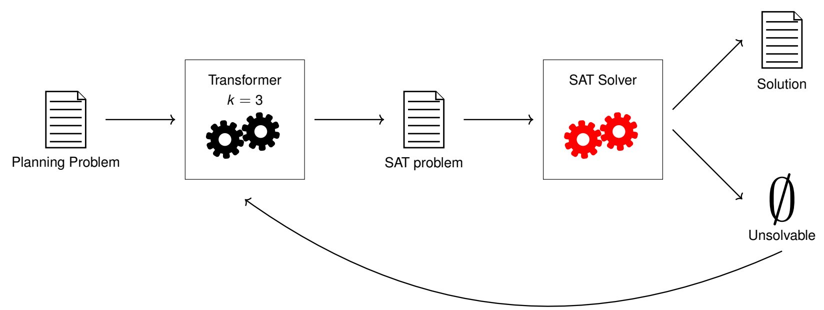
Bound Iteration



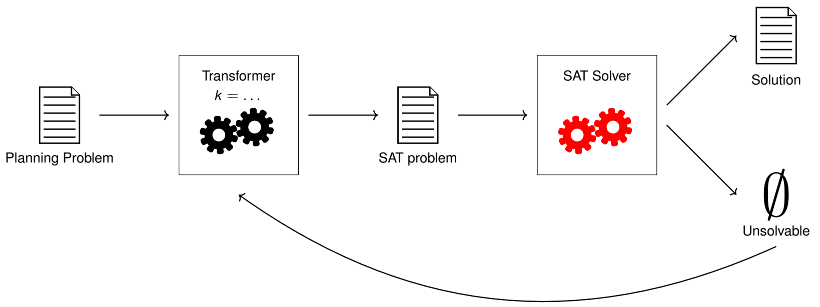
Bound Iteration



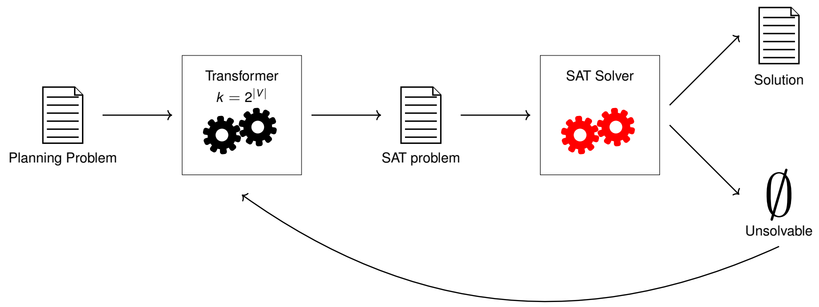
Bound Iteration



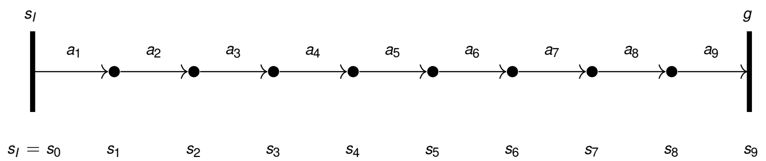
Bound Iteration



Bound Iteration



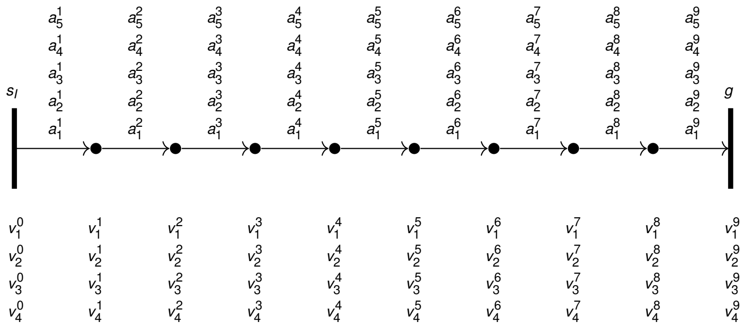
Classical Planning via SAT [Kautz&Selman'92]



A (classical) plan is just a sequence of state transitions.

- "Mechanics" is identical in all timesteps.
- Just model one timestep and copy'n'paste.
- Edge constraints!

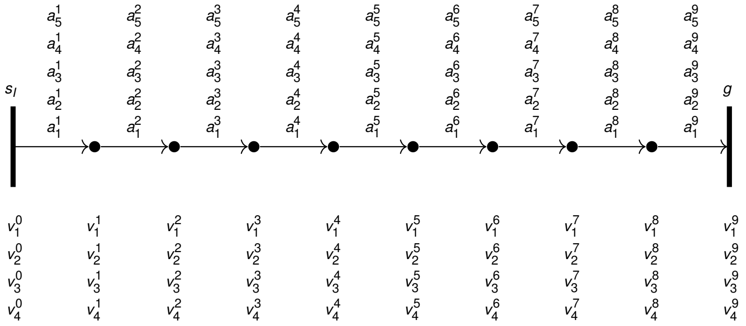
Decision Variables



We only need two types of decision variables!

1. a_i^t – Action i is executed at time t.
2. v_i^t – State variable i is true at time t.

Overall Formula

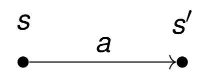


Constraints to check:

- Correctly applying actions at each time step (τ).
- s_I and g must be respected.

$$\mathcal{F} = \bigwedge_{t=0}^{k-1} \tau(t) \wedge \bigwedge_{v_i \in s_I} v_i^0 \wedge \bigwedge_{v_i \in V \setminus s_I} \neg v_i^0 \wedge \bigwedge_{v_i \in g} v_i^k \quad \text{here: } k = 9$$

Classical Planning via SAT



Constraints to check by $\tau(t)$:

- F_1 Preconditions must hold (in s).
- F_2 Effects must occur (in s').
- F_3 Unaffected state variables stay unchanged.
- F_4 At most one action per timestep.
- F_5 At least one action per timestep. Necessary? **No.**

Classical Planning via SAT

- Preconditions must hold:

$$F_1 = \bigwedge_{a \in A} \left[a^{t+1} \rightarrow \bigwedge_{v \in pre(a)} v^t \right]$$

- Effects must occur:

$$F_2 = \left[\bigwedge_{a \in A} \left(a^{t+1} \rightarrow \bigwedge_{v \in add(a)} v^{t+1} \right) \right] \wedge$$
$$\left[\bigwedge_{a \in A} \left(a^{t+1} \rightarrow \bigwedge_{v \in del(a)} \neg v^{t+1} \right) \right]$$

Classical Planning via SAT

- Variables not affected by the executed action must stay the same.

  → Frame Problem!

$$F_3 = \bigwedge_{v \in V} \left[(\neg v^t \wedge v^{t+1}) \rightarrow \bigvee_{a \in A \text{ with } v \in add(a)} a^{t+1} \right] \wedge$$
$$\bigwedge_{v \in V} \left[(v^t \wedge \neg v^{t+1}) \rightarrow \bigvee_{a \in A \text{ with } v \in del(a)} a^{t+1} \right]$$

- Only one action at a time:

$$F_4 = \text{at-most-one}(\{a^t \mid a \in A\})$$

At-most-one

Given a set of decision variables $X = \{x_1, \ldots, x_{|X|}\}$. Find a set of clauses that, if satisfied, will ensure that at most one $x \in X$ is true.

Naive encoding:

$$\bigwedge_{x_1 \in X} \bigwedge_{x_2 \in X \setminus \{x_1\}} \underbrace{\neg x_1 \vee \neg x_2}_{\substack{(x_1 \Rightarrow \neg x_2) \wedge \\ (x_2 \Rightarrow \neg x_1)}}$$

At-most-one

Idea: Introduce new variables!

f_i – from index i on all x_i will be false

i.e. it is **f**orbidden to use any x_i after i

Sequential encoding:

$$\bigwedge_{i=1}^{|X|-1} \underbrace{\neg x_i \vee f_i}_{x_i \Rightarrow f_i} \qquad \bigwedge_{i=2}^{|X|-1} \underbrace{\neg f_{i-1} \vee f_i}_{f_{i-1} \Rightarrow f_i}$$

$$\bigwedge_{i=1}^{|X|} \underbrace{\neg x_i \vee \neg f_{i-1}}_{\substack{(x_i \Rightarrow \neg f_{i-1}) \wedge \\ (f_{i-1} \Rightarrow \neg x_i)}}$$

At-most-one

Maybe this is a bit much ...

n_i – bit i (0-index) of a $\lceil \log(|X|) \rceil$-digit binary number if one

Binary encoding:

$$\neg x_i \vee n_j \qquad \text{if } \frac{i}{2^j} \mod 2 = 1$$

$$\neg x_i \vee \neg n_j \qquad \text{if } \frac{i}{2^j} \mod 2 = 0$$

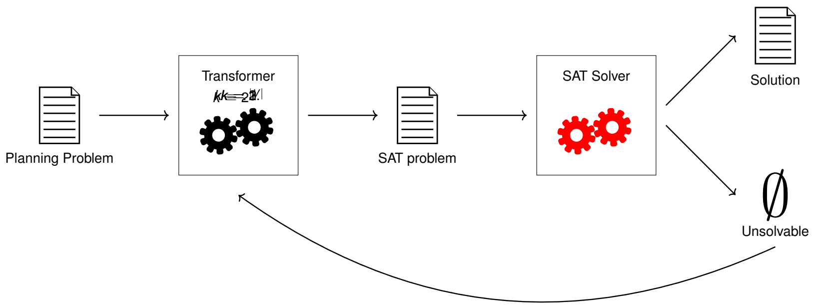
Different AMO Implementations[1]

encoding	#clauses	#new variables
binomial	n^2	0
binary	$n \log n$	$\log n$
sequential	$3n$	n
commander	$\frac{7}{2}n$	$\frac{n}{2}$
product	$2(n + n^{\frac{1}{m+1}})$	$2n^{\frac{1}{2}}$

where n is the number of atoms, i.e., $|X|$

[1] Frisch and Giannaros; SAT Encodings of the At-Most-k Constraint – Some Old, Some New, Some Fast, Some Slow; 2010

Bound Iteration



Classical Planning via SAT

There are **a lot** of improvements to this formula.

- Invariants.
- ∀-step semantics.
- ∃-step semantics.

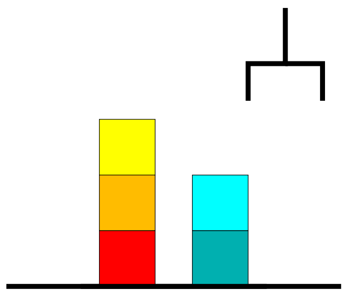
What are Invariants?

Is there **anything** we know about states in a planning problem?

Definition (Invariant)

An invariant $\mathcal{I}$ is a formula over the state variables such that for all states s reachable from s_I it holds $s \models \mathcal{I}$.

What are Invariants?



Predicates:

- $on(x, y)$ – x lies directly on y.
- $free(x)$ – x has no block above it.

Actions:

- $pickup(x)$ – pick up x, if it is free.
- $putdown(x, y)$ – put x on y, if y is free (*table* is always free).

Are the following formulae invariants?

1. $\forall b \in Block : (\exists b' \in Block : on(b', b)) \vee free(b)$ — **No.**
2. $\forall b \in Block : on(b, table)$ — **No.**
3. $\forall b, b' \in Block : \neg on(b', b) \vee \neg on(b, b')$ — **Yes.**

Invariants are Difficult

How hard is verifying an invariant?
As hard as planning.
Also there are too many invariants.

- Compute an approximation of all invariants of a fixed form.
- Restrict to binary-or invariants:

$$\ell_1 \vee \ell_2$$

Computing Invariants [Rintanen'98]

Note: Here we consider some action $a = (pre, add, del)$ and denote with $eff = add(a) \cup \{\neg v \mid v \in del(a)\}$ its effects (as a literal set).

$$\neg V = \{\neg v \mid v \in V\} \quad (\ell \in V \cup \neg V \text{ denotes a literal.})$$

$U_{\langle pre,eff\rangle}(\mathcal{I})$ gives all properties (positive or negative state variables) that hold after the execution of an action $a = \langle pre, eff\rangle$

$$U_{\langle pre,eff\rangle}(\mathcal{I}) = (\{\ell \in V \cup \neg V \mid \mathcal{I} \cup pre \models \ell\} \setminus \overbrace{\{\neg \ell \mid \ell \in eff\}}^{\equiv(\{\neg v \mid v \in add\} \cup del)}) \cup eff$$

$F_{\langle pre,eff\rangle}(\mathcal{I})$ is a *filter* for invariants, returning those that hold after the execution of an action $a = \langle pre, eff\rangle$

$$F_{\langle pre,eff\rangle}(\mathcal{I}) = \begin{cases} \mathcal{I} \quad \text{if } \mathcal{I} \cup pre \models \bot \text{ and otherwise:} \\ \{\ell_1 \vee \ell_2 \in \mathcal{I} \mid (\neg \ell_1 \notin eff \text{ or } \ell_2 \in U_{\langle pre,eff\rangle}(\mathcal{I})) \text{ and} \\ \qquad (\neg \ell_2 \notin eff \text{ or } \ell_1 \in U_{\langle pre,eff\rangle}(\mathcal{I}))\} \end{cases}$$

Computing Invariants [Rintanen'98], cont'd

Call $R_A(\mathcal{I}) := F_{a_1}(F_{a_2}(\cdots F_{a_n}(\mathcal{I})\cdots))$ with initial invariant $I_{init} = \{v \vee \ell \mid v \in s_I, \ell \in V \cup \neg V\} \cup \{\neg v \vee \ell \mid v \notin s_I, \ell \in V \cup \neg V\}$ and arbitrary linearization of action set A, $a_1, \ldots, a_n$, until $\mathcal{I}$ does not change anymore.

R stands for "*reduce* invariant set".

How to Use Invariants

What to do with an invariant $\ell_1 \vee \ell_2$?

Add it to every timestep t as $\ell_1^t \vee \ell_2^t$.

Linear Plans are Bad!

Consider the following (single) planning problem:

drive(A, B), load(B), drive(B, C), unload(C), drive(F, D), load(D), drive(D, E), unload(E)

drive(A, B)	load(B)	drive(B, C)	unload(C)
drive(F, D)	load(D)	drive(D, E)	unload(E)

∀-step [Kautz&Selman'96]

Allow parallel execution of actions.
But when?

- Let $\mathcal{A}$ be some set of actions.
- Parallel execution of $\mathcal{A}$ is safe, if all (∀) linearisations of $\mathcal{A}$ are executable. (Note the similarity to POCL planning.)
- Necessary conditions:
  - All actions are executable in the previous state as all could be the first.
  - No action can have a delete-effect that is a precondition of another action, i.e., $\forall a_1 \neq a_2 \in \mathcal{A} : del(a_1) \cap prec(a_2) = \emptyset$, as a_1 can occur before a_2.
- Sufficient conditions: Necessary conditions are already sufficient.

Encoding ∀-step

Remove the at-most-one constraints and add:

$$a_1^t \rightarrow \neg a_2^t \qquad \forall a_1, a_2 \in A \text{ with } del(a_1) \cap pre(a_2) \neq \emptyset$$

$\rightarrow$ quadratic effort.

Is this the best we can do? No!

Encoding Interference

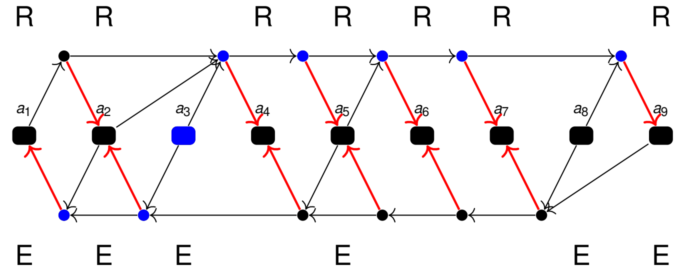
Idea 1: switch from a action-centric to a state variable-centric view.
For every $v \in V$: if $v \in add(a_1)$ and $v \in del(a_2)$ add $a_1^t \rightarrow \neg a_2^t$
Idea 2: if one action with $v \in del(a_2)$ is forbidden, so are all others.
Idea 3: express this with additional variables!
The only problem is that an operation must not disable itself.

Arrange the actions with $v \in pre(a) \cup del(a)$ as a sequence S.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
E	E	E		E			E	E
R	R		R	R	R	R		R

- E_v – subsequence of S with $v \in del(a)$ (**E**rasing)
- R_v – subsequence of S with $v \in pre(a)$ (**R**equiring)

Chains

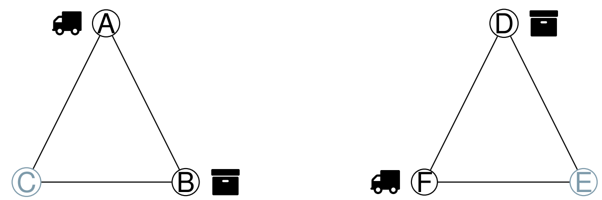


$$chain(S, E, R) = \bigwedge \{a^i \rightarrow \mathtt{f}^j \mid i < j, a^i \in E, a^j \in R, \{a_{i+1}, \ldots, a_{j-1}\} \cap R = \emptyset\} \cup$$
$$\{\mathtt{f}^i \rightarrow \mathtt{f}^j \mid i < j, \{a^i, a^j\} \in R, \{a_{i+1}, \ldots, a_{j-1}\} \cap R = \emptyset\} \cup$$
$$\{\mathtt{f}^i \rightarrow \neg a^i \mid v^i \in R\}$$

Two chains for every $v \in V$ with fresh decision variables $\mathtt{f}^i$.

Parallel Plans are (Still) Bad!

(Re-)Consider the following (single) planning problem:



drive(A, B)	load(B)	drive(B, C)	unload(C)
drive(F, D)	load(D)	drive(D, E)	unload(E)

drive(A, B)	load(B) drive(B, C)	unload(C)
drive(F, D)	load(D) drive(D, E)	unload(E)

What Kind of Parallelism do we Look for?

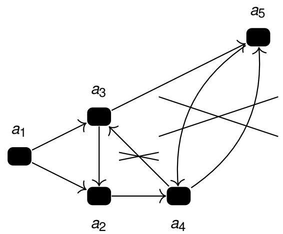
- Absolutely safe parallelism.
  - All linearisations will always be executable and lead to the same state.
  - ∀-step.
- (Sometimes) Safe parallelism.
  - At least one linearisation is executable and all executable linearisations lead to the same state.
  - ∃-step.

∃-step Parallelism

- Given a set of actions $\mathcal{A}$. We call them ∃-step executable if a linearisation exists that is executable and all executable linearisations lead to the same state.
- How difficult to determine? First part is $\mathbb{NP}$-complete.
- How to encode?
- Results in the Kautz&Selman encoding ...

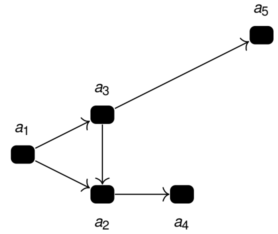
Disabling Graph [Rintanen,Heljanko,Niemelä'06]

- Approximate ∃-step semantics.
- Analyse dependency between actions.
- Similar to ∀-step:
  - If $del(a) \cap pre(a') \neq \emptyset$, execute a' before a.
  - Ignore if $\mathcal{I} \cup pre(a) \cup pre(a')$ is inconsistent.



∃-step [Rintanen,Heljanko,Niemelä'06]

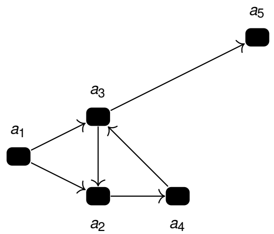
- Disabling Graph: $a \rightarrow b$ iff after executing a it may not be possible to execute b.
- We can safely execute actions in reverse topological order.



a_5, a_4, a_2, a_3, a_1

∃-step [Rintanen,Heljanko,Niemelä'06]

- Disabling Graph: $a \rightarrow b$ iff after executing a it may not be possible to execute b.
- We can safely execute actions in reverse topological order.
- DG may not be acyclic.
- Guess an order in every SCC and order SCCs in reverse topological order.
- If executed in parallel, we will always execute actions in **this** order.

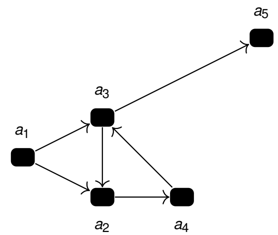


$(a_5), (a_2, a_3, a_4), (a_1)$

∃-step

What do we have to assert inside the propositional formula?

- Parallel actions must result in a consistent state. ✓
- Parallel actions must be executable.
  1. Actions must be applicable in the previous state.
  2. Reverse topological order of DG ensures that later actions are still applicable.

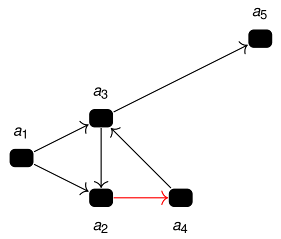


a_5, a_2, a_3, a_4, a_1

∃-step

What do we have to assert inside the propositional formula?

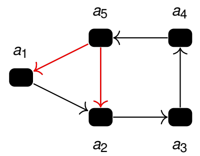
- Parallel actions must result in a consistent state. ✓
- Parallel actions must be executable.
  1. Actions must be applicable in the previous state.
  2. Reverse topological order of DG ensures that later actions are still applicable.
  3. In SCCs there might be edges opposite to the chosen order.
  4. SCC can be treated separately.
  5. If a_2 is executed, then a_4 must not.
  6. Enforced via *chaines*.



$(a_5), (a_2, a_3, a_4), (a_1)$

Chains

We are given an SCC and an ordering of its vertices.



$\pi = (a_5, a_4, a_3, a_2, a_1)$

- We want choose an acyclic subsequence of π.
- Do not choose both ends of a forward edge.
- Iterate over causes of these edges: $v \in del(a_1) \cap pre(a_2)$
  - E_v – subsequence of π with $v \in del(a)$ (**E**rasing)
  - R_v – subsequence of π with $v \in pre(a)$ (**R**equiring)
- Add $chain(\pi, E_v, R_v)$ – i.e. whenever an action erases v, we forbid any requiring action after it in π.

Further Improvements

Improvements for classical planning:

- Extension to conditional effects [Rintanen,Heljanko,Niemelä'06].
- Relaxed ∃-step [Wehrle&Rintanen'07].
- Parallel SAT search [Rintanen'04] [Rintanen,Heljanko,Niemelä'06].
- Specialised heuristics for SAT solvers [Rintanen'10a] [Rintanen'10b].
- Improved memory management [Rintanen'12].
- Incremental SAT-solving [Gocht&Balyo'17].

Extensions to non-classical planning:

- LTL [Mattmüller&Rintanen'07] [Behnke&Biundo'18].
- Partial Observability [Pandey&Rintanen'18].
- Temporal Planning [Rintanen'17].
- HTN Planning [Behnke,Höller,Biundo'17'18].

→ `https://users.aalto.fi/~rintanj1/satplan.html`

Solving Problems via Translation into SAT:

- Problem transformation is a general and important concept in computer science.
- SAT solvers are highly efficient and can be used to solve other difficult problems via transformation, even those in higher complexity classes with appropriate compilation.

Translating Classical planning into SAT:

- Classical planning problems can be translated into SAT.
- State-of-the-art improvements for this formula are based on:
  - State invariants.
  - Parallelism (∀-step, ∃-step).

Bylander'94	The Computational Complexity of Propositional STRIPS Planning.
Kautz&Selman'92	Planning as Satisfiability.
Kautz&Selman'96	Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search.
Rintanen'98	A Planning Algorithm not based on Directional Search.
Rintanen'04	Evaluation Strategies for Planning as Satisfiability.
Rintanen,Heljanko,Niemelä'06	Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search.
Wehrle&Rintanen'07	Planning as Satisfiability with Relaxed ∃-Step Plans.
Mattmüller&Rintanen'07	Planning for Temporally Extended Goals as Propositional Satisfiability.
Rintanen'10a	Heuristic Planning with SAT: Beyond Uninformed Depth-First Search.
Rintanen'10b	Heuristics for Planning with SAT.
Gocht&Balyo'17	Accelerating SAT Based Planning with Incremental SAT Solving.
Rintanen'17	Temporal Planning with Clock-Based SMT Encodings.
Behnke&Biundo'18	X and more Parallelism. Integrating LTL-Next into SAT-based Planning with Trajectory Constraints while Allowing for even more Parallelism.
Randey&Rintanen'18	Planning for Partial Observability by SAT and Graph Constraints.