Automated Planning and Scheduling

Lecture 7: SAT-based Planning
Tomáš Balyo, Dominik Schreiber | December 6, 2018
Finish basic SAT based planning
Advanced SAT based planning
Variable, Literal, Clause, Formula.

A (boolean) variable \( v \) has two possible values \( \text{true} \) or \( \text{false} \).

A literal \( l \) is either a variable \( v \) or its negated form \( \neg v \).

A clause \( c = l_1 \lor l_2 \lor \ldots \lor l_k \) is a set of literals which is \( \text{true} \) iff at least one of its literals is \( \text{true} \) (disjunction / OR of literals).

A (CNF) formula \( F = c_1 \land c_2 \land \ldots \land c_n \) is a set of clauses which is \( \text{true} \) iff all of its clauses are \( \text{true} \) (conjunction / AND of clauses).
SAT Preliminaries – reminder

Variable, Literal, Clause, Formula.

A (boolean) variable $v$ has two possible values true or false. A literal $l$ is either a variable $v$ or its negated form $\neg v$. A clause $c = l_1 \lor l_2 \lor \ldots \lor l_k$ is a set of literals which is true iff at least one of its literals is true (disjunction / OR of literals). A (CNF) formula $F = c_1 \land c_2 \land \ldots \land c_n$ is a set of clauses which is true iff all of its clauses are true (conjunction / AND of clauses).

Assignments.

An assignment $A$ maps each variable $v$ to true or false. If a formula $F$ is true under these variable values, $A$ is a satisfying assignment.
Variable, Literal, Clause, Formula.

A (boolean) variable $v$ has two possible values true or false.
A literal $l$ is either a variable $v$ or its negated form $\neg v$.
A clause $c = l_1 \lor l_2 \lor \ldots \lor l_k$ is a set of literals which is true iff at least one of its literals is true (disjunction / OR of literals).
A (CNF) formula $F = c_1 \land c_2 \land \ldots \land c_n$ is a set of clauses which is true iff all of its clauses are true (conjunction / AND of clauses).

Assignments.

An assignment $A$ maps each variable $v$ to true or false. If a formula $F$ is true under these variable values, $A$ is a satisfying assignment.

Example: $F = (x_1 \lor \neg x_2) \land (\neg x_3) \land (\neg x_1 \lor x_2)$,
Satisfying assignment $A = \{x_1 \mapsto \text{true}, x_2 \mapsto \text{true}, x_3 \mapsto \text{false}\}$
SAT Solving

SAT Problem.
Given a CNF formula $F$, find a satisfying assignment for $F$ or report that none exists (i.e. $F$ is unsatisfiable).

- Most prominent NP-complete problem
- Very efficient SAT Solvers exist
SAT Solving

SAT Problem.

Given a CNF formula $F$, find a satisfying assignment for $F$ or report that none exists (i.e. $F$ is unsatisfiable).

- Most prominent NP-complete problem
- Very efficient SAT Solvers exist

SAT Encoding.

Given a problem $\mathcal{P}$, a SAT encoding of $\mathcal{P}$ is a CNF formula $F_\mathcal{P}$ such that:

- $F_\mathcal{P}$ is satisfiable if and only if $\mathcal{P}$ has a solution.
- If $F_\mathcal{P}$ is satisfiable, then a solution to $\mathcal{P}$ can be (easily) extracted from a satisfying assignment $\mathcal{A}$ of $F_\mathcal{P}$.
Limit encoding of planning problem to at most $n$ steps (actions)

- When unsatisfiable, increase $n$ and try again
  - $\Rightarrow$ Top-level procedure similar to Graphplan, iter. deepening search
SAT Encoding of Planning

Encoded variables: “is” and “do”

- Variable $is^t_p$ for each atom $p$ and each step $t = 0, \ldots, n$
  - “Atom $p$ holds at step $t$”
- Variable $do^t_a$ for each action $a$ and each step $t = 0, \ldots, n - 1$
  - “Action $a$ is applied at step $t$”
- Found plan can be read directly from true action variables

<table>
<thead>
<tr>
<th>Step</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t@A$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t@B$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t@C$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$p1@A$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p1@C$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$p1@T$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p2@B$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p2@C$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$p2@T$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>loadp1@A</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>loadp2@B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dropp1@C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dropp2@C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>moveAtoB</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>moveBtoC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Clauses of the encoding:

1. The initial state must hold at $t = 0$.

$$\forall p \in s_0 : is^0_p \quad \forall p \notin s_0 : \neg is^0_p$$
SAT Encoding of Planning (2)

**Clauses** of the encoding:

1. The initial state must hold at $t = 0$.

   \[
   \forall p \in s_0 : is^0_p \quad \forall p \notin s_0 : \neg is^0_p
   \]

2. At every step, at least one action is applied.

   (Assume $A = \{a_1, \ldots, a_k\}$)

   \[
   \forall t \in \{0, \ldots, n-1\} : (do^t_{a_1} \lor do^t_{a_2} \lor \ldots do^t_{a_k})
   \]
SAT Encoding of Planning (2)

Clauses of the encoding:

1. The initial state must hold at \( t = 0 \).

\[
\forall p \in s_0 : is_p^0 \quad \forall p \notin s_0 : \neg is_p^0
\]

2. At every step, at least one action is applied.
   (Assume \( A = \{a_1, \ldots, a_k\} \))

\[
\forall t \in \{0, \ldots, n - 1\} : (do_{a_1}^t \lor do_{a_2}^t \lor \ldots do_{a_k}^t)
\]

3. At every step, at most one action is applied.

\[
\forall t \in \{0, \ldots, n - 1\}, \forall a_1 \neq a_2 : (\neg do_{a_1}^t \lor \neg do_{a_2}^t)
\]
4. If action \( a \) is applied at step \( t \), then \( \text{pre}(a) \) hold at step \( t \).

\[
\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in \text{pre}^+(a) : (\text{do}_a^t \rightarrow \text{is}_p^t) \\
\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in \text{pre}^-(a) : (\text{do}_a^t \rightarrow \neg \text{is}_p^t)
\]
If action $a$ is applied at step $t$, then $\text{pre}(a)$ hold at step $t$.

\[
\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in \text{pre}^+(a) : (\text{do}_a^t \rightarrow \text{is}_p^t) \\
\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in \text{pre}^-(a) : (\text{do}_a^t \rightarrow \neg\text{is}_p^t)
\]

If action $a$ is applied at step $t$, then $\text{eff}(a)$ hold at step $t + 1$.

\[
\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in \text{eff}^+(a) : (\text{do}_a^t \rightarrow \text{is}_p^{t+1}) \\
\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in \text{eff}^-(a) : (\text{do}_a^t \rightarrow \neg\text{is}_p^{t+1})
\]
SAT Encoding of Planning (3)

4. If action $a$ is applied at step $t$, then $\text{pre}(a)$ hold at step $t$.

$$\forall t \in \{0, \ldots, n-1\}, \forall a \in A, \forall p \in \text{pre}^+(a) : (do_a^t \rightarrow is_p^t)$$

$$\forall t \in \{0, \ldots, n-1\}, \forall a \in A, \forall p \in \text{pre}^-(a) : (do_a^t \rightarrow \neg is_p^t)$$

5. If action $a$ is applied at step $t$, then $\text{eff}(a)$ hold at step $t + 1$.

$$\forall t \in \{0, \ldots, n-1\}, \forall a \in A, \forall p \in \text{eff}^+(a) : (do_a^t \rightarrow is_p^{t+1})$$

$$\forall t \in \{0, \ldots, n-1\}, \forall a \in A, \forall p \in \text{eff}^-(a) : (do_a^t \rightarrow \neg is_p^{t+1})$$

6. The goal $g$ holds at step $n$.

$$\forall p \in g : \ is_p^n$$
Almost finished . . .

With clauses (1)–(6) for \( n = 1 \), the following is a solution for Trucking:

- Set \( d_0 \) \( \text{move}_{A\to B} \) to true, all other actions to false
- Set \( is^1_{p1@c} \) and \( is^1_{p2@c} \) to true

Why can the packages teleport to \( C \) without any action?
Almost finished . . .

With clauses (1)–(6) for $n = 1$, the following is a solution for Trucking:

- Set $do^0_{moveAtoB}$ to true, all other actions to false
- Set $is^1_{p1@c}$ and $is^1_{p2@c}$ to true

Why can the packages teleport to C without any action?

- Atoms can change arbitrarily without actions – explicitly disallow this
Almost finished . . .

With clauses (1)–(6) for \( n = 1 \), the following is a solution for Trucking:

- Set \( do^0_{\text{moveAtoB}} \) to true, all other actions to false
- Set \( is^1_{p1@c} \) and \( is^1_{p2@c} \) to true

Why can the packages teleport to \( C \) without any action?

- Atoms can change arbitrarily without actions – explicitly disallow this

If atom \( p \) changes between steps \( t \) and \( t + 1 \), an action which supports this change must be applied at \( t \):

\[
\forall t \in \{0, \ldots, n - 1\}, \forall p \in P : \left( (is^t_p \land \neg is^{t+1}_p) \rightarrow \bigvee_{a \in \text{support}(\neg p)} do^t_a \right)
\]

\[
\forall t \in \{0, \ldots, n - 1\}, \forall p \in P : \left( (\neg is^t_p \land is^{t+1}_p) \rightarrow \bigvee_{a \in \text{support}(p)} do^t_a \right)
\]
SAT Encoding: Properties

Clauses (1)–(7) together form a correct SAT encoding of planning

- Variable complexity?

More details on variable complexity:

- Clause complexity:
  - (3): $O(n \cdot |A|^2)$ clauses
  - (7): $O(n \cdot |P|)$ clauses
  - (4), (5): $O(n \cdot |A| \cdot |E|)$ clauses
  - $E := \max_{a \in A} \{|\text{pre}(a)| + |\text{eff}(a)|\}$

Can be significant e.g. if using quantified conditions

Total:

$O(n \cdot (|A|^2 + |P| + |A| \cdot |E|))$ clauses

Amount of clauses and variables linear in amount of steps

All clauses only contain variables of neighbored steps

Each clause "belongs" to some $t \Rightarrow$ Only contains variables from steps $t$ and $t + 1$
SAT Encoding: Properties

Clauses (1)–(7) together form a correct SAT encoding of planning

- Variable complexity?
  - \( n \cdot |P| \) “is” variables + \( n \cdot |A| \) “do” variables = \( n(|A| + |P|) \) variables

- Clause complexity?

\( E := \max_{a \in A} \{ |\text{pre}(a)| + |\text{eff}(a)| \} \)
Clauses (1)–(7) together form a correct SAT encoding of planning

- **Variable complexity?**
  \[ n \cdot |P| \text{ “is” variables} + n \cdot |A| \text{ “do” variables} = n(|A| + |P|) \text{ variables} \]

- **Clause complexity?**
  
  - (3): \( \mathcal{O}(n \cdot |A|^2) \) clauses, (7): \( \mathcal{O}(n \cdot |P|) \) clauses
  
  - (4),(5): \( \mathcal{O}(n \cdot |A| \cdot |E|) \) clauses
    
    - \( E := \max_{a \in A} \{|\text{pre}(a)| + |\text{eff}(a)|\} \)
    
    - Can be significant e.g. if using quantified conditions

- **Total:** \( \mathcal{O}(n \cdot (|A|^2 + |P| + |A| \cdot |E|)) \) clauses
SAT Encoding: Properties

Clauses (1)–(7) together form a correct SAT encoding of planning

- Variable complexity?
  - $n \cdot |P|$ “is” variables + $n \cdot |A|$ “do” variables = $n(|A| + |P|)$ variables

- Clause complexity?
  - (3): $O(n \cdot |A|^2)$ clauses, (7): $O(n \cdot |P|)$ clauses
  - (4),(5): $O(n \cdot |A| \cdot |E|)$ clauses
    - $E := \max_{a \in A} \{|\text{pre}(a)| + |\text{eff}(a)|\}$
    - Can be significant e.g. if using quantified conditions
  - Total: $O(n \cdot (|A|^2 + |P| + |A| \cdot |E|))$ clauses

- Amount of clauses and variables linear in amount of steps $n$

- All clauses only contain variables of neighbored steps
  - Each clause “belongs” to some $t$
    - Only contains variables from steps $t$ and $t + 1$
Structure of a Planning Formula

- $n = 0$: One set each of variables $V_0$, clauses $C_0$
- $n = 1$: New clauses $C_1$ as “logical glue” between $V_0$ and $V_1$
- $n = 2$: $C_2$ glue $V_1$ and $V_2$ together, etc.
  
  \[ \Rightarrow \text{Formula grows incrementally; cumulative set of clauses} \]

  \[ \Rightarrow \text{Only non-cumulative part: (6)} \]
  
  (goal holds at step $n$)

  - Can be enforced separately
Planning as Incremental SAT

- Avoid re-encoding entire problem for each $n = 0, 1, 2, \ldots$:
  - Maintain one single, growing formula
Avoid re-encoding entire problem for each $n = 0, 1, 2, \ldots$:
- Maintain one single, growing formula

Incremental SAT Solving: For each $n$,
- Add clauses (1)–(5), (7) permanently
- Assume clauses (6)
  - Literals are considered for one single solving attempt, then dropped
- Let SAT Solver search for a solution
  - Satisfiable? $\Rightarrow$ Finished
  - Unsatisfiable? $\Rightarrow$ Continue
Planning as Incremental SAT

Avoid re-encoding entire problem for each \( n = 0, 1, 2, \ldots \):
- Maintain one single, growing formula

Incremental SAT Solving: For each \( n \),
- Add clauses (1)–(5), (7) permanently
- Assume clauses (6)
  - Literals are considered for one single solving attempt, then dropped
- Let SAT Solver search for a solution
  - Satisfiable? \( \Rightarrow \) Finished
  - Unsatisfiable? \( \Rightarrow \) Continue

Solver can learn conflicts from unsatisfiable increments to speed up subsequent increments

More in “Practical SAT Solving” lecture

Implementation: edu.kit.aquaplanning.planners.SimpleSatPlanner
Improving the encoding

What is the weakness of the simple encoding?
Improving the encoding

What is the weakness of the simple encoding?

- Too many clauses: $O(n \cdot (|A|^2 + |P| + |A| \cdot |E|))$ clauses

Which term is the worst?
Improving the encoding

What is the weakness of the simple encoding?

- Too many clauses: $O(n \cdot (|A|^2 + |P| + |A| \cdot |E|))$ clauses

Which term is the worst?

- $|A|^2$ representing the at most one action

What to do?

- SAT Encoder Person: use better encoding for at-most-one constraint
  - doable with $|A| \log(|A|)$ clauses

More actions in each step $\Rightarrow$ less SAT solver calls $\Rightarrow$ PROFIT
Improving the encoding

What is the weakness of the simple encoding?
- Too many clauses: $O(n \cdot (|A|^2 + |P| + |A| \cdot |E|))$ clauses

Which term is the worst?
- $|A|^2$ representing the at most one action

What to do?
- SAT Encoder Person: use better encoding for at-most-one constraint
  - doable with $|A| \log(|A|)$ clauses
- Planner Person: Why only one action per step?
  - More actions in each step $\implies$ less SAT solver calls $\implies$ PROFIT
  - How many actions can be in one step?
Parallel Plans

Definition

A sequence of sets of actions \( P = [A_1, \ldots, A_k] \) is a parallel plan for a planning task \( \Pi \) if there is an action ordering function \( \trianglelefteq \) such that \( \trianglelefteq (A_1) \oplus \cdots \oplus \trianglelefteq (A_k) \) is a (sequential) plan for \( \Pi \), where \( \oplus \) denotes the concatenation of sequences. The sets \( A_i \) are called parallel steps and \( k \) is called the makespan of \( P \).

Example:

- \( A_1 = \{\text{loadP1, move-A-B}\} \)
- \( A_2 = \{\text{loadP2, move-B-C}\} \)
- \( A_3 = \{\text{unloadP1, unloadP2}\} \)
### Foreach step semantics

#### Interfering Actions

- A set of atoms $S$ is consistent if $S$ does not contain $x$ and $\neg x$ for some atom $x$.
- Two actions $a_1 = (p_1, e_1)$ and $a_2 = (p_2, e_2)$ **do not interfere** if:
  - $p_1 \cup p_2$ is consistent (consistent preconditions)
  - $e_1 \cup e_2$ is consistent (consistent effects)
  - $e_1 \cup p_2$ and $e_2 \cup p_1$ are consistent

#### Proposition

Let $A$ be a set of actions such that $\forall a_i \neq a_j \in A$ the actions $a_i$ and $a_j$ do not interfere. If there is an ordering $\sqsubseteq$ such that $\sqsubseteq(A)$ transforms the state $s_1$ to $s_2$ then all the possible orderings of $A$ transform $s_1$ to $s_2$.

- It is enough to suppress compatible interfering action pairs.
- For each ordering we get a valid plan, hence the name.
Foreach step semantics

A parallel plan \( P = [A_1, \ldots, A_k] \) satisfies the \( \forall \)-Step semantics if

- each action in \( A_j \) is applicable in the state \( s_j \),
- the effects of all the actions in \( A_j \) are applied in \( s_{j+1} \),
- each pair of actions in \( A_j \) do not interfere.
- \( [\triangledown(A_1) \oplus \cdots \oplus \triangledown(A_k)] \) is a valid plan for some ordering function \( \triangledown \).

Example:

- \( A_1 = \{\text{loadP1}\} \)
- \( A_2 = \{\text{move-A-B}\} \)
- \( A_3 = \{\text{loadP2}\} \)
- \( A_4 = \{\text{move-B-C}\} \)
- \( A_5 = \{\text{unloadP1, unloadP2}\} \)
 Exists step semantics

A parallel plan $P = [A_1, \ldots, A_k]$ satisfies the $\forall$-Step semantics if

- each action in $A_j$ is applicable in the state $s_j$,
- the effects of all the actions in $A_j$ are applied in $s_{j+1}$,
- each pair of actions in $A_j$ do not interfere,
- $[\sqsubseteq(A_1) \oplus \cdots \oplus \sqsubseteq(A_k)]$ is a valid plan for some ordering function $\sqsubseteq$.

Example:

- $A_1 = \{\text{loadP1, move-A-B}\}$
- $A_2 = \{\text{loadP2, move-B-C}\}$
- $A_3 = \{\text{unloadP1, unloadP2}\}$

Only 3 steps required
Relaxed Exists step semantics

A parallel plan $P = [A_1, \ldots, A_k]$ satisfies the $\forall$-Step semantics if

- each action in $A_j$ is applicable in the state $s_j$,
- the effects of all the actions in $A_j$ are applied in $s_{j+1}$,
- each pair of actions in $A_j$ do not interfere,
- $[\sqsubseteq(A_1) \oplus \cdots \oplus \sqsubseteq(A_k)]$ is a valid plan for some ordering function $\sqsubseteq$.

Example:

- $A_1 = \{\text{loadP1, move-A-B, loadP2}\}$
- $A_2 = \{\text{move-B-C, unloadP1, unloadP2}\}$

Only 2 steps required!

- What can we drop?
Relaxed Relaxed Exists step semantics

A parallel plan $P = [A_1, \ldots, A_k]$ satisfies the $\forall$-Step semantics if

- each action in $A_j$ is applicable in the state $s_j$,
- the effects of all the actions in $A_j$ are applied in $s_{j+1}$,
- each pair of actions in $A_j$ do not interfere.
- $[\sqsubseteq(A_1) \oplus \cdots \oplus \sqsubseteq(A_k)]$ is a valid plan for some ordering function $\sqsubseteq$.

Example:

$A_1 = \{\text{loadP1, move-A-B, loadP2, move-B-C, unloadP1, unloadP2}\}$

Only 1 steps required! We probably cannot do better than that :)}
How to encode the (relaxed) Exists step semantics into SAT?

Overview of Basic Ideas

The SAT encoding only approximates the semantics, i.e., the satisfiability of the constructed formula $F_k$ implies the existence of a $k$-step plan (not vice versa). The actions are ranked arbitrarily. The goal is to guess the order of actions in the final plan. A heuristic: use cycle–ignoring topological sorting on the enabling graph. Better ranking heuristics wanted (Master thesis anyone?)

The encoding allows only lower ranking actions before higher ranking ones in a step. The encoding uses implication chains.
How to encode the (relaxed) Exists step semantics into SAT?

Overview of Basic Ideas

- The SAT encoding only approximates the semantics, i.e., the satisfiability of the constructed formula $F_k$ implies the existence of a $k$-step plan (not vice versa)
- The actions are ranked arbitrarily
  - goal is to guess the order of actions in the final plan
  - heuristic: use cycle–ignoring topological sorting on the enabling graph
  - better ranking heuristics wanted (Master thesis anyone?)
- The encoding allows only lower ranking actions before higher ranking ones in a step.
- The encoding uses implication chains.
Enabling Graph

Definition

The *enabling graph* $G$ for a set of actions $A$ is a directed graph where vertices represent actions and there is an edge $(a, a')$ if $a$ supports $a'$, i.e., $G = (A, \{ a \rightarrow a' \mid a, a' \in A; \text{eff}(a) \cap \text{pre}(a') \neq \emptyset \})$. 
Ranking Actions

The actions are ranked using cycle–ignoring topological sorting on the enabling graph.

\[
\text{topologicalRanking}(O)
\]

\[
\begin{align*}
T_1 & \quad \text{global } lastRank := 0 \\
T_2 & \quad \text{global } visited := \{False, \ldots, False\} \\
T_3 & \quad \textbf{foreach } a \in O \ \textbf{do} \\
T_4 & \quad \text{rankAction}(a)
\end{align*}
\]

\[
\text{rankAction}(a)
\]

\[
\begin{align*}
R_1 & \quad \textbf{if } visited[a] = False \ \textbf{then} \\
R_2 & \quad visited[a] := True \\
R_3 & \quad \textbf{foreach } s \in \text{supportingActions}(a) \ \textbf{do} \\
R_4 & \quad \text{rankAction}(s) \\
R_5 & \quad r(a) := lastRank \\
R_6 & \quad lastRank := lastRank + 1
\end{align*}
\]
Encoding – these clauses same as before

1. The initial state must hold at $t = 0$.
   \[ \forall p \in s_0 : is^0_p \quad \forall p \notin s_0 : \neg is^0_p \]

2. The goal state must hold at $t = n$.
   \[ \forall p \in G : is^n_p \]

3. If atom $p$ changes between steps $t$ and $t + 1$, an action which supports this change must be applied at $t$:
   \[ \forall t \in \{0, \ldots, n - 1\}, \forall p \in P : \]
   \[
   \left( (is^t_p \land \neg is^{t+1}_p) \rightarrow \bigvee_{a \in \text{support}(\neg p)} do^t_a \right)
   \]
   \[
   \left( (\neg is^t_p \land is^{t+1}_p) \rightarrow \bigvee_{a \in \text{support}(p)} do^t_a \right)
   \]
Encoding Clauses – Preconditions

These clauses are added $\forall t \in \{0, \ldots, n - 1\}, \forall a \in A$

For Foreach and Exists Step:

4 If action $a$ is applied at step $t$, then $pre(a)$ hold at step $t$:

$$\forall p \in pre^+(a) : (do^t_a \rightarrow is^t_p)$$

$$\forall p \in pre^-(a) : (do^t_a \rightarrow \neg is^t_p)$$

For Relaxed and Relaxed Relaxed Exists Step:

4 If action $a$ is applied at step $t$, then $pre(a)$ hold at step $t$ or some supporting action happens before:

$$\forall p \in pre^+(a) : (do^t_a \rightarrow (is^t_p \vee do^t_{a^1_p} \vee \ldots \vee do^t_{a^k_p}))$$

$$\forall p \in pre^-(a) : (do^t_a \rightarrow (\neg is^t_p \vee do^t_{a^1_{\neg p}} \vee \ldots \vee do^t_{a^k_{\neg p}}))$$

where $a^*_p$ are supports for $p$ with lower rank than $a$. 
Encoding Clauses – Effects

For Foreach, Exists and Relaxed Exists Step:

5 If action $a$ is applied at step $t$, then $\text{eff}(a)$ hold at step $t + 1$ or some other action sets in later:

\[
\forall p \in \text{eff}^+(a) : (do_a^t \rightarrow is_p^{t+1})
\]
\[
\forall p \in \text{eff}^-(a) : (do_a^t \rightarrow \neg is_p^{t+1})
\]

For Relaxed Relaxed Exists Step:

5 If action $a$ is applied at step $t$, then $\text{eff}(a)$ hold at step $t + 1$ or some other action sets in later:

\[
\forall p \in \text{eff}^+(a) : (do_a^t \rightarrow (is_p^{t+1} \lor do_{a_p}^t \lor \ldots \lor do_{a_k}^t))
\]
\[
\forall p \in \text{eff}^-(a) : (do_a^t \rightarrow (\neg is_p^{t+1} \lor do_{a_{-p}}^t \lor \ldots \lor do_{a_{-k}}^t))
\]

where $a_p^*$ are supports for $p$ and $\neg p$ with higher rank than $a$. 
Dealing with action interference

Easy in Foreach step

- add $\neg do_{a_i} \lor \neg do_{a_j}$ for each pair of interfering actions $a_i, a_j$

How about (Relaxed)* Exists step?
What could go wrong?
Dealing with action interference

Easy in Foreach step

- add \((\neg do_{a_i} \lor \neg do_{a_j})\) for each pair of interfering actions \(a_i, a_j\)

How about (Relaxed)* Exists step?

What could go wrong?

- **In Exists Step:** one actions destroys the precondition for another action coming later

- **In (Relaxed)+ Exists Step:**
  - one actions destroys the precondition for another action
  - one actions destroys the effect of another action
  - one actions \(a_1\) sets up the precondition for \(a_2\) but then some action \(a_3\) between them destroys it again
  - one actions \(a_1\) set up the precondition for \(a_2\) but then some action \(a_3\) between them destroys it again, but then \(a_4\) between \(a_3\) and \(a_2\) may restore it back ...
We have a chain for each positive and negative atom

- $h_i$ are helper variables, $a_i = do_{a_i}$, full arrow is implication ($h_i \implies a_j$) dashed arrow is negative implication ($h_i \implies \neg a_j$)
- an action $a$ activates the chain for $\neg p$ if $p \in eff(a)$
- the chain for $\neg p$ deactivates an action $a$ if $p \in pre(a)$
- if one action activates the chain all actions with higher rank that require that atom in their preconditions are disabled
- $a_1$, $a_2$, $a_3$ and $a_5$ are opponents of the given atom $p$ (they have $\neg p$ as an effect).
- $a_3$ and $a_6$ require $p$ (they have $p$ in their preconditions).
- $a_3$ both requires and opposes $p$
  - Example: $p = truckAt_A$, $a_3 = move(A, B)$
- we need one helper variable $h$ in $p$’s chain for each action that requires $p$
- $a_1, a_2, a_3$ and $a_5$ are opponents of the given atom $p$ (they have $\neg p$ as an effect).
- $a_3, a_4$ and $a_6$ require $p$ (they have $p$ in their preconditions).
- $a_3$ both requires and opposes $p$
- $a_4$ supports $p$ (has $p$ as an effect)
- $a_4$ can break the chain between $h_3$ and $h_4$: $h_3 \implies (h_4 \lor a_4)$
Implication Chain Clauses

- each involved action gets a helper variable (except the last one)

For each atom $p$ at each time step $t$

1. $(h_{j-1} \rightarrow h_j)$ for each $j$ such that $a_j$ is not a support for $p$
2. $(h_{j-1} \rightarrow (h_j \lor \text{do}_{a_j}))$ for each $j$ such that $a_j$ is a support for $p$
3. $(\text{do}_{a_j} \rightarrow h_j)$ for each $j$ such that $a_j$ is an opponent for $p$
4. $(h_{j-1} \rightarrow \neg \text{do}_{a_j})$ for each $j$ such that $a_j$ requires for $p$
Extracting the Plan

How to get the plan from the satisfying assignment of the formula?
Extracting the Plan

How to get the plan from the satisfying assignment of the formula?

- Get the parallel plan (sequence of sets of actions)
  - check which $do^t_{ai}$ variables are true in each step
- Turn it into a regular plan
  - order the actions in each step according to the ranking
Exists (R)* Step Encodings properties

- The better you rank the actions (predict their ordering)
  - the least SAT solving steps you need
  - the faster and with less memory you find a plan
- For some domains 1 step is enough
  - if each actions is required only once
  - if you can guess the order of actions
- Number of variables/clauses?
  - Homework :)
- Future Work:
  - better ranking heuristics!
How about chaining “n++” part of the algorithm?

Why?
Runtime Profile

Evaluation times: logistics39-0

- Time in secs
- Time points

<table>
<thead>
<tr>
<th>Time Points</th>
<th>Evaluation Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>0</td>
</tr>
<tr>
<td>2-4</td>
<td>0</td>
</tr>
<tr>
<td>4-6</td>
<td>0</td>
</tr>
<tr>
<td>6-8</td>
<td>700</td>
</tr>
<tr>
<td>8-10</td>
<td>0</td>
</tr>
<tr>
<td>10-12</td>
<td>0</td>
</tr>
<tr>
<td>12-14</td>
<td>0</td>
</tr>
<tr>
<td>14-16</td>
<td>0</td>
</tr>
<tr>
<td>16-18</td>
<td>0</td>
</tr>
<tr>
<td>18-20</td>
<td>0</td>
</tr>
</tbody>
</table>

Tomáš Balyo, Dominik Schreiber – Planning and Scheduling

December 6, 2018

34/40
Runtime Profile

Evaluation times: gripper10

The diagram shows a histogram of evaluation times for gripper10, with time points on the x-axis and time in seconds on the y-axis. The data is represented by vertical bars, with the height indicating the frequency of evaluation times at each time point.
Runtime Profile

Evaluation times: blocks22
Scheduling Strategies by Rintanen

- Classical scheduling: solve 1, 2, 3, ... sequentially

- Algorithm A
  - start $n$ solvers in parallel solving 1, 2, ..., $n$
  - if a formula found unsat continue with the smallest not solved yet
  - can get past hard UNSAT formulas if $n$ is big enough
  - in the worst case $n$ times slower than sequential
  - bigger formulas – higher memory requirement
  - skipping lengths is ok, i.e., solve 10, 20, 30, ..., 10$n$

- Algorithm B – geometric
  - start $n$ solvers in parallel solving 1, 2, ..., $n$
  - solver solving step $k + 1$ has a time limit $g$ times less than the time limit of the solver solving $k$, for some constant $g < 1$
  - focus on solving the smaller steps
  - can choose higher $n$ than before
Runtime Profile

Finding a plan for blocks22 with Algorithm B

![Graph showing runtime profile for blocks22 with Algorithm B]
- Algorithm C – exponential
  - start $n$ solvers in parallel solving $1, 2, 4, 8, \ldots$
  - works surprisingly well
  - easy to run out of memory
  - finds very long plans
  - solves dozens of problems unsolved by the previous strategies
Stay tuned!

Next lecture: Hierarchical Task Network (HTN) Planning