Automated Planning and Scheduling

Lecture 6: Graphplan and SAT-based Planning
Tomáš Balyo, Dominik Schreiber | November 30, 2018
Outline

- Planning graphs and their properties
- The Graphplan procedure [BF97]
- Fundamentals of SAT-based planning
Planning graphs: Introduction

Remember relaxed planning graphs? Here comes their origin story . . .

- Enumerate reachable atoms and actions to understand problem
Remember relaxed planning graphs? Here comes their origin story . . .

- Enumerate reachable atoms and actions to understand problem
- Infer plan from “finished” planning graph
  - FF heuristic: extracts a delete-relaxed plan, guiding actual planning
Remember relaxed planning graphs? Here comes their origin story . . .

- Enumerate reachable atoms and actions to understand problem
- Infer plan from “finished” planning graph
  - FF heuristic: extracts a delete-relaxed plan, guiding actual planning
  - Graphplan: extracts a sequence of sets of actions $\langle A_1, \ldots, A_k \rangle$
    - Can be transformed easily into an actual, valid plan
    - Overall different planning approach; not a relaxation of the problem
Layers $i$ of possible atoms $P_i$ and potential actions $A_i$

One layer of atoms+actions $\cong$ one time step

- Multiple actions per step allowed when they do not conflict: any ordering must be valid and lead to identical results

- Negative atoms are included as a complementary atom set
For each atom $p$ at each layer, add persistence action $nop_p$

- $pre(nop_p) = eff(nop_p) = \{p\}$ (nop = “no operation”)
- Make explicit that an atom remains unchanged between layers
- Also for negative atoms
In addition to atoms $P_i$ and actions $A_i$, maintain sets of conflicts $M_i$:

- Identify pairs of atoms / of actions which logically cannot co-occur
- Remember these as mutually exclusive (mutex)
- Limits possible degree of action parallelism per step
Opposite atom mutex:

- Atom pairs \( \{p, \bar{p}\} \) are obviously mutex
- Notation for mutex: \( \{p, q\} \in M_i \) if \( p \) and \( q \) are mutex at layer \( i \)
- Example: \( \{t@A, \neg t@A\} \in M_1 \) (even: \( \{t@A, \neg t@A\} \in M_i \) for all \( i \)
- **Conflicting effects:**

  - Actions $a_1, a_2$ are mutex if an effect of $a_1$ is mutex with an effect of $a_2$
  - Example: $\{\text{driveAtoB}, \text{driveBtoC}\} \in M_2$ because $\{\text{t@B}, \neg\text{t@B}\} \in M_2$
Interference between actions:

- Actions \( \{a_1, a_2\} \) are mutex if an effect of \( a_1 \) interferes with a precondition of \( a_2 \):  \( \exists p \in \text{eff}(a_1) : \bar{p} \in \text{pre}(a_2) \)
- Example: \( \text{driveAtoB} \) deletes \( \text{t@A} \) which is needed by \( \text{loadp1@A} \)  
  \( \Rightarrow \{\text{driveAtoB, loadp1@A}\} \in M_1 \)
Conflicting enabling actions:

- Atoms \( \{p_1, p_2\} \) are mutex if each pair of enabling actions is mutex.
- Example: \( \{t@A, t@B\} \in M_1 \) because \( \{\text{driveAtoB}, \text{nop}_{t@A}\} \in M_1 \).
Conflicting enabling actions:

- Atoms \( \{p_1, p_2\} \) are mutex if each pair of enabling actions is mutex
- Example: \( \{t@A, t@B\} \in M_1 \) because \( \{\text{driveAtoB, nop}_{t@A}\} \in M_1 \)
- Similarly, \( \{t@B, p1@T\} \in M_1 \) because \( \{\text{driveAtoB, loadp1@A}\} \in M_1 \)
Decide if goal can be met at some layer

- At $P_3$, $p1@C$ and $p2@C$ are both reachable
- Still, $\{p1@C, p2@C\} \in M_3$ (see illustration for “‘proof’’’)
- As a consequence, goal is not satisfiable at $P_3$
  $\Rightarrow$ Expand graph until goals are not mutex any more (?)
1. The set of atoms in $P_i$ grows monotonically in $i$
   - Each atom $p$ at $P_{i-1}$ will also be at $P_i$ (due to $nop_p$)

2. The set of actions in $A_i$ grows monotonically in $i$
   - If some action $a$ is reachable at layer $i-1$, then it is also reachable at layer $i$ (monotonicity of atoms)

3. The sets of atoms and actions $(P_i, A_i)$ will eventually reach a fixpoint at some layer $k$, i.e. $P_k = P_{k+1} = \ldots$ and $A_k = A_{k+1} = \ldots$

   Maximum size of atoms and actions is bounded by problem size

   The sets always either increase or remain constant (1), (2)
   - If both remain constant, a fixpoint is reached
1. The **set of atoms** in $P_i$ grows monotonically in $i$
   - Each atom $p$ at $P_{i-1}$ will also be at $P_i$ (due to $\text{nop}_p$)

2. The **set of actions** in $A_i$ grows monotonically in $i$
   - If some $a$ is reachable at layer $i-1$,
     then it is also reachable at layer $i$ (monotonicity of atoms)
The set of atoms in $P_i$ grows monotonically in $i$
- Each atom $p$ at $P_{i-1}$ will also be at $P_i$ (due to $\text{nop}_p$)

The set of actions in $A_i$ grows monotonically in $i$
- If some $a$ is reachable at layer $i-1$,
  then it is also reachable at layer $i$ (monotonicity of atoms)

The sets of atoms and actions ($P_i, A_i$) will eventually reach a fixpoint at some layer $k$, i.e. $P_k = P_{k+1} = \ldots$ and $A_k = A_{k+1} = \ldots$
- Maximum size of atoms and actions is bounded by problem size
- The sets always either increase or remain constant (1), (2)
- If both remain constant, a fixpoint is reached
The set of mutexes $M_i$ eventually decreases monotonically in $i$:

**Theorem.**

Let $k \in \mathbb{N}$ such that $\hat{P} := P_k = P_{k+1} = \ldots$ and $\hat{A} := A_k = A_{k+1} = \ldots$, i.e. the sets of atoms and actions have reached a fixpoint at layer $k$. Then the size of $M_{k+j}$ will decrease monotonically in $j \geq 0$. 
The set of mutexes $M_i$ eventually decreases monotonically in $i$:

**Theorem.**

Let $k \in \mathbb{N}$ such that $\hat{P} := P_k = P_{k+1} = \ldots$ and $\hat{A} := A_k = A_{k+1} = \ldots$, i.e. the sets of atoms and actions have reached a fixpoint at layer $k$. Then the size of $M_{k+j}$ will decrease monotonically in $j \geq 0$.

Needed key observations:

- **Non-mutex pairs** at layer $i$ will never become mutex
  - Non-mutex pairs do not have any intrinsic logical conflicts
  - No new conflicts will arise because $A_i$ and $P_i$ remain unchanged

- **Mutex pairs** at layer $i$ may later become not mutex
  - Intuition: More layers will allow some mutex actions to be executed one after another, opening up new possibilities
Properties (1)–(4) imply the following:

**Theorem.**

A planning graph eventually reaches a fixpoint where no atoms, actions, or mutexes will change in any subsequent layers.
Planning graph: Termination

- Properties (1)–(4) imply the following:

**Theorem.**
A planning graph eventually reaches a fixpoint where no atoms, actions, or mutexes will change in any subsequent layers.

- Consequence: Building a planning graph always terminates (and: the planning graph is finite)
Planning graph: Termination

- Properties (1)–(4) imply the following:

**Theorem.**

A planning graph eventually reaches a fixpoint where no atoms, actions, or mutexes will change in any subsequent layers.

- Consequence: Building a planning graph always terminates (and: the planning graph is finite)
- Complexity: polynomial in amount of atoms $P$ and actions $A$
  - Construction of $P_i$, $A_i$, $M_i$ polynomial for each $i$ (easy construction rules, quadratic amount of checks for mutexes)
  - Due to monotonicity, at most $|P| + |A|$ layers until $\hat{P}$ and $\hat{A}$ reached (at each layer, at least one atom or action joins)
  - Then, at most $|P|^2 + |A|^2$ more layers until $\hat{M}$ reached (at each layer, at least one mutex joins)
Making use of a planning graph

Properties of $\hat{P}, \hat{A}, \hat{M}$ give "one-sided hints" to solvability:

- If $\exists p \in g : p \notin \hat{P}$, goal is unreachable
- If for some $\{p, q\} \subseteq g$, $\{p, q\} \in \hat{M}$, goal is unreachable as well
- Otherwise: There might be a plan
Making use of a planning graph

Properties of $\hat{P}$, $\hat{A}$, $\hat{M}$ give “one-sided hints” to solvability:
- If $\exists p \in g : p \notin \hat{P}$, goal is unreachable
- If for some $\{p, q\} \subseteq g$, $\{p, q\} \in \hat{M}$, goal is unreachable as well
- Otherwise: There might be a plan

How to use planning graph to find an actual plan?
- If a plan may exist for layer $l$, search for it
- Use collected problem properties from planning graph
AlGORITHM 1 Abstract Graphplan

1: \( G := \langle A_0, M_0, P_0 \rangle = \langle \{\}, \{\}, s_0 \rangle \)
2: \( l := 0 \)
3: while TRUE do
4:    if \( g \in P_l \) and \( \forall g_1, g_2 \in g : \{ g_1, g_2 \} \notin M_l \) then
5:        result := extractPlan(\( G \))
6:        if result \( \neq \) FAILURE then return result
7:    end if
8:    \( l := l + 1 \)
9:    \((A_l, M_l, P_l) := \) expand(\( G \))
10:   \( G := G \cup \langle A_l, M_l, P_l \rangle \)
11:   if \( G \) completely converged then return FAILURE
12: end while
The Graphplan procedure (2)

How does $\text{expand}(G)$ work?

- Calculate all applicable actions in the last $P_i$
- Calculate the resulting next set of atoms
- Calculate all mutexes
The Graphplan procedure (2)

How does \textit{expand}(G) work?
- Calculate all applicable actions in the last $P_i$
- Calculate the resulting next set of atoms
- Calculate all mutexes

How does \textit{extractPlan}(G) work?
- \textbf{Backwards search} algorithm to satisfy goal (see last lecture)
- Atom required $\Rightarrow$ Pick \textit{any} applicable enabling action (OR)
- Action required $\Rightarrow$ Add all preconditions to required atoms (AND)
- Branching on OR choices, backtracking etc.

What does completely converged mean?
Atoms, actions, mutexes reached fixpoint
Last search yielded no new information on conflicting goal atom sets
The Graphplan procedure (2)

How does $\text{expand}(G)$ work?
- Calculate all applicable actions in the last $P_i$
- Calculate the resulting next set of atoms
- Calculate all mutexes

How does $\text{extractPlan}(G)$ work?
- **Backwards search** algorithm to satisfy goal (see last lecture)
- Atom required $\Rightarrow$ Pick *any* applicable enabling action (**OR**)
- Action required $\Rightarrow$ Add all preconditions to required atoms (**AND**)
- Branching on **OR** choices, backtracking etc.

What does completely converged mean?
- Atoms, actions, mutexes reached **fixpoint**
- Last search yielded *no new information* on conflicting goal atom sets
Graphplan on Trucking

- Assume one single goal atom: \( p1@B \)
- Planning graph is built until \( P_2 \), where goal is first produced
  - Backward search unsuccessful: No valid path exists
Graphplan on Trucking

- Assume one single goal atom: $p1@B$
- Increase graph until $P_3$: Backward search successful
  - “Plan”: $\langle \{\text{nop}_{t@A}, \text{loadp1@A}\}, \{\text{nop}_{p1@T}, \text{driveAtoB}\}, \{\text{dropp1@B}\} \rangle$
Graphplan: Realization

- Use collected mutex information
  - Prune search space: Disallow impossible action choices
  - Add new mutexes when a sub-search fails
Graphplan: Realization

- Use collected mutex information
  - Prune search space: Disallow impossible action choices
  - Add new mutexes when a sub-search fails
- If entire backwards search fails: Remember reason for failing
  - Store conflicting goal atoms as *no-goods* at this layer
  - Just like mutexes, eventually converges
  - Termination criterium for non-trivially unsolvable problems
Graphplan: Realization

- Use collected mutex information
  - Prune search space: Disallow impossible action choices
  - Add new mutexes when a sub-search fails
- If entire backwards search fails: Remember reason for failing
  - Store conflicting goal atoms as no-goods at this layer
  - Just like mutexes, eventually converges
  - Termination criterium for non-trivially unsolvable problems
- Heuristics for search,
  e.g. nop actions first, atom with least enabling actions first
Graphplan: Realization

- Use collected mutex information
  - Prune search space: Disallow impossible action choices
  - Add new mutexes when a sub-search fails
- If entire backwards search fails: Remember reason for failing
  - Store conflicting goal atoms as no-goods at this layer
  - Just like mutexes, eventually converges
  - Termination criterium for non-trivially unsolvable problems
- Heuristics for search,
  e.g. nop actions first, atom with least enabling actions first
- Technical optimizations
  - Preprocess problem, removing all rigid predicates
  - Ground problem while building planning graph, as little as possible
  - When atoms and actions remain unchanged, only update mutexes
Graphplan: Conclusion

What does Graphplan do right?

- Very **careful expansion** of search space, with **polynomial complexity**, as long as possible
  - Also leads to **efficient termination condition** in case of failure
- Maintains **non-trivial logical properties** about problem, useful during entire algorithm
- Explicit ordering of actions only where it matters
Graphplan: Conclusion

What does Graphplan do right?

- Very careful expansion of search space, with polynomial complexity, as long as possible
  - Also leads to efficient termination condition in case of failure
- Maintains non-trivial logical properties about problem, useful during entire algorithm
- Explicit ordering of actions only where it matters

Limitations? Problems?

- Essentially, Graphplan = iterative deepening backwards search (with some additional knowledge)
  - For complex problems, depends on good heuristics again
  - Forward search more common / better practical performances
From Graphplan to SAT

- Graphplan backwards search: Problem of logical nature
  - Resolve causal and set-theoretic dependencies
  - Notion of "learning conflicts" (mutexes) during search

Some Graphplan realizations use CSP (Constraint Satisfaction Problem) or SAT (Satisfiability) solvers
Translation of problem into low-level logical language
Resolution of translated problem using efficient solving techniques
Extraction of found solution

Next up: Use SAT solving as engine for entire planning procedure

[KS+92], [KS96]
From Graphplan to SAT

- Graphplan backwards search: Problem of logical nature
  - Resolve causal and set-theoretic dependencies
  - Notion of “learning conflicts” (mutexes) during search
- Some Graphplan realizations use CSP (Constraint Satisfaction Problem) or SAT (Satisfiability) solvers
  - Translation of problem into low-level logical language
  - Resolution of translated problem using efficient solving techniques
  - Extraction of found solution

Next up: Use SAT solving as engine for entire planning procedure

[KS92], [KS96]
From Graphplan to SAT

- Graphplan backwards search: Problem of logical nature
  - Resolve causal and set-theoretic dependencies
  - Notion of “learning conflicts” (mutexes) during search

- Some Graphplan realizations use CSP (Constraint Satisfaction Problem) or SAT (Satisfiability) solvers
  - Translation of problem into low-level logical language
  - Resolution of translated problem using efficient solving techniques
  - Extraction of found solution

- Next up: Use SAT solving as engine for entire planning procedure [KS⁺92], [KS96]
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).
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* $\mathcal{A}$ maps each variable $v$ to *true* or *false*. If a formula $F$ is *true* under these variable values, $\mathcal{A}$ is a *satisfying assignment*. 
SAT Preliminaries

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 \textit{unsatisfiable}).

- Most prominent \textbf{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 $\textbf{NP}$-complete problem
- Very efficient SAT Solvers exist

SAT Encoding.

Given a problem $\mathcal{P}$, a \textit{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 $F_\mathcal{P}$.
Objective: Find procedure to encode given planning problem as a CNF formula; let SAT solver find a plan for you
Objective: Find procedure to encode given planning problem as a CNF formula; let SAT solver find a plan for you

Theoretical issue with SAT encodings of planning problems: PLANSAT is PSPACE-complete, SAT is NP-complete
  - A single, polynomial-size SAT encoding for general planning problems implies NP = PSPACE (contrary to our best knowledge!)
  - SAT encodings for entire planning problem will explode in size
Towards a SAT Encoding of Planning (2)

General procedure of SAT-based planning:

- 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

\[ \text{Encoder for } n \text{ steps} \]
\[ \text{SAT planner} \]
\[ n := 0 \]
\[ \text{Actions} \]
\[ \text{Initial state} \]
\[ \text{Goal(s)} \]
\[ \text{Decoder} \]
\[ \text{SAT solver} \]
\[ \text{Result?} \]
\[ \text{UNSAT: } n++ \]

\[ \text{Plan: } 1 \cdots 2 \cdots 3 \cdots \]
\[ \text{SAT: } 1 \ 2 \ 3 \ -4 \ -5 \ -6 \ 7 \ 8 \ -9 \ \cdots \]
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 \(t\)”

<table>
<thead>
<tr>
<th>Step</th>
<th>(a)</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Encoded variables: “is” and “do”

- Variable $is_p^t$ for each atom $p$ and each step $t = 0, \ldots, n$
  - “Atom $p$ holds at step $t$”
- Variable $do_a^t$ for each action $a$ and each step $t = 0, \ldots, n - 1$
  - “Action $a$ is applied at step $t$”
Encoded variables: “is” and “do”

- Variable $is_p^t$ for each atom $p$ and each step $t = 0, \ldots, n$
  - “Atom $p$ holds at step $t$”
- Variable $do_a^t$ 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

<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 $t = 0$.

$$\forall p \in s_0 : is^0_p \quad \forall p \not\in s_0 : \neg is^0_p$$
SAT Encoding of Planning (2)

**Clauses** of the encoding:

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

2. At every step, at least one action is applied.
   (Assume $A = \{a_1, \ldots, a_k\}$)
   \[
   \forall t \in \{0, \ldots, n - 1\} : (do_{a_1}^t \lor do_{a_2}^t \lor \ldots do_{a_k}^t)
   \]
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 $A = \{a_1, \ldots, a_k\}$)

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

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

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

$$\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in pre^+(a) : (do^t_a \rightarrow is^t_p)$$

$$\forall t \in \{0, \ldots, n - 1\}, \forall a \in A, \forall p \in pre^-(a) : (do^t_a \rightarrow \neg is^t_p)$$
SAT Encoding of Planning (3)

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

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

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

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

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

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

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

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

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

$$\forall t \in \{0, \ldots, n-1\}, \forall a \in A, \forall p \in \textit{eff}^+(a) : (\textit{do}_a^t \rightarrow \textit{is}_p^{t+1})$$

$$\forall t \in \{0, \ldots, n-1\}, \forall a \in A, \forall p \in \textit{eff}^-(a) : (\textit{do}_a^t \rightarrow \neg \textit{is}_p^{t+1})$$

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

$$\forall p \in g : \textit{is}_p^n$$
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?
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
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

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

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

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

Next lecture: Advanced SAT-based planning; Plan optimization
