COMP3630 / COMP6363

week 12: **Automated (Classical) Planning**

(A subdiscipline of Artificial Intelligence)

slides created by: Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- Introduction to Classical Planning
- Complexity Studies

Disclaimer

Why do we have this week's content?

- I wanted to provide additional examples to strengthen your current understanding rather than including additional content. Compared to $\leq$ 2022 you will miss out on:
  - Approximations: Being guaranteed to be within a factor of *i* to the optimum.
  - Probabilistic Algorithms (and TMs): TMs with error probabilities. (Of course this comes with language classes that we can relate again!)
- To make the point that this isn't just "theory for the sake of having theory", but:
  - its used in disciplines other than Theoretical Computer Science and
  - has actual applications/implications (e.g., algorithm and heuristic ideas/design)
- To promote this exciting discipline! For two purposes:
  - To spread the word! You (or your future boss or colleagues) might be able to use it. Everyboody knows Operations Research (SAT/SMT/ILP solving etc.) to tackle NP-complete problems. But only a fragment knows AI planning for tackling problems beyond NP.
  - To find PhD students! The ANU has at least 8 planning experts, and we are all internationally connected (in case you want to do research Overseas). But note that ANU's Foundations Cluster has just as much staff with theory-heavy topics!

What it is about

We always have:

- An initial world description (start state)
- A desired world description (end state)
- Actions (how can states be changed?)

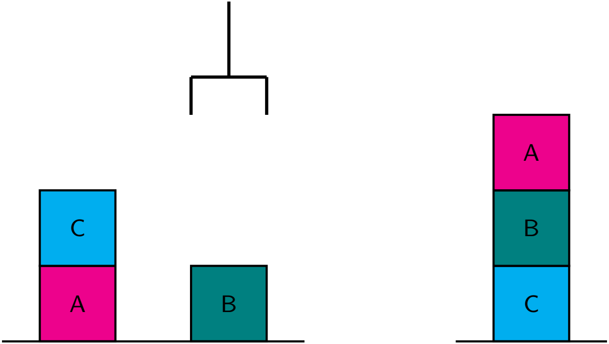
There are tons of variants:

- Do we know/see everything? we: Yes
- Is it entirely clear what an action does? we: Yes
- Are (other) agents involved? we: No
- Can we produce 'objects', use functions? we: No
- Is there time involved? we: No
- Any additional constraints on solution plans? we: No and Yes

  Well... Yes for HTN planning!

Classical Planning is the simplest form of planning! But HTN Planning is more complex.

Artificial Toy Problems, e.g., Blocksworld



Start Configuration Desired Configuration

- Standard Planning Benchmark in the International Planning Competition
- . . . and every planning lecture! (Like this and the one below.)
- Here (`https://www.youtube.com/watch?v=pfNb0IAkbcQ&t=308s`) you find a 90 minute hands-on lecture by me on modeling Blocksword using planning. (I.e., you will actually model it during the lecture and use an online planner to solve it.)

Games, e.g., Solitaire

Source: `https://commons.wikimedia.org/wiki/File:GNOME_Aisleriot_Solitaire.png`

Games, e.g., Rush Hour (or: from practice to games to AI models)

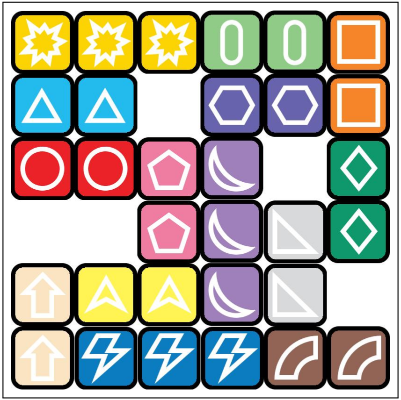
Photo made out of Hanna Neumann (between HN, Birch, CSIT, December 2020).

Games, e.g., Rush Hour (or: from practice to games to AI models)

- Start: any configuration of cars with an exit on one specific side.
- Goal: Get the red car out.

Games, e.g., Rush Hour (or: from practice to games to AI models)

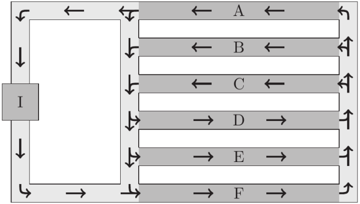
- Start: any configuration of cars with an exit on one specific side.
- Goal: Get the red car out.



Modeling this, including the automated video creation was (is) a 6 pt. project in S1 2023.

Automated Factories (here: Greenhouse)

- Factory takes imagines of all plants, and decides on their further treatments.
- Factory controls their movements via the conveyor belts.

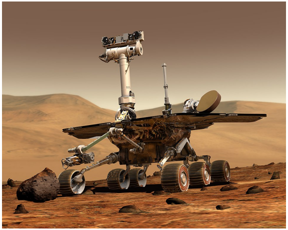


Source: https://www.lemnatec.com/

Further reading:

- Malte Helmert and Hauke Lasinger. "The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem". In: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010). AAAI Press, 2010, pp. 234-237
- The IPC Scanalizer Domain in PDDL (see paper above).

Robotics (here: Mars Rovers Spirit and Opportunity)



Source: left `https://commons.wikimedia.org/wiki/File:KSC-03PD-0786.jpg`
middle `https://commons.wikimedia.org/wiki/File:Curiosity_Self-Portrait_at_%27Big_Sky%27_Drilling_Site.jpg`
right `https://commons.wikimedia.org/wiki/File:NASA_Mars_Rover.jpg`

Further reading:

- Pascal Bercher and Daniel Höller. “Interview with David E. Smith”. In: Künstliche Intelligenz 30.1 (2016). Special Issue on Companion Technologies, pp. 101-105. DOI: 10.1007/s13218-015-0403-y
- `https://www.nasa.gov/` and papers about MAPGEN (for references, see also article above).

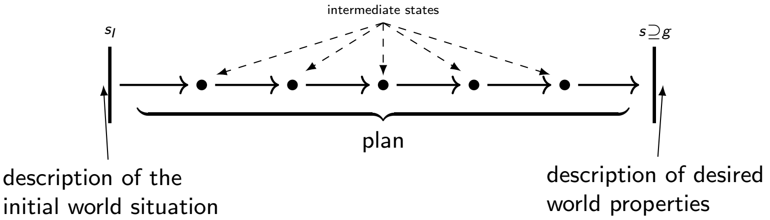
Informal Problem Introduction

We consider classical planning problems, which consist of:

- An initial state s_I – all "world properties" true in the beginning.
- A set of available actions – how world states can be changed.
- A goal description g – all properties we'd like to hold.

What do we want?

$\rightarrow$ Find a plan that transforms s_I into g.



Problem Definition

A classical (or STRIPS) planning problem $\langle V, A, s_I, g \rangle$ consists of:

- V is a finite set of state variables (also called: facts or propositions).
  - States are collections of state variables.
  - We assume the closed world assumption, i.e., all variables not mentioned in a state s do not hold in that state (in contrast to: it's not known whether they hold or not).
  - $S = 2^V$ is called the state space.
- $A \subseteq 2^V \times 2^V \times 2^V$ is a finite set of actions. Each action $a \in A$ is a tuple (pre, add, del) consisting of a precondition pre, add list add, and delete list del.
- $s_I \in S$ is the initial state (complete state description).
- $g \subseteq V$ is the goal description (partial state description).

Q. Something (extremely important) is still missing... What?
A. What a solution is!

Problem Definition, cont'd (Solutions)

Action application:

- An action $a \in A$ is called applicable (or executable) in a state $s \in S$ if and only if $pre(a) \subseteq s$. Often, this is given by a function: $\tau(a, s) \Leftrightarrow pre(a) \subseteq s$.
- If $\tau(a, s)$ holds, its application results into the successor state $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. $\gamma : A \times S \to S$ is called the state transition function.
- An action sequence $\bar{a} = a_0, \ldots, a_{n-1}$ is applicable in a state s_0 if and only if for all $0 \leq i \leq n-1$ a_i is applicable in s_i, where for all $1 \leq i \leq n$ s_i is the resulting state of applying $a_0, \ldots, a_i$ to $s_0 = s_I$. Often, the state transition function is extended to work on action sequences as well $\gamma : A^* \times S \to S$.

Solution:

An action sequence $\bar{a} \in A^*$ consisting of 0 (empty sequence) or more actions is called a plan or solution to a planning problem $\langle V, A, s_I, g\rangle$ if and only if:

- $\bar{a}$ is applicable in s_I.
- $\bar{a}$ results into a goal state, i.e., $\gamma(\bar{a}, s_I) \supseteq g$.

PLANEX $= \{\langle\mathcal{P}\rangle : \mathcal{P}$ is a classical planning problem $\langle V, A, s_I, g\rangle$ that has a solution.$\}$.

Example Problem

Let $s_I = \{\mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{At}_{\mathrm{Garage,Remote}}, \mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{TV}_{\mathrm{Off}}\}$

Rick's actions:

- $\mathrm{PushBox}_R$: $(\{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}\},\{\mathrm{At}_{\mathrm{LivingRoom,M}}\},\emptyset)$
- $\mathrm{GoToGarage}_R$: $(\{\mathrm{At}_{\mathrm{LivingRoom,R}}\},\{\mathrm{At}_{\mathrm{Garage,R}}\},\{\mathrm{At}_{\mathrm{LivingRoom,R}}\})$
- $\mathrm{GoToLivingRoom}_R$: $(\{\mathrm{At}_{\mathrm{Garage,R}}\},\{\mathrm{At}_{\mathrm{LivingRoom,R}}\},\{\mathrm{At}_{\mathrm{Garage,R}}\})$
- $\mathrm{PickUpRemote}_R$: $(\{\mathrm{At}_{\mathrm{Garage,R}}, \mathrm{At}_{\mathrm{Garage,Remote}}\},\{\mathrm{Has}_{\mathrm{Remote,R}}\},\{\mathrm{At}_{\mathrm{Garage,Remote}}\})$
- $\mathrm{TurnTVOn}_R$: $(\{\mathrm{Has}_{\mathrm{Remote,R}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{TV}_{\mathrm{Off}}\},\{\mathrm{TV}_{\mathrm{On}}\},\{\mathrm{TV}_{\mathrm{Off}}\})$

Meeseeks's actions:

- $\mathrm{GoToGarage}_M$: $(\{\mathrm{At}_{\mathrm{LivingRoom,M}}\},\{\mathrm{At}_{\mathrm{Garage,M}}\},\{\mathrm{At}_{\mathrm{LivingRoom,M}}\})$
- $\mathrm{GoToLivingRoom}_M$: $(\{\mathrm{At}_{\mathrm{Garage,M}}\},\{\mathrm{At}_{\mathrm{LivingRoom,M}}\},\{\mathrm{At}_{\mathrm{Garage,M}}\})$
- $\mathrm{PickUpRemote}_M$: $(\{\mathrm{At}_{\mathrm{Garage,M}}, \mathrm{At}_{\mathrm{Garage,Remote}}\},\{\mathrm{Has}_{\mathrm{Remote,M}}\},\{\mathrm{At}_{\mathrm{Garage,Remote}}\})$
- $\mathrm{GiveRemote}_M$: $(\{\mathrm{Has}_{\mathrm{Remote,M}}, \mathrm{At}_{\mathrm{LivingRoom,M}}, \mathrm{At}_{\mathrm{LivingRoom,R}}\},\{\mathrm{Has}_{\mathrm{Remote,R}}\},$
  $\{\mathrm{Has}_{\mathrm{Remote,M}}, \mathrm{At}_{\mathrm{LivingRoom,M}}\})$

$g = \{\mathrm{TV}_{\mathrm{On}}\}$

Example Problem, Solutions

Recap: $s_I = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{At}_{\mathrm{Garage,Remote}}, \mathrm{TV}_{\mathrm{Off}}\}$.

Solution 1 (Rick does it himself):

1. $\mathrm{GoToGarage}_R$: $s_1 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{Garage,R}}, \mathrm{At}_{\mathrm{Garage,Remote}}, \mathrm{TV}_{\mathrm{Off}}\}$
2. $\mathrm{PickUpRemote}_R$: $s_2 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{Garage,R}}, \mathrm{Has}_{\mathrm{Remote,R}}, \mathrm{TV}_{\mathrm{Off}}\}$
3. $\mathrm{GoToLivingRoom}_R$: $s_3 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{Has}_{\mathrm{Remote,R}}, \mathrm{TV}_{\mathrm{Off}}\}$
4. $\mathrm{TurnTVOn}_R$: $s_4 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{Has}_{\mathrm{Remote,R}}, \mathrm{TV}_{\mathrm{On}}\}$

Solution 2 (Rick uses a Meeseeks):

1. $\mathrm{PushBox}_R$: $s_1 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{At}_{\mathrm{Garage,Remote}}, \mathrm{At}_{\mathrm{LivingRoom,M}}, \mathrm{TV}_{\mathrm{Off}}\}$
2. $\mathrm{GoToGarage}_M$: $s_2 = \{\mathrm{At}_{\mathrm{LivingRoom,BMSox}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{At}_{\mathrm{Garage,Remote}}, \mathrm{At}_{\mathrm{Garage,M}}, \mathrm{TV}_{\mathrm{Off}}\}$
3. $\mathrm{PickUpRemote}_M$: $s_3 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{At}_{\mathrm{Garage,M}}, \mathrm{Has}_{\mathrm{Remote,M}}, \mathrm{TV}_{\mathrm{Off}}\}$
4. $\mathrm{GoToLivingRoom}_M$: $s_4 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{At}_{\mathrm{LivingRoom,M}}, \mathrm{Has}_{\mathrm{Remote,M}}, \mathrm{TV}_{\mathrm{Off}}\}$
5. $\mathrm{GiveRemote}_M$: $s_5 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{Has}_{\mathrm{Remote,R}}, \mathrm{TV}_{\mathrm{Off}}\}$
6. $\mathrm{TurnTVOn}_R$: $s_6 = \{\mathrm{At}_{\mathrm{LivingRoom,Box}}, \mathrm{At}_{\mathrm{LivingRoom,R}}, \mathrm{Has}_{\mathrm{Remote,R}}, \mathrm{TV}_{\mathrm{On}}\}$

Recap: $g = \{\mathrm{TV}_{\mathrm{On}}\}$.

Classical Planning is in **PSPACE**

- Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be our plannig problem.
- Note that if a solution $\overline{a}$ exists then one exists with $|\overline{a}| \leq 2^{|V|}$. This is because this is the maximal number of distinct states. If there is a plan that's longer, it "walks in a loop", which can be removed.
- Guess and verify would however be too expensive...
- We want to use recursive doubling! Let $P(s_1, s_2, k)$ represent whether there exists a plan from state s_1 to state s_2 with size $\leq k$.
- We don't have a goal state, but a goal description, so we can't use $P(s_I, g, 2^{|V|})$, since g is just one of potentially exponentially many states. But we can:
  - put a new variable $v_1 \notin V$ into V, now V', and into all action preconditions,
  - create new action $(g, \{v_2\}, V)$, where $v_2 \notin V$ is new.
  - Now $g' = \{v_2\}$ is our unique goal and $\mathcal{P}$ has a solution iff $\mathcal{P}'$ has one.
  - We could also have iterated over all states s with $s \supseteq g$.
- Now we can decide $P(s_I, g', 2^{|V|})$ in the usual way, i.e., $P(s_1, s_2, k)$ iff there exists an s, such that $P(s_1, s, {}^{k}\!/_{2})$ and $P(s, s_2, {}^{k}\!/_{2})$.
- Each state is only polynomially large, and we only need to do this split $\log(2^{|V|})$ often. So we only need poly space to do this search.
- Thus, PLANEX $\in$ **PSPACE**.

Classical Planning is **PSPACE**-hard

We reduce from a poly-space-bounded Turing Machine.

- We define $s_I = \{at_{1,q_0}, in_{0,B}, in_{1,w_1}, \ldots in_{|w|,w_{|w|}}, in_{|w|+1,B}, \ldots, in_{pol(|w|)-1,B}\}$ with
  - $in_{i,x}$ – Symbol x is in tape position i.
  - $at_{i,q}$ – TM's head is over position i and its state is q.
- For the actions, assume TM is in state q, head is over i and reads x, and it shall write y, move right, and transition into q'. This is implemented by three actions, executed in order:
  1. $(\{at_{i,q}, in_{i,x}\}, \{do_{i,q,x}\}, \{at_{i,q}\})$
  2. $(\{do_{i,q,x}, in_{i,x}\}, \{in_{i,y}\}, \{in_{i,x}\})$
  3. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$
  4. don't provide actions for $at_{-1,q}$ and $at_{pol(|w|),q}$ (for any q)

  $\rightarrow$ Uses the new variable(s) $do_{i,q,x}$. Provide the analogous action for left-movement.
- Whenever the TM is in an accepting state, the problem is solved:
  - Set $g = \{accept\}$ (using the new variable $accept$).
  - For all final states $q \in F$ and all i, define $(\{at_{i,q}\}, \{accept\}, \emptyset)$.

Thus, PLANEX is **PSPACE**-complete. (Proof(s) by Bylander, 1994)

Q. Why do we have three actions? Why not just one (as in the live-lecture)?!

A. So that we can say: Planning is even **PSPACE**-hard if we have only 2 preconditions and 2 effects! Think of 2-SAT vs. 3-SAT! (Here, the 2 precs/effs correspond to the 3!)

Optimal (or: Cost-Bounded) Classical Planning is **PSPACE**-complete

$\text{PLANEX}_k = \{\langle \mathcal{P}, k \rangle : \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Note that the k in the index here is a String, i.e., literally the letter k, not a number.
So the k in the set is (clearly) different, since one is a number, the other a letter.

PLANEX_k is **PSPACE**-complete:

- **PSPACE** membership:
  - We know that if a solution exists at all, then one exists up to length $2^{|V|}$.
    Recap: This is because there is no point in repeating any of the $2^{|V|}$ states.
  - We can thus check for plan existence up to the number $\min(k, 2^{|V|})$.
  - We already have a decision procedure for bound $2^{|V|}$, which runs in **PSPACE**.
- We now show **PSPACE**-hardness:
  - We again exploit that if there exists a plan at all, there is one up to length $2^{|V|}$.
  - We thus reduce from PLANEX: We take an arbitrary problem $\mathcal{P} \in$ PLANEX and create a cost-bounded one by choosing $k = 2^{|V|}$, where V are the variables of $\mathcal{P}$. Note that this construction is polytime because we can encode k using only $\log(k)$ bits.

Disclaimer / Recap

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! What about the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT – which is **NP**-complete. How about 2-*CNF-SAT*?
- So, which factor(s) make planning hard? And what if they were not there?
  - If we identify such a special case in a given instance we could use a more efficient algorithm than one designed for the general case.
  - If we can establish a special case we can solve the easier case and use its solution as approximation to the solution of the actual problem. (E.g., as heuristic in a search.)

Delete-free (or Delete-relaxed) Problems, Definition

Reminder: Classical planning problems have the form (V, A, s_I, g).

- A problem (V, A, s_I, g) is called delete-free if the following holds: for all $(pre, add, del) \in A$ holds: $del = \emptyset$
- Given a problem $\mathcal{P} = (V, A, s_I, g)$, we call $\mathcal{P}' = (V', A', s_I', g')$ delete-relaxed version of $\mathcal{P}$ if $V = V'$, $s_I' = s_i$, $g' = g$, and $A' = \{(pre, add, \emptyset) : (pre, add, del) \in A\}$.
- $\text{PLANEX}_{DR} = \{\langle\mathcal{P}\rangle : \mathcal{P}$ is a solvable classical delete-free planning problem.$\}$

Now, what's true?

- PLANEX_{DR} is **PSPACE**-complete (?)
- PLANEX_{DR} is **NP**-complete (?)
- PLANEX_{DR} is in **NP**, not **NP**-hard, and not in **P** (?)
- PLANEX_{DR} is in **P** (?)

Delete-free Planning is in **P**

Algorithm 1: Decision-procedure for delete-free planning.

Data: Set A of delete-free actions, initial state s_I, goal description g
Result: Whether the delete-free problem is solvable

```
s ← s_I;
repeat
    foreach action a ∈ A do
        if pre(a) ⊆ s then
            s = s ∪ add(a);
            delete a from A;
until A is not modified;
return s ⊇ g;
```

Observations:

- Applying an action twice is pointless, so we can delete each applied action.
- Each iteration costs at most $\mathcal{O}(|A|)$ and we can delete at most $|A|$ times.
- Thus, runtime is in $\mathcal{O}(|A|^2)$, so $\text{PLANEX}_{DR} \in \mathbf{P}$.

Cost-bound Delete-Free Planning is in **NP**

$\text{PLANEX}_{k-DR} = \{\langle \mathcal{P}, k \rangle : \mathcal{P}$ is a delete-free planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$
As before: The k in the index is a String, not a number. (To name this problem class.)

We can show $\text{PLANEX}_{k-DR} \in$ **NP**

- Let $\mathcal{P}$ (delete-free problem) and number k be given.
- Guess up to k actions and an order among them.
- Return true if sequence is executable and makes goal true. Right?
- No! That's a **NEXPTIME**-procedure! k is encoded binarily...
  Instead, we limit the number of actions that we guess.
- No action has to be executed twice! So we only guess up to $|A|$ (distinct) actions.
- Thus, we perform the above procedure for the number $\min(k, |A|)$.
- This results in an **NP** membership procedure/proof.

Cost-bound Delete-Free Planning is **NP**-hard

We show that PLANEX_{k-DR} is **NP**-hard.

- We reduce from CNF-SAT.
- Let $\varphi = \underbrace{\{C_1, \ldots, C_n\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \ldots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \ldots, x_m\}}_{\text{variables}}$.
- For each boolean variable $x_i \in V$ add two actions to A:

  $\boxed{x_i \mapsto \top}$ — effects: $x_i - \top$, $x_i - set$

  $\boxed{x_i \mapsto \bot}$ — effects: $x_i - \bot$, $x_i - set$

- For each <u>positive</u> $\varphi_{j_i} = x_{j_i}$ or <u>negative</u> $\varphi_{j_i} = \neg x_{j_i}$ add

  precondition $x_{j_i} - \top$ — $\boxed{\text{“}x_{j_i} = \top\text{” } C_j \mapsto \top}$ — effect $C_j - \top$ or precondition $x_{j_i} - \bot$ — $\boxed{\text{“}x_{j_i} = \bot\text{” } C_j \mapsto \top}$ — effect $C_j - \top$

- $g = \{x_i - set \mid 1 \leq i \leq m\} \cup \{C_j - \top \mid 1 \leq j \leq n\}$
- φ is satisfiable if and only if a plan of size $n + m$ exists.

You are not done yet! Don't forget to show this is a reduction!