COMP3630 / COMP6363

week 11: Examples from Classical Planning

All taken from literature

slides created by: Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2025

Content of this Chapter

- Planning: What assumptions?
- Example Planning Problems
- Propositional Classical Planning:
  - Problem Definition
  - Complexity Results
- Lifted Classical Planning:
  - Problem Definition
  - Complexity Results
- Expressivity Analysis

Disclaimer

Why do we have this week 11 and 12 content?

- I wanted to provide additional examples to strengthen your current understanding rather than including additional content (that was barely ever examined, anyway). Compared to $\leq$ 2022 (I, Pascal, had it in 2023 the first time) 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!)

Disclaimer

Why do we have this week 11 and 12 content?

- I wanted to provide additional examples to strengthen your current understanding rather than including additional content (that was barely ever examined, anyway). Compared to $\leq$ 2022 (I, Pascal, had it in 2023 the first time) 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!)
- **Research-led teaching!** I.e., to show and illustrate that:
  - The content is used in disciplines other than Theoretical Computer Science and
  - has actual applications/implications (e.g., algorithm and heuristic ideas/design)

Disclaimer

Why do we have this week 11 and 12 content?

- I wanted to provide additional examples to strengthen your current understanding rather than including additional content (that was barely ever examined, anyway). Compared to $\leq$ 2022 (I, Pascal, had it in 2023 the first time) 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!)
- **Research-led teaching!** I.e., to show and illustrate that:
  - The content is 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:

Disclaimer

Why do we have this week 11 and 12 content?

- I wanted to provide additional examples to strengthen your current understanding rather than including additional content (that was barely ever examined, anyway). Compared to $\leq$ 2022 (I, Pascal, had it in 2023 the first time) 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!)
- **Research-led teaching!** I.e., to show and illustrate that:
  - The content is 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. Everybody 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**.

Disclaimer

Why do we have this week 11 and 12 content?

- I wanted to provide additional examples to strengthen your current understanding rather than including additional content (that was barely ever examined, anyway). Compared to $\leq$ 2022 (I, Pascal, had it in 2023 the first time) 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!)
- **Research-led teaching!** I.e., to show and illustrate that:
  - The content is 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. Everybody 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!

Automated Planning Introduction

What it is about

We always have:

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

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything?

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything?
- Are action outcomes certain (deterministic)?

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything?
- Are action outcomes certain (deterministic)?
- Are (other) agents involved?

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything?
- Are action outcomes certain (deterministic)?
- Are (other) agents involved?
- Can we produce 'objects' or use functions?

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything?
- Are action outcomes certain (deterministic)?
- Are (other) agents involved?
- Can we produce 'objects' or use functions?
- Do actions have durations?

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything?
- Are action outcomes certain (deterministic)?
- Are (other) agents involved?
- Can we produce 'objects' or use functions?
- Do actions have durations?
- Any additional constraints on solution plans?

What it is about

We always have:

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

There are tons of variants:

- Do we know/see everything? we: Yes
- Are action outcomes certain (deterministic)? we: Yes
- Are (other) agents involved? we: No
- Can we produce 'objects' or use functions? we: No
- Do actions have durations? we: No
- Any additional constraints on solution plans? we: No

Classical Planning is the simplest form of planning!

What it is about

We always have:

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

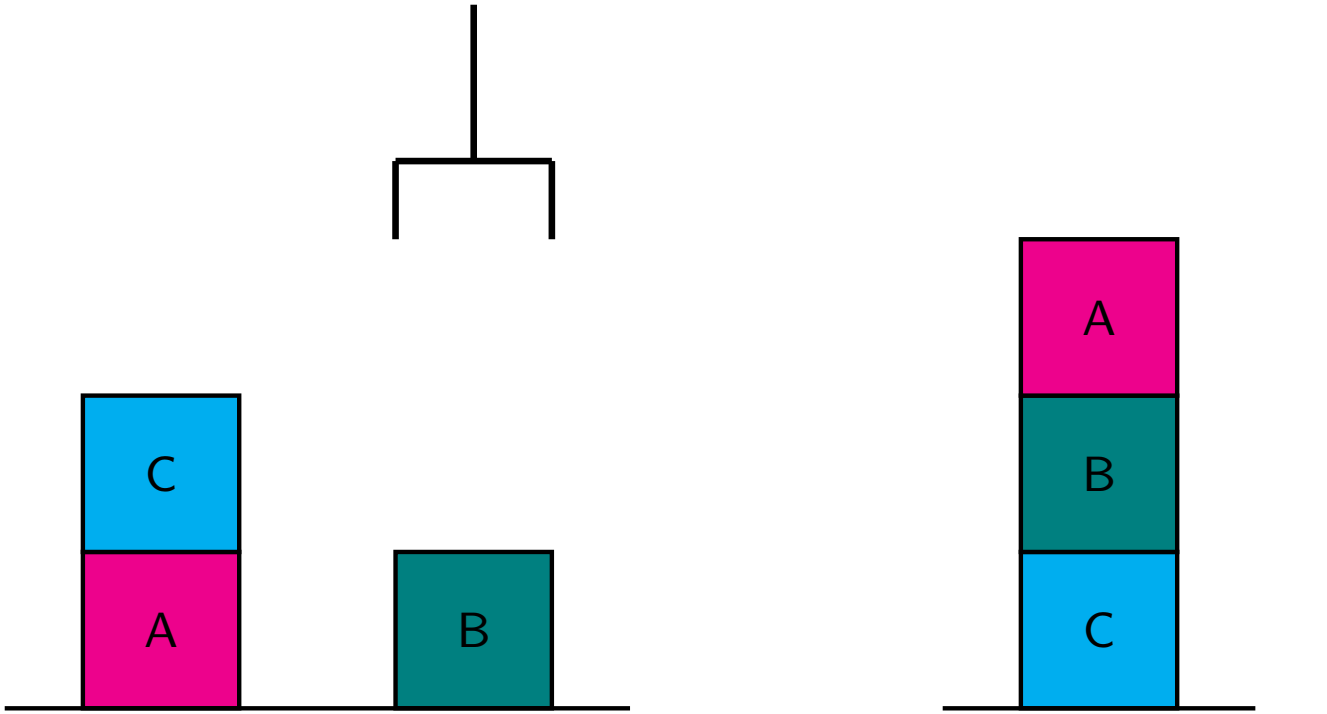
There are tons of variants:

- Do we know/see everything? we: Yes
- Are action outcomes certain (deterministic)? we: Yes
- Are (other) agents involved? we: No
- Can we produce 'objects' or use functions? we: No
- Do actions have durations? 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.

Examples

Artificial Toy Problems, e.g., Blocksworld

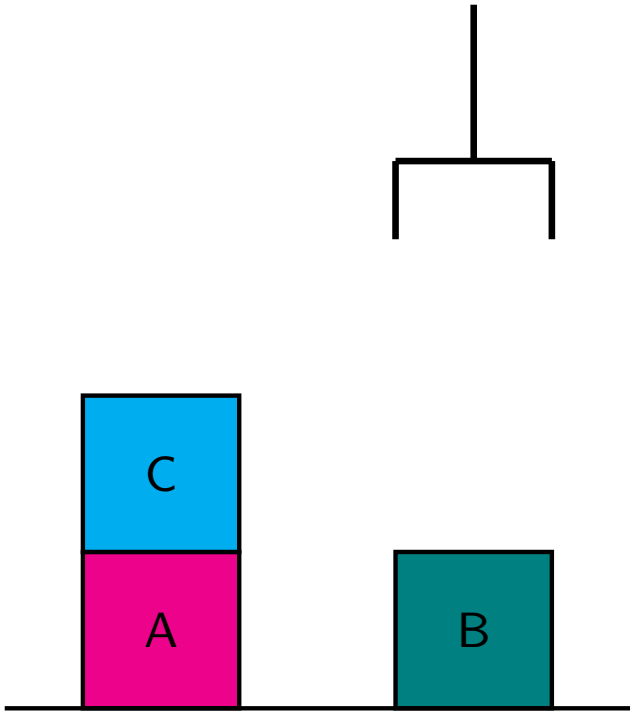


Start Configuration

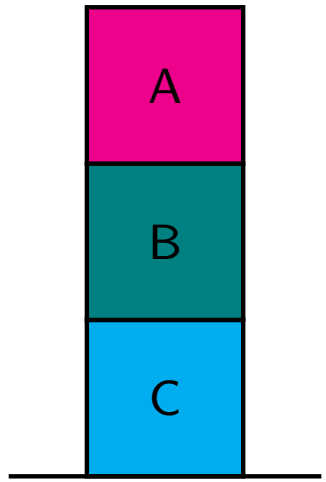
Desired Configuration

- Standard Planning Benchmark in the International Planning Competition

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

License: GNU General Public License v2 or later https://www.gnu.org/licenses/gpl.html

Copyright: Authors of Gnome Aisleriot https://gitlab.gnome.org/GNOME/aisleriot/blob/master/AUTHORS

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

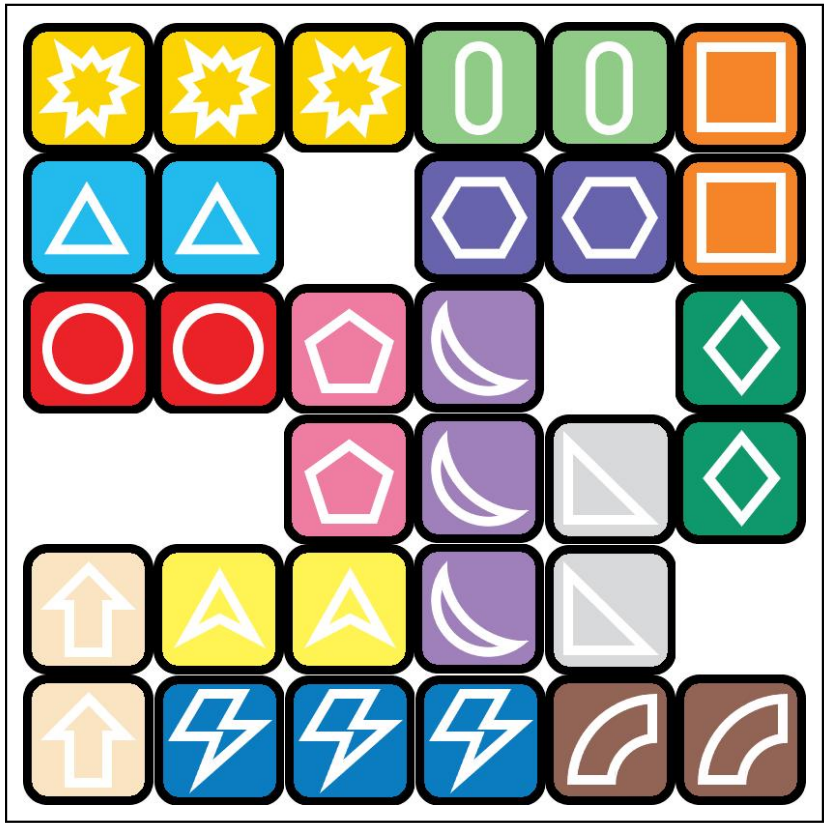
Photo made out of HN between HN, Birch, and SD (Bercher, December 2020).

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 (i.e., any state not containing the red car).

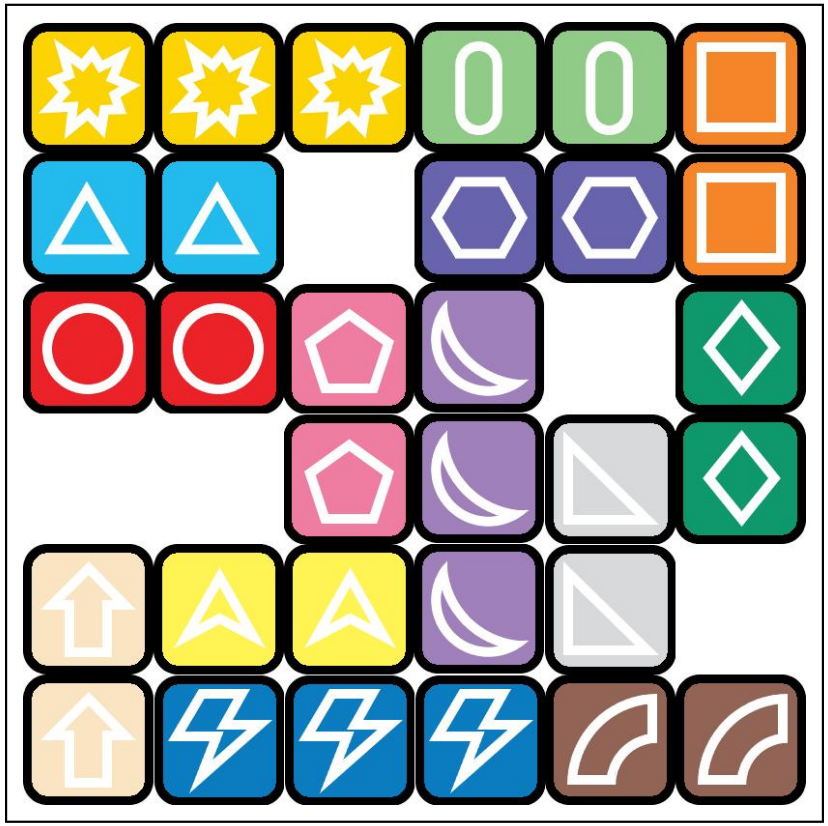
Games, e.g., Rush Hour (or: from practice to games to AI models)

- Start: any configuration of cars with an exit on one specific side.
- Goal: Get the red car out (i.e., any state not containing the red car).



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

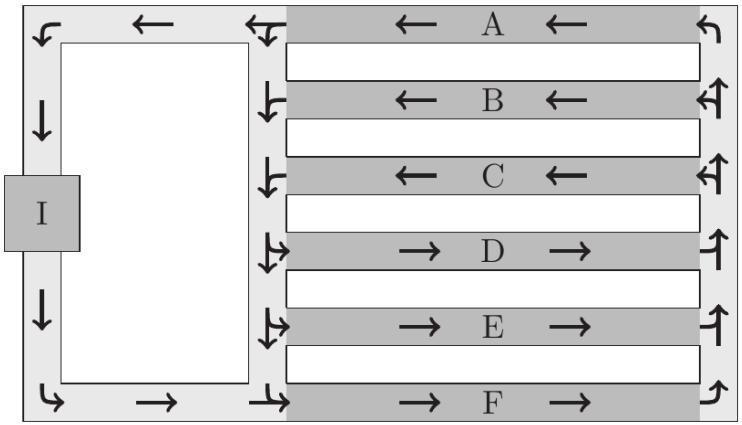
- Start: any configuration of cars with an exit on one specific side.
- Goal: Get the red car out (i.e., any state not containing the red car).



Modeling this, including the automated video creation was 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/`

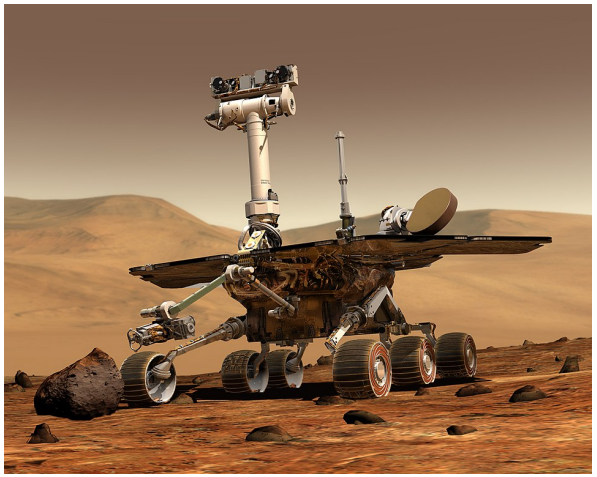
Copyright: With kind permission from LemnaTec GmbH

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`

Copyright: public domain

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).

Propositional Classical Planning

Informal Problem Introduction

We consider classical planning problems, which consist of:

- An initial state s_I – all "world properties" true in the beginning.

What do we want?

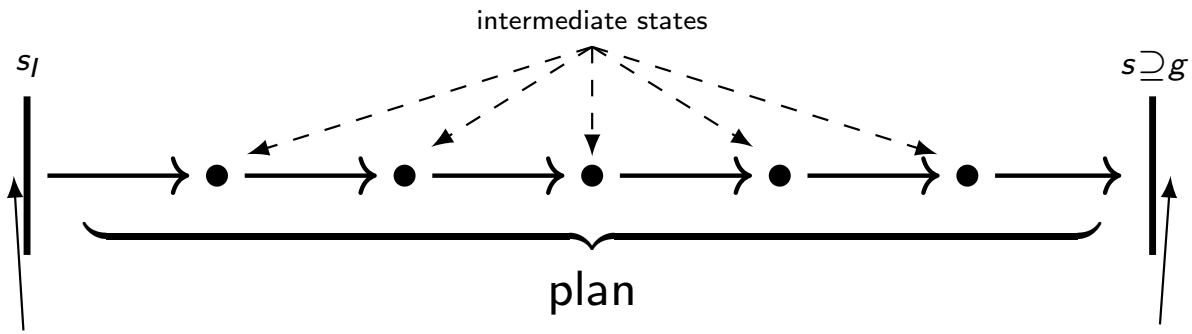
Informal Problem Introduction

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.

What do we want?

→ Find a plan that transforms s_I into g.



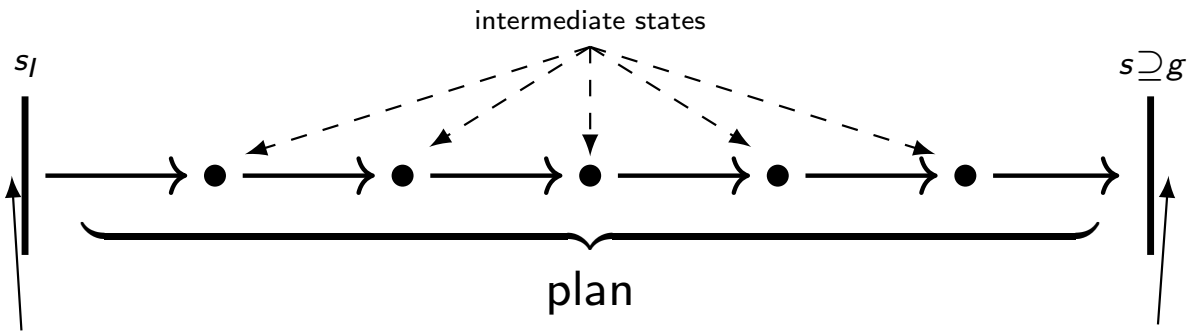
Informal Problem Introduction

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?

→ 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.

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.
- $A \subseteq \Sigma^* \times 2^V \times 2^V \times 2^V$ is a finite set of actions, where Σ is a set of characters. Each action $a \in A$ is a tuple $(name, pre, add, del)$ consisting of a name $name$, precondition pre, add list add, and delete list del. (Called lists, but they are sets.)

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.
- $A \subseteq \Sigma^* \times 2^V \times 2^V \times 2^V$ is a finite set of actions, where Σ is a set of characters. Each action $a \in A$ is a tuple $(name, pre, add, del)$ consisting of a name $name$, precondition pre, add list add, and delete list del. (Called lists, but they are sets.)
- $s_I \in S$ is the initial state (complete state description).
- $g \subseteq V$ is the goal description (partial state description).

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.
- $A \subseteq \Sigma^* \times 2^V \times 2^V \times 2^V$ is a finite set of actions, where Σ is a set of characters. Each action $a \in A$ is a tuple $(name, pre, add, del)$ consisting of a name $name$, precondition pre, add list add, and delete list del. (Called lists, but they are sets.)
- $s_I \in S$ 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 is it?

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.
- $A \subseteq \Sigma^* \times 2^V \times 2^V \times 2^V$ is a finite set of actions, where Σ is a set of characters. Each action $a \in A$ is a tuple $(name, pre, add, del)$ consisting of a name $name$, precondition pre, add list add, and delete list del. (Called lists, but they are sets.)
- $s_I \in S$ 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 is it?
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: $\tau(a, s) \Leftrightarrow pre(a) \subseteq s$.

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.

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 $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. $\gamma : A \times S \rightarrow S$ 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 $\gamma : A^* \times S \rightarrow S$.

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 $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. $\gamma : A \times S \rightarrow S$ 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 $\gamma : A^* \times S \rightarrow S$.

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:

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 $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. $\gamma : A \times S \rightarrow S$ 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 $\gamma : A^* \times S \rightarrow S$.

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., $\gamma(\bar{a}, s_I) \supseteq g$.

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$
- $\text{GoToGarage}_{\text{R}}$: $(\{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\})$
- $\text{GoToLivingRoom}_{\text{R}}$: $(\{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\})$

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$
- $\text{GoToGarage}_{\text{R}}$: $(\{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\})$
- $\text{GoToLivingRoom}_{\text{R}}$: $(\{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\})$
- The next two actions represent four, just replace X by Box and Remote.

  $\text{PickUp}_{X,\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{Garage,X}}\})$
  $\text{PickUp}_{X,\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{LivingRoom,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{LivingRoom,X}}\})$

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$
- $\text{GoToGarage}_{\text{R}}$: $(\{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\})$
- $\text{GoToLivingRoom}_{\text{R}}$: $(\{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\})$
- The next two actions represent four, just replace X by Box and Remote.

  $\text{PickUp}_{X,\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{Garage,X}}\})$

  $\text{PickUp}_{X,\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{LivingRoom,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{LivingRoom,X}}\})$
- Again, these actions represent four, again replace X by Box and Remote.

  $\text{Give}_{X,\text{LivingRoom,R}}$: $(\{\text{Has}(\text{X,R}), \text{At}(\text{LivingRoom,R}), \text{At}(\text{LivingRoom,M})\}, \{\text{Has}(\text{X,M})\}, \{\text{Has}(\text{X,R})\})$

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$
- $\text{GoToGarage}_{\text{R}}$: $(\{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\})$
- $\text{GoToLivingRoom}_{\text{R}}$: $(\{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\})$
- The next two actions represent four, just replace X by Box and Remote.

  $\text{PickUp}_{X,\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{Garage,X}}\})$
  $\text{PickUp}_{X,\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{LivingRoom,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{LivingRoom,X}}\})$
- Again, these actions represent four, again replace X by Box and Remote.

  $\text{Give}_{X,\text{LivingRoom,R}}$: $(\{\text{Has}(\text{X,R}), \text{At}(\text{LivingRoom,R}), \text{At}(\text{LivingRoom,M})\}, \{\text{Has}(\text{X,M})\},$
  $\text{Give}_{X,\text{Garage,R}}$: (Replace LivingRoom by Garage) $\{\text{Has}(\text{X,R})\})$

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

Example: Propositional Classical Planning Problem

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$
- $\text{GoToGarage}_{\text{R}}$: $(\{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\})$
- $\text{GoToLivingRoom}_{\text{R}}$: $(\{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\})$
- The next two actions represent four, just replace X by Box and Remote.

  $\text{PickUp}_{X,\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{Garage,X}}\})$
  $\text{PickUp}_{X,\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{LivingRoom,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{LivingRoom,X}}\})$
- Again, these actions represent four, again replace X by Box and Remote.

  $\text{Give}_{X,\text{LivingRoom,R}}$: $(\{\text{Has}(\text{X,R}), \text{At}(\text{LivingRoom,R}), \text{At}(\text{LivingRoom,M})\}, \{\text{Has}(\text{X,M})\},$
  $\text{Give}_{X,\text{Garage,R}}$: (Replace LivingRoom by Garage) $\{\text{Has}(\text{X,R})\})$
- $\text{TurnTVOn}_{\text{R}}$: $(\{\text{Has}_{\text{Remote,R}}, \text{At}_{\text{LivingRoom,R}}, \text{TV}_{\text{Off}}\}, \{\text{TV}_{\text{On}}\}, \{\text{TV}_{\text{Off}}\})$

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

Example Problem, Solutions

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

Solution 1 (Rick does it himself):

$\text{GoToGarage}_{\text{R}}$: $s_1 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,Remote}}, \text{TV}_{\text{Off}}\}$

$\text{PickUp}_{\text{Remote,Garage,R}}$: $s_2 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{Garage,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{Off}}\}$

GoToLivingRoom_R: $s_3 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{Off}}\}$

$\text{TurnTVOn}_{\text{R}}$: $s_4 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{On}}\}$

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

Example Problem, Solutions

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

Solution 1 (Rick does it himself):

$\text{GoToGarage}_{\text{R}}$: $s_1 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,Remote}}, \text{TV}_{\text{Off}}\}$

$\text{PickUp}_{\text{Remote,Garage,R}}$: $s_2 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{Garage,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{Off}}\}$

$\text{GoToLivingRoom}_{R}$: $s_3 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{Off}}\}$

$\text{TurnTVOn}_{\text{R}}$: $s_4 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{On}}\}$

Solution 2 (Rick uses a Meeseeks):

$\text{PushBox}_{\text{LivingRoom,R}}$: $s_1 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{Garage,Remote}}, \text{At}_{\text{LivingRoom,M}}, \text{TV}_{\text{Off}}\}$

$\text{GoToGarage}_{\text{M}}$: $s_2 = \{\text{At}_{\text{LivingRoom,BMSox}}, \text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{Garage,Remote}}, \text{At}_{\text{Garage,M}}, \text{TV}_{\text{Off}}\}$

$\text{PickUp}_{\text{Remote,Garage,M}}$: $s_3 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{Garage,M}}, \text{Has}_{\text{Remote,M}}, \text{TV}_{\text{Off}}\}$

$\text{GoToLivingRoom}_{M}$: $s_4 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{LivingRoom,M}}, \text{Has}_{\text{Remote,M}}, \text{TV}_{\text{Off}}\}$

$\text{Give}_{\text{Remote,LivingRoom,M}}$: $s_5 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{Off}}\}$

TurnTVOn_{R}: $s_6 = \{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}, \text{Has}_{\text{Remote,R}}, \text{TV}_{\text{On}}\}$

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

Complexity Results for Propositional Planning

Complexity of the General Case

PLANEX $= \{\langle \mathcal{P} \rangle \mid \mathcal{P}$ is a solvable planning problem.$\}$

Theorem w11.1 (Bylander (1994), Thm. 3.1)

PLANEX is **PSPACE***-complete*

Proof overview.

Membership:

Complexity of the General Case

PLANEX $= \{\langle \mathcal{P} \rangle \mid \mathcal{P}$ is a solvable planning problem.$\}$

Theorem w11.1 (Bylander (1994), Thm. 3.1)

PLANEX is **PSPACE***-complete*

Proof overview.

Membership:

- Main idea follows QBF membership proof and Savitch's theorem.
- I.e., recursive doubling: always check for plan existence up to a middle state.
- An alternative proof would be to give an encoding for Sokoban or Rush Hour!

Hardness:

Complexity of the General Case

PLANEX $= \{\langle \mathcal{P} \rangle \mid \mathcal{P}$ is a solvable planning problem.$\}$

Theorem w11.1 (Bylander (1994), Thm. 3.1)

PLANEX is **PSPACE***-complete*

Proof overview.

Membership:

- Main idea follows QBF membership proof and Savitch's theorem.
- I.e., recursive doubling: always check for plan existence up to a middle state.
- An alternative proof would be to give an encoding for Sokoban or Rush Hour!

Hardness:

- We reduce from a polyspace-bounded TM.
- Actions encode valid transitions.

□

Classical Planning is in **PSPACE**

- Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be our planning problem.
- Note that if a solution $\overline{a}$ exists then one exists with $|\overline{a}| \leq 2^{|V|}$. This is because

Classical Planning is in **PSPACE**

- Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be our planning 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 contains a loop, which can be removed.
- Guess and verify would however be too expensive...

Classical Planning is in **PSPACE**

- Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be our planning 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 contains 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.

Classical Planning is in **PSPACE**

- Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be our planning 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 contains 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,
  - put v_1 into s_I and create new action $(g, \{v_2\}, V')$, where $v_2 \notin V$ is also 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 $s \supseteq g$.)

Classical Planning is in **PSPACE**

- Let $\mathcal{P} = \langle V, A, s_I, g \rangle$ be our planning 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 contains 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,
  - put v_1 into s_I and create new action $(g, \{v_2\}, V')$, where $v_2 \notin V$ is also 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 (as we've seen in other proofs).
- Thus, PLANEX $\in$ **PSPACE**.

Classical Planning is **PSPACE**-hard

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

› We define $s_I = \{in_{1,w_1}, \ldots in_{|w|,w_{|w|}}, in_{|w|+1,B}, \ldots, in_{pol(|w|),B}, at_{1,q_0}\}$ 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.

Classical Planning is **PSPACE**-hard

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

- We define $s_I = \{in_{1,w_1}, \ldots in_{|w|,w_{|w|}}, in_{|w|+1,B}, \ldots, in_{pol(|w|),B}, at_{1,q_0}\}$ 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'.

Classical Planning is **PSPACE**-hard

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

- We define $s_I = \{in_{1,w_1}, \ldots in_{|w|,w_{|w|}}, in_{|w|+1,B}, \ldots, in_{pol(|w|),B}, at_{1,q_0}\}$ 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: (other cases analogous)
  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. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$

  $\rightarrow$ don't provide actions for $at_{0,q}$ and $at_{pol(|w|)+1,q}$ (for any q)
  We proved semi-infinite tapes are equivalent to infinite ones.

Classical Planning is **PSPACE**-hard

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

- We define $s_I = \{in_{1,w_1}, \ldots in_{|w|,w_{|w|}}, in_{|w|+1,B}, \ldots, in_{pol(|w|),B}, at_{1,q_0}\}$ 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: (other cases analogous)
  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. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$

  $\rightarrow$ don't provide actions for $at_{0,q}$ and $at_{pol(|w|)+1,q}$ (for any q)
  We proved semi-infinite tapes are equivalent to infinite ones.
- Whenever the TM is in an accepting state, the problem is solved:
  - For all final states $q \in F$ and all i, define $(\{at_{i,q}\}, \{accept\}, \emptyset)$.
  - Set $g = \{accept\}$ (using the new variable $accept$).

Classical Planning is **PSPACE**-hard

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

› We define $s_I = \{in_{1,w_1}, \ldots in_{|w|,w_{|w|}}, in_{|w|+1,B}, \ldots, in_{pol(|w|),B}, at_{1,q_0}\}$ 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: (other cases analogous)

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. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$

→ don't provide actions for $at_{0,q}$ and $at_{pol(|w|)+1,q}$ (for any q)
We proved semi-infinite tapes are equivalent to infinite ones.

› Whenever the TM is in an accepting state, the problem is solved:
- For all final states $q \in F$ and all i, define $(\{at_{i,q}\}, \{accept\}, \emptyset)$.
- Set $g = \{accept\}$ (using the new variable $accept$).

› Also, every plan corresponds to a TM transition into an accepting state.

Thus, PLANEX is **PSPACE**-hard and hence -complete. (Proof(s) by Bylander, 1995)

Recap on the hardness proof

Recap the three actions we had:

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. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$

Q. Could we have provided just one action instead?

Recap on the hardness proof

Recap the three actions we had:

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. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$

Q. Could we have provided just one action instead?
A. Yes! Just stack effects.

Q. So, why might we have used three instead?

Recap on the hardness proof

Recap the three actions we had:

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. $(\{do_{i,q,x}, in_{i,y}\}, \{at_{i+1,q'}\}, \{do_{i,q,x}\})$

Q. Could we have provided just one action instead?
A. Yes! Just stack effects.

Q. So, why might we have used three instead?
A. This allows us to state a stronger result: Planning remains **PSPACE**-hard even if all actions have only 2 preconditions and 2 effects!

Think of 2-SAT vs. 3-SAT! (Here, the 2 precs/effs correspond to the 3!)
So, what about having only 2 preconditions or effects? (Worth exploring...)

Length-Bounded Propositional Planning is **PSPACE**-complete

PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Bylander (1994), Thm. 3.1)

Length-Bounded Propositional Planning is **PSPACE**-complete

PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Bylander (1994), Thm. 3.1)

PLANMIN is **PSPACE***-complete*

Proof.

› **PSPACE** membership:

Length-Bounded Propositional Planning is **PSPACE**-complete

PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Bylander (1994), Thm. 3.1)

PLANMIN is **PSPACE***-complete*

Proof.

› **PSPACE** membership:

- We know that if a solution exists at all, then one exists up to length $2^{|V|}$.
- 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 still need to adapt it slightly to work with non-exponential numbers (e.g., 5).

Length-Bounded Propositional Planning is **PSPACE**-complete

PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Bylander (1994), Thm. 3.1)

PLANMIN is **PSPACE***-complete*

Proof.

- **PSPACE** membership:
  - We know that if a solution exists at all, then one exists up to length $2^{|V|}$.
  - 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 still need to adapt it slightly to work with non-exponential numbers (e.g., 5).
- We now show **PSPACE**-hardness:

Length-Bounded Propositional Planning is **PSPACE**-complete

PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Bylander (1994), Thm. 3.1)

PLANMIN is **PSPACE***-complete*

Proof.

- **PSPACE** membership:
  - We know that if a solution exists at all, then one exists up to length $2^{|V|}$.
  - 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 still need to adapt it slightly to work with non-exponential numbers (e.g., 5).
- 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 $k = 2^{|V|}$. This works because we can encode k using only $\log(k)$ bits.

□

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:
  - Guess a $k' \leq k$, then guess a plan of length k.
  - Verify that plan and return YES if it works and no otherwise.

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:
  - Guess a $k' \leq k$, then guess a plan of length k.
  - Verify that plan and return YES if it works and no otherwise.
- **NP**-hardness:

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:
  - Guess a $k' \leq k$, then guess a plan of length k.
  - Verify that plan and return YES if it works and no otherwise.
- **NP**-hardness:
  - We reduce from 3-SAT. Let it have n variables x_i and m clauses C_i.
  - For each x_i design two actions, one sets $X_i - T$, the other $X_i - F$. Enforce that each plan can contain only one of each.

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:
  - Guess a $k' \leq k$, then guess a plan of length k.
  - Verify that plan and return YES if it works and no otherwise.
- **NP**-hardness:
  - We reduce from 3-SAT. Let it have n variables x_i and m clauses C_i.
  - For each x_i design two actions, one sets $X_i - T$, the other $X_i - F$. Enforce that each plan can contain only one of each.
  - For each clause C_i, have three actions to make C_i true, according to the respective literals. Set the goal to $\{C_1, \ldots, C_m\}$.

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:
  - Guess a $k' \leq k$, then guess a plan of length k.
  - Verify that plan and return YES if it works and no otherwise.
- **NP**-hardness:
  - We reduce from 3-SAT. Let it have n variables x_i and m clauses C_i.
  - For each x_i design two actions, one sets $X_i - T$, the other $X_i - F$. Enforce that each plan can contain only one of each.
  - For each clause C_i, have three actions to make C_i true, according to the respective literals. Set the goal to $\{C_1, \ldots, C_m\}$.
  - How do we set k?

Unary Length-Bounded Classical Planning is **NP**-complete

unary-PLANMIN $= \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k$, and k is encoded unarily (i.e., as a sequence of k 1s) $\}$

Theorem w11.3

unary-PLANMIN is **NP***-complete*

Proof.

- **NP** membership:
  - Guess a $k' \leq k$, then guess a plan of length k.
  - Verify that plan and return YES if it works and no otherwise.
- **NP**-hardness:
  - We reduce from 3-SAT. Let it have n variables x_i and m clauses C_i.
  - For each x_i design two actions, one sets $X_i - T$, the other $X_i - F$. Enforce that each plan can contain only one of each.
  - For each clause C_i, have three actions to make C_i true, according to the respective literals. Set the goal to $\{C_1, \ldots, C_m\}$.
  - How do we set k? To $n + m$, which is polynomial.

□

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)
- What's wrong with those examples?

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)
- What's wrong with those examples?
  - These were no structural special cases, they were single instances.
  - Single instances are always in $\mathcal{O}(1)$, since they don't "scale".
  - Actually, "instances" don't have any complexity. Only languages do.

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)
- What's wrong with those examples?
  - These were no structural special cases, they were single instances.
  - Single instances are always in $\mathcal{O}(1)$, since they don't "scale".
  - Actually, "instances" don't have any complexity. Only languages do.
- What would be "structural restrictions" of SAT?

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)
- What's wrong with those examples?
  - These were no structural special cases, they were single instances.
  - Single instances are always in $\mathcal{O}(1)$, since they don't "scale".
  - Actually, "instances" don't have any complexity. Only languages do.
- What would be "structural restrictions" of SAT?
  - planary 3-SAT (though that's "not really just syntax")
  - 2-SAT, polytime!
  - each clause has at most one positive literal (called Horn), polytime!

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)
- What's wrong with those examples?
  - These were no structural special cases, they were single instances.
  - Single instances are always in $\mathcal{O}(1)$, since they don't "scale".
  - Actually, "instances" don't have any complexity. Only languages do.
- What would be "structural restrictions" of SAT?
  - planary 3-SAT (though that's "not really just syntax")
  - 2-SAT, polytime!
  - each clause has at most one positive literal (called Horn), polytime!
- What would be "structural restrictions" of PLANEX?

Motivation

- We know that – no matter which instance – planning problems are in **PSPACE**.
- But is every instance **PSPACE**-hard?
  - Clearly not! How easy is it to decide PLANEX for the problem $(\emptyset, \emptyset, \emptyset, \emptyset)$?
  - Think of SAT. How hard is it to decide $\neg x_1$? (Not very...)
- What's wrong with those examples?
  - These were no structural special cases, they were single instances.
  - Single instances are always in $\mathcal{O}(1)$, since they don't "scale".
  - Actually, "instances" don't have any complexity. Only languages do.
- What would be "structural restrictions" of SAT?
  - planary 3-SAT (though that's "not really just syntax")
  - 2-SAT, polytime!
  - each clause has at most one positive literal (called Horn), polytime!
- What would be "structural restrictions" of PLANEX?
  - Number of preconditions (see before: hard, even for at most 2).
  - Number of (positive/negative) effects.

Motivation, part 1: The real World

Before you drink:

Motivation, part 1: The real World

Before you drink:

After you drink:

Motivation, part 2: Delete Relaxation

Before you drink:

Motivation, part 2: Delete Relaxation

Before you drink:

After you drink: (both!)

Motivation, part 2: Delete Relaxation

Before you drink:

After you drink: (both!)

The same applies to buying the beer: You have it, but you keep your money!

Delete-free (or Delete-relaxed) Problems

- A problem (V, A, s_I, g) is called delete-free if the following holds: for all $(name, pre, add, del) \in A$ holds: $del = \emptyset$

Delete-free (or Delete-relaxed) Problems

- A problem (V, A, s_I, g) is called delete-free if the following holds: for all $(name, pre, add, del) \in A$ holds: $del = \emptyset$
- Given a problem $\mathcal{P} = (V, A, s_I, g)$, we call $\mathcal{P}' = (V, A', s_I, g)$ its delete-relaxed version of $\mathcal{P}$ if $A' = \{(name, pre, add, \emptyset) \mid (name, pre, add, del) \in A\}$.

Delete-free (or Delete-relaxed) Problems

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

Delete-free Planning is in **P**

Theorem w11.4 (Bylander (1994), footnote without proof on p.4)

$PLANEX_{DR} \in \mathbf{P}$.

Observations:

- Applying an action twice is pointless, so we can delete each applied action.

Delete-free Planning is in **P**

Theorem w11.4 (Bylander (1994), footnote without proof on p.4)

$PLANEX_{DR} \in \mathbf{P}$.

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

$s \leftarrow s_I$;
repeat
  foreach action $a \in A$ **do**
    if $pre(a) \subseteq s$ **then**
      $s = s \cup add(a)$;
      delete a from A;
until A is not modified;
return $s \supseteq g$;

Observations:

- Applying an action twice is pointless, so we can delete each applied action.

Delete-free Planning is in **P**

Theorem w11.4 (Bylander (1994), footnote without proof on p.4)

$PLANEX_{DR} \in \mathbf{P}$.

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

$s \leftarrow s_I$;
repeat
  foreach action $a \in A$ **do**
    if $pre(a) \subseteq s$ **then**
      $s = s \cup add(a)$;
      delete a from A;
until A is not modified;
return $s \supseteq g$;

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 $\mathcal{O}(|A|^2)$.

Length-bound Delete-Free Planning is **NP**-complete

$\text{PLANMIN}_{DR} = \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a delete-free planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.5 (Bylander (1994), Thm. 4.2/Cor. 4.3)

*$PLANMIN_{DR}$ is **NP**-complete.*

Proof overview.

Membership:

Length-bound Delete-Free Planning is **NP**-complete

$\text{PLANMIN}_{DR} = \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a delete-free planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.5 (Bylander (1994), Thm. 4.2/Cor. 4.3)

*$PLANMIN_{DR}$ is **NP**-complete.*

Proof overview.

Membership:

› Guess and Verify proof, as usual.

Hardness:

Length-bound Delete-Free Planning is **NP**-complete

$\text{PLANMIN}_{DR} = \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a delete-free planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.5 (Bylander (1994), Thm. 4.2/Cor. 4.3)

*$PLANMIN_{DR}$ is **NP**-complete.*

Proof overview.

Membership:

- Guess and Verify proof, as usual.

Hardness:

- We again reduce from 3-SAT.
- Very similar to the proof for unary length bounds. But now we don't use delete effects. (Maybe the other proof also didn't use delete effects already, depending on how you implemented it.)

□

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

Membership proof:

- 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.

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

Membership proof:

- 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?

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

Membership proof:

- 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 a number bounded by $\min(k, |A|)$.
- We now have poly-runtime, and thus an **NP** membership procedure/proof.

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

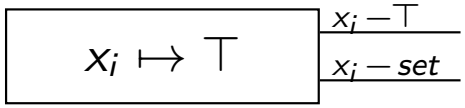
Hardness proof:

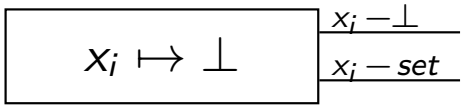
- We reduce from CNF-SAT.
- Let $\varphi = \underbrace{\{C_1, \ldots, C_m\}}_{\text{clauses}}$, $C_j = \underbrace{\{\varphi_{j_1}, \ldots, \varphi_{j_k}\}}_{\text{literals}}$, and $V = \underbrace{\{x_1, \ldots, x_n\}}_{\text{variables}}$.

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

Hardness proof:

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

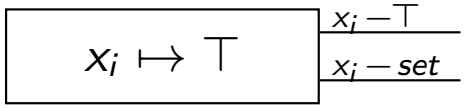


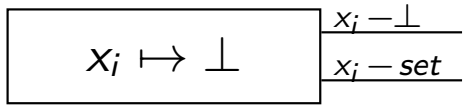


Length-bound Delete-Free Planning is **NP**-hard

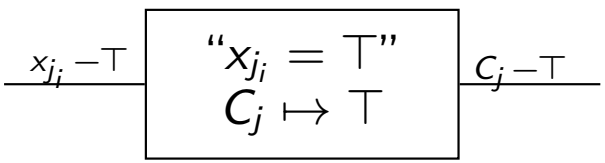
Hardness proof:

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

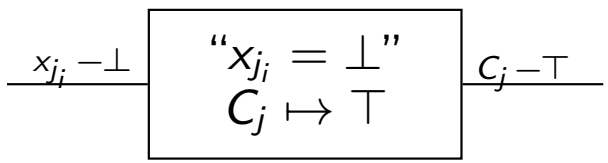




- For each <u>positive</u> $\varphi_{j_i} = x_{j_i}$ or <u>negative</u> $\varphi_{j_i} = \neg x_{j_i}$ add



or



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

Hardness proof:

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

$$\boxed{x_i \mapsto \top} \quad \begin{array}{l} x_i - \top \\ \hline x_i - set \end{array} \qquad \boxed{x_i \mapsto \bot} \quad \begin{array}{l} x_i - \bot \\ \hline x_i - set \end{array}$$

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

$$\frac{x_{j_i} - \top \qquad \boxed{\begin{array}{c} \text{“}x_{j_i} = \top\text{”} \\ C_j \mapsto \top \end{array}}}{C_j - \top} \quad \text{or} \quad \frac{x_{j_i} - \bot \qquad \boxed{\begin{array}{c} \text{“}x_{j_i} = \bot\text{”} \\ C_j \mapsto \top \end{array}}}{C_j - \top}$$

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

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

Hardness proof:

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

$x_i \mapsto \top$ with $x_i - \top$, $x_i - set$ $\qquad$ $x_i \mapsto \bot$ with $x_i - \bot$, $x_i - set$

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

$x_{j_i} - \top$ | "$x_{j_i} = \top$" $C_j \mapsto \top$ | $C_j - \top$ or $x_{j_i} - \bot$ | "$x_{j_i} = \bot$" $C_j \mapsto \top$ | $C_j - \top$

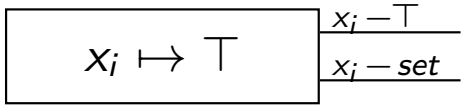
- $g = \{x_i - set \mid 1 \leq i \leq n\} \cup \{C_j - \top \mid 1 \leq j \leq m\}$
- φ is satisfiable if and only if a plan of size $n + m$ exists.

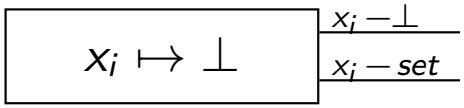
You are not done yet! Don't forget to show this is a reduction!

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

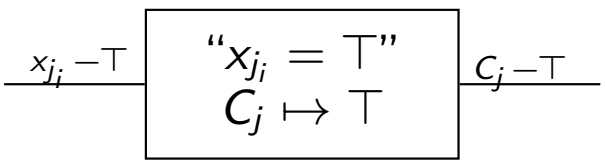
Hardness proof:

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

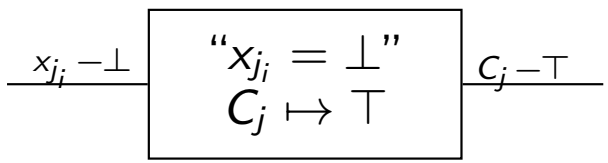




- For each <u>positive</u> $\varphi_{j_i} = x_{j_i}$ or <u>negative</u> $\varphi_{j_i} = \neg x_{j_i}$ add



or



- $g = \{x_i - set \mid 1 \leq i \leq n\} \cup \{C_j - \top \mid 1 \leq j \leq m\}$
- 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!

You could also make this work with only one effect: Create two new actions per x_i with precondition $x_i - \top$ (or $x_i - \bot$, respectively) and effect $x_i - set$. Then, replace n by $2n$.

Lifted Classical Planning

Problem Definition

A lifted classical planning problem $\langle \mathcal{T}, \mathcal{P}, \mathcal{A}, \mathcal{O}, s_I, g \rangle$ consists of:

- $\mathcal{T}$ is a finite set of types. They can form a hierarchy. *Example(s):* character – object
- $\mathcal{P}$ is a finite set of predicate symbols, each with fixed arity, i.e., it takes a sequence of variables, each of some type. *Examples:* At(?room – room,?object – object)

Problem Definition

A lifted classical planning problem $\langle \mathcal{T}, \mathcal{P}, \mathcal{A}, \mathcal{O}, s_I, g \rangle$ consists of:

- $\mathcal{T}$ is a finite set of types. They can form a hierarchy. *Example(s):* character – object
- $\mathcal{P}$ is a finite set of predicate symbols, each with fixed arity, i.e., it takes a sequence of variables, each of some type. *Examples:* At(?room – room,?object – object)
- $\mathcal{A}$ is a finite set of action schemas of the form:

$$(\texttt{name}(\vec{x}), \texttt{pre}(\vec{x}), \texttt{add}(\vec{x}), \texttt{del}(\vec{x}))$$

  where $\vec{x}$ is a list of (typed) variables. *Examples:* next slide!

Problem Definition

A lifted classical planning problem $\langle \mathcal{T}, \mathcal{P}, \mathcal{A}, \mathcal{O}, s_I, g \rangle$ consists of:

- $\mathcal{T}$ is a finite set of types. They can form a hierarchy. *Example(s):* character – object
- $\mathcal{P}$ is a finite set of predicate symbols, each with fixed arity, i.e., it takes a sequence of variables, each of some type. *Examples:* At(?room – room,?object – object)
- $\mathcal{A}$ is a finite set of action schemas of the form:

$$(\texttt{name}(\vec{x}), \texttt{pre}(\vec{x}), \texttt{add}(\vec{x}), \texttt{del}(\vec{x}))$$

  where $\vec{x}$ is a list of (typed) variables. *Examples:* next slide!
- $\mathcal{O}$ is a finite set of (typed) objects used to ground action schemas (and predicates). *Examples:* Rick, Meeseeks – character, Box, Remote – object

Problem Definition

A lifted classical planning problem $\langle \mathcal{T}, \mathcal{P}, \mathcal{A}, \mathcal{O}, s_I, g \rangle$ consists of:

- $\mathcal{T}$ is a finite set of types. They can form a hierarchy. *Example(s):* character – object
- $\mathcal{P}$ is a finite set of predicate symbols, each with fixed arity, i.e., it takes a sequence of variables, each of some type. *Examples:* At(?room – room,?object – object)
- $\mathcal{A}$ is a finite set of action schemas of the form:

$$(\texttt{name}(\vec{x}), \texttt{pre}(\vec{x}), \texttt{add}(\vec{x}), \texttt{del}(\vec{x}))$$

  where $\vec{x}$ is a list of (typed) variables. *Examples:* next slide!
- $\mathcal{O}$ is a finite set of (typed) objects used to ground action schemas (and predicates). *Examples:* Rick, Meeseeks – character, Box, Remote – object
- s_I is the initial state, given as a finite set of ground atoms (complete state).

Problem Definition

A lifted classical planning problem $\langle \mathcal{T}, \mathcal{P}, \mathcal{A}, \mathcal{O}, s_I, g \rangle$ consists of:

- $\mathcal{T}$ is a finite set of types. They can form a hierarchy. *Example(s):* character – object
- $\mathcal{P}$ is a finite set of predicate symbols, each with fixed arity, i.e., it takes a sequence of variables, each of some type. *Examples:* At(?room – room,?object – object)
- $\mathcal{A}$ is a finite set of action schemas of the form:

$$(\texttt{name}(\vec{x}), \texttt{pre}(\vec{x}), \texttt{add}(\vec{x}), \texttt{del}(\vec{x}))$$

  where $\vec{x}$ is a list of (typed) variables. *Examples:* next slide!
- $\mathcal{O}$ is a finite set of (typed) objects used to ground action schemas (and predicates). *Examples:* Rick, Meeseeks – character, Box, Remote – object
- s_I is the initial state, given as a finite set of ground atoms (complete state).
- g is the goal description, a finite set of ground atoms (partial state).

Problem Definition

A lifted classical planning problem $\langle \mathcal{T}, \mathcal{P}, \mathcal{A}, \mathcal{O}, s_I, g \rangle$ consists of:

- $\mathcal{T}$ is a finite set of types. They can form a hierarchy. *Example(s):* character – object
- $\mathcal{P}$ is a finite set of predicate symbols, each with fixed arity, i.e., it takes a sequence of variables, each of some type. *Examples:* At(?room – room,?object – object)
- $\mathcal{A}$ is a finite set of action schemas of the form:

$$(\texttt{name}(\vec{x}), \texttt{pre}(\vec{x}), \texttt{add}(\vec{x}), \texttt{del}(\vec{x}))$$

  where $\vec{x}$ is a list of (typed) variables. *Examples:* next slide!
- $\mathcal{O}$ is a finite set of (typed) objects used to ground action schemas (and predicates). *Examples:* Rick, Meeseeks – character, Box, Remote – object
- s_I is the initial state, given as a finite set of ground atoms (complete state).
- g is the goal description, a finite set of ground atoms (partial state).

Solutions are defined analogously to propositional planning: Any lifted problem is just a compact representation of its ground instantiation – which is equivalent to a propositional problem. (Each ground predicate equals a proposition.)

Example: Propositional Classical Planning Problem (Reminder)

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

Available actions: (Replace R by M to obtain the other actions.)

- $\text{PushBox}_{\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,Box}}, \text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{LivingRoom,M}}\}, \emptyset)$
- $\text{PushBox}_{\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,Box}}, \text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{Garage,M}}\}, \emptyset)$
- $\text{GoToGarage}_{\text{R}}$: $(\{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\})$
- $\text{GoToLivingRoom}_{\text{R}}$: $(\{\text{At}_{\text{Garage,R}}\}, \{\text{At}_{\text{LivingRoom,R}}\}, \{\text{At}_{\text{Garage,R}}\})$
- The next two actions represent four, just replace X by Box and Remote.

  $\text{PickUp}_{X,\text{Garage,R}}$: $(\{\text{At}_{\text{Garage,R}}, \text{At}_{\text{Garage,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{Garage,X}}\})$
  $\text{PickUp}_{X,\text{LivingRoom,R}}$: $(\{\text{At}_{\text{LivingRoom,R}}, \text{At}_{\text{LivingRoom,X}}\}, \{\text{Has}_{\text{X,R}}\}, \{\text{At}_{\text{LivingRoom,X}}\})$
- Again, these actions represent four, again replace X by Box and Remote.

  $\text{Give}_{X,\text{LivingRoom,R}}$: $(\{\text{Has}(\text{X,R}), \text{At}(\text{LivingRoom,R}), \text{At}(\text{LivingRoom,M})\}, \{\text{Has}(\text{X,M})\},$
  $\text{Give}_{X,\text{Garage,R}}$: (Replace LivingRoom by Garage) $\{\text{Has}(\text{X,R})\})$
- $\text{TurnTVOn}_{\text{R}}$: $(\{\text{Has}_{\text{Remote,R}}, \text{At}_{\text{LivingRoom,R}}, \text{TV}_{\text{Off}}\}, \{\text{TV}_{\text{On}}\}, \{\text{TV}_{\text{Off}}\})$

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

Example: Lifted Classical Planning Problem

Types: room, object; character – object (i.e., character is-a object)
Objects: Remote, Box – object; R, M – character; LivingRoom, Garage – room

Let $s_I = \{\text{At}(\text{LivingRoom},\text{R}), \text{At}(\text{Garage},\text{Remote}), \text{At}(\text{LivingRoom},\text{Box}), \text{TV-Off}()\}$

Available action schemata:

$g = \{\text{TV-On}()\}$

Example: Lifted Classical Planning Problem

Types: room, object; character – object (i.e., character is-a object)
Objects: Remote, Box – object; R, M – character; LivingRoom, Garage – room

Let $s_I = \{\text{At}(\text{LivingRoom},\text{R}), \text{At}(\text{Garage},\text{Remote}), \text{At}(\text{LivingRoom},\text{Box}), \text{TV-Off}()\}$

Available action schemata:

› PushBox(?room,?character): $(\{\text{At}(?\text{room},\text{Box}), \text{At}(?\text{room},?\text{character})\}, \{\text{At}(?\text{room},\text{M})\}, \emptyset)$

$g = \{\text{TV-On}()\}$

Example: Lifted Classical Planning Problem

Types: room, object; character – object (i.e., character is-a object)
Objects: Remote, Box – object; R, M – character; LivingRoom, Garage – room

Let $s_I = \{\text{At}(\text{LivingRoom},\text{R}), \text{At}(\text{Garage},\text{Remote}), \text{At}(\text{LivingRoom},\text{Box}), \text{TV-Off}()\}$

Available action schemata:

- PushBox(?room,?character): $(\{\text{At}(?\text{room},\text{Box}), \text{At}(?\text{room},?\text{character})\}, \{\text{At}(?\text{room},\text{M})\}, \emptyset)$
- GoTo(?room-f,?room-t,?character): $(\{\text{At}(?\text{room-f},?\text{character})\}, \{\text{At}(?\text{room-t},?\text{character})\}, \{\text{At}(?\text{room-f},?\text{character})\})$

$g = \{\text{TV-On}()\}$

Example: Lifted Classical Planning Problem

Types: room, object; character – object (i.e., character is-a object)
Objects: Remote, Box – object; R, M – character; LivingRoom, Garage – room

Let $s_I = \{\text{At}(\text{LivingRoom},\text{R}), \text{At}(\text{Garage},\text{Remote}), \text{At}(\text{LivingRoom},\text{Box}), \text{TV-Off}()\}$

Available action schemata:

- PushBox(?room,?character): $(\{\text{At}(\text{?room},\text{Box}), \text{At}(\text{?room},\text{?character})\}, \{\text{At}(\text{?room},\text{M})\}, \emptyset)$
- GoTo(?room-f,?room-t,?character): $(\{\text{At}(\text{?room-f},\text{?character})\}, \{\text{At}(\text{?room-t},\text{?character})\}, \{\text{At}(\text{?room-f},\text{?character})\})$
- PickUp(?object,?room,?character): $(\{\text{At}(\text{?room},\text{?character}), \text{At}(\text{?room},\text{?object})\}, \{\text{Has}(\text{?object},\text{?character})\}, \{\text{At}(\text{?room},\text{?object})\})$

$g = \{\text{TV-On}()\}$

Example: Lifted Classical Planning Problem

Types: room, object; character – object (i.e., character is-a object)
Objects: Remote, Box – object; R, M – character; LivingRoom, Garage – room

Let $s_I = \{\text{At}(\text{LivingRoom},\text{R}), \text{At}(\text{Garage},\text{Remote}), \text{At}(\text{LivingRoom},\text{Box}), \text{TV-Off}()\}$

Available action schemata:

- PushBox(?room,?character): $(\{\text{At}(\text{?room},\text{Box}), \text{At}(\text{?room},\text{?character})\}, \{\text{At}(\text{?room},\text{M})\}, \emptyset)$
- GoTo(?room-f,?room-t,?character): $(\{\text{At}(\text{?room-f},\text{?character})\}, \{\text{At}(\text{?room-t},\text{?character})\}, \{\text{At}(\text{?room-f},\text{?character})\})$
- PickUp(?object,?room,?character): $(\{\text{At}(\text{?room},\text{?character}), \text{At}(\text{?room},\text{?object})\}, \{\text{Has}(\text{?object},\text{?character})\}, \{\text{At}(\text{?room},\text{?object})\})$
- Give(?object,?room,?character-f,?character-t): $(\{\text{Has}(\text{?object},\text{?character-f}), \text{At}(\text{?room},\text{?character-f}), \text{At}(\text{?room},\text{?character-t})\}, \{\text{Has}(\text{?object},\text{?character-t})\}, \{\text{Has}(\text{?object},\text{?character-f})\})$

$g = \{\text{TV-On}()\}$

Example: Lifted Classical Planning Problem

Types: room, object; character – object (i.e., character is-a object)
Objects: Remote, Box – object; R, M – character; LivingRoom, Garage – room

Let $s_I = \{\text{At}(\text{LivingRoom},\text{R}), \text{At}(\text{Garage},\text{Remote}), \text{At}(\text{LivingRoom},\text{Box}), \text{TV-Off}()\}$

Available action schemata:

- PushBox(?room,?character): $(\{\text{At}(?\text{room},\text{Box}), \text{At}(?\text{room},?\text{character})\}, \{\text{At}(?\text{room},\text{M})\}, \emptyset)$
- GoTo(?room-f,?room-t,?character): $(\{\text{At}(?\text{room-f},?\text{character})\}, \{\text{At}(?\text{room-t},?\text{character})\}, \{\text{At}(?\text{room-f},?\text{character})\})$
- PickUp(?object,?room,?character): $(\{\text{At}(?\text{room},?\text{character}), \text{At}(?\text{room},?\text{object})\}, \{\text{Has}(?\text{object},?\text{character})\}, \{\text{At}(?\text{room},?\text{object})\})$
- Give(?object,?room,?character-f,?character-t): $(\{\text{Has}(?\text{object},?\text{character-f}), \text{At}(?\text{room},?\text{character-f}), \text{At}(?\text{room},?\text{character-t})\}, \{\text{Has}(?\text{object},?\text{character-t})\}, \{\text{Has}(?\text{object},?\text{character-f})\})$
- TurnTVOn(?character): $(\{\text{Has}(\text{Remote},?\text{character}), \text{At}(\text{LivingRoom},?\text{character}), \text{TV-Off}()\}, \{\text{TV-On}()\}, \{\text{TV-Off}()\})$

$g = \{\text{TV-On}()\}$

Further Practice for Lifted Planning

Need further practice on lifted planning?

- Check out a 60-minute hands-on lecture on classical planning:
  - video/lecture: `https://www.youtube.com/watch?v=pfNb0IAkbcQ`
  - hands-on material: `https://bercher.net/data/teaching/2022/2022-S1--CCSE--PlanningIntroHandsOn.zip`
- It covers both propositional and lifted planning.
- It teaches how to model in PDDL, approx. 20 to 30 minutes had to be invested into exercises.

Complexity Results for Lifted Planning

Complexity of the General Case

$\text{PLANEX}_L = \{\langle \mathcal{P} \rangle \mid \mathcal{P} \text{ is a solvable lifted planning problem.}\}$

Theorem w11.1 (Erol et al. (1991), Thm. 5.7)

$PLANEX_L$ is **EXPSPACE***-complete*

Proof.

Membership:

Complexity of the General Case

$\text{PLANEX}_L = \{\langle \mathcal{P} \rangle \mid \mathcal{P} \text{ is a solvable lifted planning problem.}\}$

Theorem w11.1 (Erol et al. (1991), Thm. 5.7)

$PLANEX_L$ is **EXPSPACE***-complete*

Proof.

Membership:

- Generate the fully instantiated ground problem, which is exponentially larger. (An action schema with o objects and arity a has o^a many ground action instantiations.)
- Then, decide it in **PSPACE** w.r.t. the new size, which gives **EXPSPACE** membership.

Hardness:

Complexity of the General Case

$\text{PLANEX}_L = \{\langle \mathcal{P} \rangle \mid \mathcal{P} \text{ is a solvable lifted planning problem.}\}$

Theorem w11.1 (Erol et al. (1991), Thm. 5.7)

$PLANEX_L$ is **EXPSPACE***-complete*

Proof.

Membership:

- Generate the fully instantiated ground problem, which is exponentially larger. (An action schema with o objects and arity a has o^a many ground action instantiations.)
- Then, decide it in **PSPACE** w.r.t. the new size, which gives **EXPSPACE** membership.

Hardness:

- We reduce from an expspace-bounded TM.
- Exact encoding given in tutorials. (Follows similar idea as **PSPACE** hardness proof.)

□

Length-Bounded Lifted Planning is **NEXPTIME**-complete

$\text{PLANMIN}_L = \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a lifted planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Erol et al. (1991), Thm. 5.12)

$PLANMIN_L$ is **NEXPTIME***-complete*

Proof.

Membership:

Length-Bounded Lifted Planning is **NEXPTIME**-complete

$\text{PLANMIN}_L = \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a lifted planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Erol et al. (1991), Thm. 5.12)

PLANMIN_L is **NEXPTIME**-*complete*

Proof.

Membership:

- Guess a number $k' \leq k$. Guess a plan of length k, which is in $\mathcal{O}(2^{|k|})$.
- Return yes iff if the plan was successful.

Hardness:

Length-Bounded Lifted Planning is **NEXPTIME**-complete

$\text{PLANMIN}_L = \{\langle \mathcal{P}, k \rangle \mid \mathcal{P}$ is a lifted planning problem with a solution $\bar{a}$, $|\bar{a}| \leq k.\}$

Theorem w11.2 (Erol et al. (1991), Thm. 5.12)

PLANMIN_L is **NEXPTIME***-complete*

Proof.

Membership:

- Guess a number $k' \leq k$. Guess a plan of length k, which is in $\mathcal{O}(2^{|k|})$.
- Return yes iff if the plan was successful.

Hardness:

- We reduce from an exponential-time-bounded non-deterministic TM.
- Exact encoding given in tutorials. (Follows similar idea as **PSPACE** hardness proof.)

□

Expressivity Analysis

Language of a Planning Problem

Recap: A language L is a set of strings over symbols.

- Let $\mathcal{P}$ be a (classical) planning problem and $sol(\mathcal{P})$ its set of solutions. If we interpret any action as a symbol, then $sol(\mathcal{P})$ is a language!
- Recall that every action had a name, which was a string. We didn't require it to be unique, but that was just for the sake of simplicity!

Language of a Planning Problem

Recap: A language L is a set of strings over symbols.

- Let $\mathcal{P}$ be a (classical) planning problem and $sol(\mathcal{P})$ its set of solutions. If we interpret any action as a symbol, then $sol(\mathcal{P})$ is a language!
- Recall that every action had a name, which was a string. We didn't require it to be unique, but that was just for the sake of simplicity!
- So, we now assume that each action (3-tuple of preconditions/effects) can be represented by their name.
- Thus, each plan can equivalently be described by its action name sequence.
- So, we can define: $L(\mathcal{P}) := sol(\mathcal{P})$ if $\mathcal{P}$ is a classical problem.
- We can now compare planning problems (and their special cases) with regard to the Chomsky Hierarchy.

Chomsky Hierarchy

We can define the following Language classes:

- Let $\mathcal{A}ll = \{L \mid L \text{ is a language}\}$
- Let $\mathcal{CSL} = \{L \mid L \text{ is a context-sensitive language}\}$
- Let $\mathcal{CF} = \{L \mid L \text{ is a context-free language}\}$
- Let $\mathcal{R}eg = \{L \mid L \text{ is a regular language}\}$

Chomsky Hierarchy, extended

We can define the following Language classes:

- Let $\mathcal{All} = \{L \mid L \text{ is a language}\}$
- Let $\mathcal{CSL} = \{L \mid L \text{ is a context-sensitive language}\}$
- Let $\mathcal{CF} = \{L \mid L \text{ is a context-free language}\}$
- Let $\mathcal{Reg} = \{L \mid L \text{ is a regular language}\}$
- Let $\mathcal{CLASSIC} = \{L(\mathcal{P}) \mid \mathcal{P} \text{ is a (propositional) classical planning problem.}\}$

We know that $\mathcal{Reg} \subsetneq \mathcal{CF} \subsetneq \mathcal{CSL} \subsetneq \mathcal{All}$.

So ... Where does $\mathcal{CLASSIC}$ sit?

Expressivity of Classical Problems

Theorem w11.1 (Höller et al. (2016), Thm. 1, Thm. 2)

$\mathcal{CLASSIC} \subsetneq \mathcal{R}eg$

Proof.

We first show $\mathcal{CLASSIC} \subseteq \mathcal{R}eg$.

Expressivity of Classical Problems

Theorem w11.1 (Höller et al. (2016), Thm. 1, Thm. 2)

$\mathcal{CLASSIC} \subsetneq \mathcal{Reg}$

Proof.

We first show $\mathcal{CLASSIC} \subseteq \mathcal{Reg}$.

› For this, notice that each planning problem $\mathcal{P}$ encodes a DFA D.

Expressivity of Classical Problems

Theorem w11.1 (Höller et al. (2016), Thm. 1, Thm. 2)

$\mathcal{CLASSIC} \subsetneq \mathcal{Reg}$

Proof.

We first show $\mathcal{CLASSIC} \subseteq \mathcal{Reg}$.

- For this, notice that each planning problem $\mathcal{P}$ encodes a DFA D.
  - The nodes are states, the edges are action names.
  - D is exponentially larger than the propositional planning problem $\mathcal{P}$ (it's "factored"), or doubly-exponentially larger when lifted. (But still just a DFA.)
  - Solving a $\mathcal{P}$ means finding a path in D from the initial state to a goal state.

Expressivity of Classical Problems

Theorem w11.1 (Höller et al. (2016), Thm. 1, Thm. 2)

$\mathcal{CLASSIC} \subsetneq \mathcal{Reg}$

Proof.

We first show $\mathcal{CLASSIC} \subseteq \mathcal{Reg}$.

- For this, notice that each planning problem $\mathcal{P}$ encodes a DFA D.
  - The nodes are states, the edges are action names.
  - D is exponentially larger than the propositional planning problem $\mathcal{P}$ (it's "factored"), or doubly-exponentially larger when lifted. (But still just a DFA.)
  - Solving a $\mathcal{P}$ means finding a path in D from the initial state to a goal state.
- We know that each DFA describes a regular language, thus showing the claim.

We now show $\mathcal{CLASSIC} \subsetneq \mathcal{Reg}$.

Expressivity of Classical Problems

Theorem w11.1 (Höller et al. (2016), Thm. 1, Thm. 2)

$\mathcal{CLASSIC} \subsetneq \mathcal{R}eg$

Proof.

We first show $\mathcal{CLASSIC} \subseteq \mathcal{R}eg$.

- For this, notice that each planning problem $\mathcal{P}$ encodes a DFA D.
  - The nodes are states, the edges are action names.
  - D is exponentially larger than the propositional planning problem $\mathcal{P}$ (it's "factored"), or doubly-exponentially larger when lifted. (But still just a DFA.)
  - Solving a $\mathcal{P}$ means finding a path in D from the initial state to a goal state.
- We know that each DFA describes a regular language, thus showing the claim.

We now show $\mathcal{CLASSIC} \subsetneq \mathcal{R}eg$.

- We prove that for all $\mathcal{P}$, $L(\mathcal{P}) \neq \{aa\}$. (But $\{aa\} \in \mathcal{R}eg$.)

Expressivity of Classical Problems

Theorem w11.1 (Höller et al. (2016), Thm. 1, Thm. 2)

$\mathcal{CLASSIC} \subsetneq \mathcal{Reg}$

Proof.

We first show $\mathcal{CLASSIC} \subseteq \mathcal{Reg}$.

- For this, notice that each planning problem $\mathcal{P}$ encodes a DFA D.
  - The nodes are states, the edges are action names.
  - D is exponentially larger than the propositional planning problem $\mathcal{P}$ (it's "factored"), or doubly-exponentially larger when lifted. (But still just a DFA.)
  - Solving a $\mathcal{P}$ means finding a path in D from the initial state to a goal state.
- We know that each DFA describes a regular language, thus showing the claim.

We now show $\mathcal{CLASSIC} \subsetneq \mathcal{Reg}$.

- We prove that for all $\mathcal{P}$, $L(\mathcal{P}) \neq \{aa\}$. (But $\{aa\} \in \mathcal{Reg}$.)
- Assume $aa \in L(\mathcal{P})$ for some classical problem $\mathcal{P}$. Show that $aaa \in L(\mathcal{P})$.
- Since a is applicable in the state after executing a in s_I, $pre(a)$ must be contained in the state resulting from aa. But then, aaa is also executable.

□

Summary

Summary and Conclusions

We saw/know:

- DFAs are exactly regular, yet classical problems are not, so even DFAs are more expressive than classical problems.
- Finding a word in the language of a DFA is in **P**.
- Yet, for – the less expressive – classical planning, it's between **PSPACE** and **EXPSPACE**.
- So, there is no direct relationship between expressivity and computational complexity.

Summary and Conclusions

We saw/know:

- DFAs are exactly regular, yet classical problems are not, so even DFAs are more expressive than classical problems.
- Finding a word in the language of a DFA is in **P**.
- Yet, for – the less expressive – classical planning, it's between **PSPACE** and **EXPSPACE**.
- So, there is no direct relationship between expressivity and computational complexity.

Next week, we will:

- Investigate HTN planning, which adds more constraints on solutions.
- Specifically,
  - Complexity Results
  - Expressivity Results

Literture/References

Complexity Results in Planning:

- *The Computational Complexity of Propositional STRIPS Planning* by T. Bylander, AIJ 1994: is the "complexity compendium" for propositional planning.
- *Complexity, Decidability and Undecidability Results for Domain-Independent Planning* by K. Erol et al., AIJ 1995 and techreport 1991: complexities for lifted planning. (The '91 work contains the proofs for the results mentioned in the '95 work.)

Expressivity Results in Planning:

- *Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages* by D. Höller et al., ICAPS 2016: the title says it all.