COMP3630 / COMP6363

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

(A subdiscipline of Artificial Intelligence)

slides created by: Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- Introduction to Hierarchical Planning
- Complexity Studies
- Expressivity Studies (also for Classical Planning)

Terminology and some Background

- HTN Planning is short for Hierarchical Task Network Planning.

Terminology and some Background

- HTN Planning is short for Hierarchical Task Network Planning.
- It's an extension of classical planning where:
  - We don't plan for some goal but want to refine some initial tasks.
  - We also can't insert actions in every state, but need to adhere certain rules.

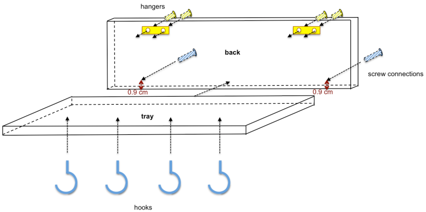
Terminology and some Background

- HTN Planning is short for Hierarchical Task Network Planning.
- It's an extension of classical planning where:
  - We don't plan for some goal but want to refine some initial tasks.
  - We also can't insert actions in every state, but need to adhere certain rules.
- Historical remarks:
  - Whereas first versions date back to the 70s, the first decent formalization comes from the early 90s.
  - Some central idea was to introduce expert knowledge: What do we need to do to achieve a certain task?

Terminology and some Background

- HTN Planning is short for Hierarchical Task Network Planning.
- It's an extension of classical planning where:
  - We don't plan for some goal but want to refine some initial tasks.
  - We also can't insert actions in every state, but need to adhere certain rules.
- Historical remarks:
  - Whereas first versions date back to the 70s, the first decent formalization comes from the early 90s.
  - Some central idea was to introduce expert knowledge: What do we need to do to achieve a certain task?
- Why defining/solving a <u>hierarchical</u> problem?
  - As above: In many real-world applications, knowledge is given in form of control rules: we know the steps required to perform some task.
  - More control on the generated plans, since all the "rules" need to be obeyed. We can <u>exclude</u> (more) undesired plans!
  - Plans can be presented more abstract by relying on task hierarchies.
  - We can solve/express more complex problems! (Spoiler)

Example: Do-It-Yourself (DIY) Assistant, The Task

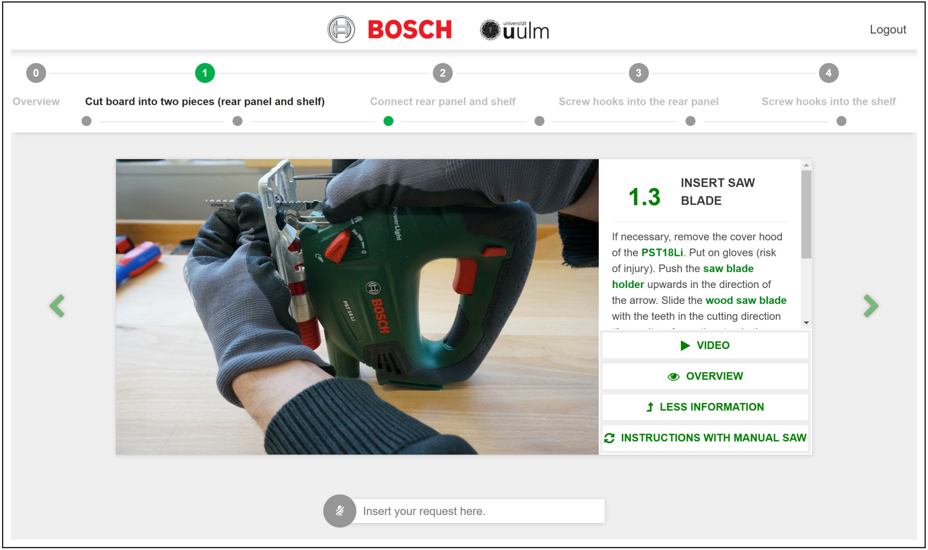


The material:

- Boards (need to be cut first)
- Electrical devices like drills and saws
- Attachments like drill bits and materials like nails

Further reading: Pascal Bercher et al. "Do It Yourself, but Not Alone: Companion-Technology for Home Improvement – Bringing a Planning-Based Interactive DIY Assistant to Life." Künstliche Intelligenz – Special Issue on NLP and Semantics, 35: 367–375. 2021.

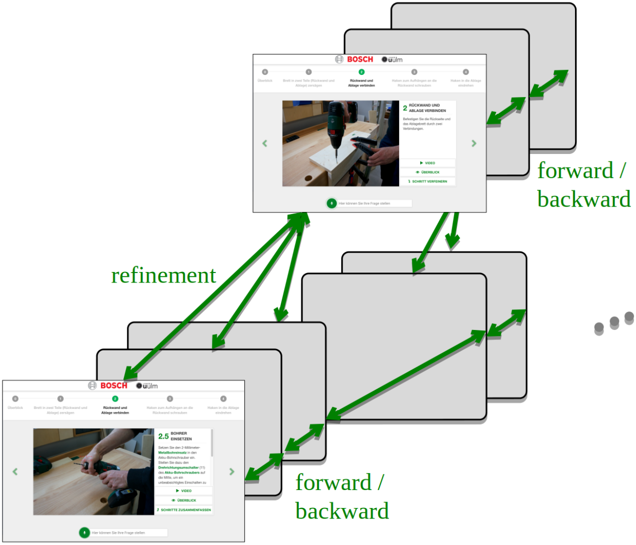
Example: Do-It-Yourself (DIY) Assistant, User Interface



Example: Do-It-Yourself (DIY) Assistant, Task Hierarchy

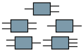
Abstract Level

Detailed Level



Introduction to HTN Planning

primitive tasks



compound tasks

$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \to (2^V)^3$ the task name mapping
- C a set of compound task names

Introduction to HTN Planning

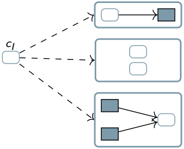
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \to (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task

A solution task network *tn* must:

- be a refinement of c_I,

Introduction to HTN Planning



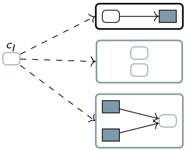
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \to (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



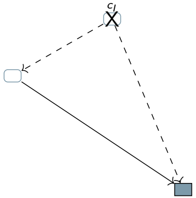
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \to (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



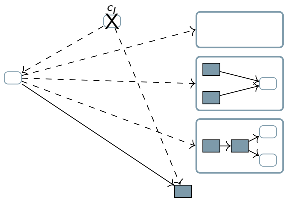
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \to (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



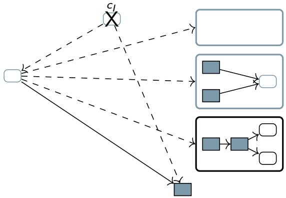
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \rightarrow (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



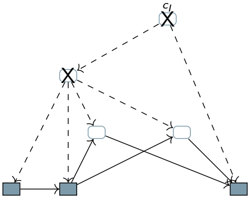
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \rightarrow (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



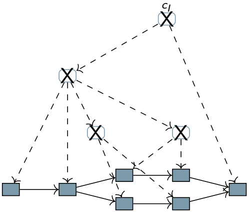
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \rightarrow (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



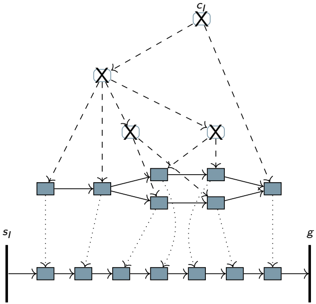
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \rightarrow (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and

Introduction to HTN Planning



$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- V a set of state variables
- P a set of primitive task names
- $\delta : P \to (2^V)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- $s_I \in 2^V$ the initial state
- $g \subseteq V$ the (optional) goal description

A solution task network *tn* must:

- be a refinement of c_I,
- only contain primitive tasks, and
- have an executable linearization that makes the goals in g true.

HTN Planning, Problem Definition

- Task network: $tn = (T, \prec, \alpha)$ consists of:
  - T, a possibly empty set of task identifier symbols (IDs)
  - $\prec \subseteq T \times T$, a partial order on the task IDs,
  - $\alpha : T \to P \dot{\cup} C$, the task mapping function

HTN Planning, Problem Definition

- Task network: $tn = (T, \prec, \alpha)$ consists of:
  - T, a possibly empty set of task identifier symbols (IDs)
  - $\prec \subseteq T \times T$, a partial order on the task IDs,
  - $\alpha : T \to P \dot{\cup} C$, the task mapping function
- Executability of primitive tasks is defined as in classical planning.
  (But since we deal with task networks we also demand that the respective (primitive) task network possesses an executable linearization that makes the goals true.)

HTN Planning, Problem Definition

- Task network: $tn = (T, \prec, \alpha)$ consists of:
  - T, a possibly empty set of task identifier symbols (IDs)
  - $\prec \subseteq T \times T$, a partial order on the task IDs,
  - $\alpha : T \rightarrow P \dot{\cup} C$, the task mapping function
- Executability of primitive tasks is defined as in classical planning. (But since we deal with task networks we also demand that the respective (primitive) task network possesses an executable linearization that makes the goals true.)
- A decomposition method $m \in M$ is a tuple $m = (c, tn_m)$ with a compound task c and task network $tn_m = (T_m, \prec_m, \alpha_m)$

HTN Planning, Problem Definition

- Task network: $tn = (T, \prec, \alpha)$ consists of:
  - T, a possibly empty set of task identifier symbols (IDs)
  - $\prec \subseteq T \times T$, a partial order on the task IDs,
  - $\alpha : T \rightarrow P \dot{\cup} C$, the task mapping function
- Executability of primitive tasks is defined as in classical planning. (But since we deal with task networks we also demand that the respective (primitive) task network possesses an executable linearization that makes the goals true.)
- A decomposition method $m \in M$ is a tuple $m = (c, tn_m)$ with a compound task c and task network $tn_m = (T_m, \prec_m, \alpha_m)$
- Let $tn = (T, \prec, \alpha)$ be a task network, $t \in T$ a task identifier, and $\alpha(t) = c$ a compound task to be decomposed by $m = (c, tn_m)$. We assume $T \cap T_m = \emptyset$. Then, the application of m to tn results into the task network $tn' = ((T \setminus \{t\}) \cup T_m, \prec \cup \prec_m \cup \prec_X, \alpha \cup \alpha_m)|_{(T \setminus \{t\}) \cup T_m}$ with:

$$\begin{aligned} \prec_X := & \{(t', t'') \mid (t', t) \in \prec, t'' \in T_m\} \cup \\ & \{(t'', t') \mid (t, t') \in \prec, t'' \in T_m\} \end{aligned}$$

  where $(X_1, \ldots, X_n)|_Y$ restricts the sets X_i to elements in Y.

HTN Planning, Problem Definition (Solution Criteria)

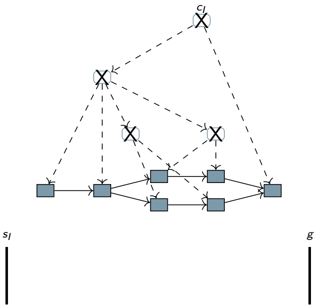
A task network *tn* is a solution if and only if:

- There is a sequence of decomposition methods $\overline{m}$ that transforms c_I into *tn*,

HTN Planning, Problem Definition (Solution Criteria)

A task network *tn* is a solution if and only if:

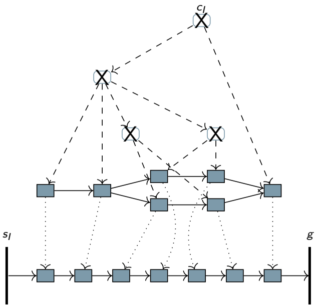
- There is a sequence of decomposition methods $\overline{m}$ that transforms c_I into *tn*,
- *tn* contains only primitive tasks, and



HTN Planning, Problem Definition (Solution Criteria)

A task network *tn* is a solution if and only if:

- There is a sequence of decomposition methods $\overline{m}$ that transforms c_I into *tn*,
- *tn* contains only primitive tasks, and
- the (still partially ordered) task network *tn* admits an executable linearization $\bar{t}$ of its tasks leading to some state $s \supseteq g$.



HTN Planning is in RE

To prove RE membership, we give a partial decision procedure.

There are (at least) two:

1. Systematically generate all refinements (e.g., via "progression search").
   Accept if we found an executable one.
2. Systematically generate all action sequences and verify whether they are executable and can be generated by the task hierarchy (via "plan verification").
   Accept if we found an executable one that can be generated by the hierarchy.

Details omitted.

HTN Planning is not in R

We reduce from the (undecidable) grammar intersection problem. (Chap.9, slide 31)

More specifically, deploy HTN planning to solve the following problem: Given context-free grammars G and G', is $L(G) \cap L(G') = \emptyset$?

HTN Planning is not in R

We reduce from the (undecidable) grammar intersection problem. (Chap.9, slide 31)

More specifically, deploy HTN planning to solve the following problem: Given context-free grammars G and G', is $L(G) \cap L(G') = \emptyset$?

Decision procedure:

- Construct an HTN planning problem $\mathcal{P}$ that has a solution if and only if the correct answer is yes.

HTN Planning is not in R

We reduce from the (undecidable) grammar intersection problem. (Chap.9, slide 31)

More specifically, deploy HTN planning to solve the following problem: Given context-free grammars G and G', is $L(G) \cap L(G') = \emptyset$?

Decision procedure:

- Construct an HTN planning problem $\mathcal{P}$ that has a solution if and only if the correct answer is yes.
- Translate the production rules to decomposition methods in a way that only words in both $L(G)$ and $L(G')$ can be produced.

HTN Planning is not in R

We reduce from the (undecidable) grammar intersection problem. (Chap.9, slide 31)

More specifically, deploy HTN planning to solve the following problem: Given context-free grammars G and G', is $L(G) \cap L(G') = \emptyset$?

Decision procedure:

- Construct an HTN planning problem $\mathcal{P}$ that has a solution if and only if the correct answer is yes.
- Translate the production rules to decomposition methods in a way that only words in both $L(G)$ and $L(G')$ can be produced.
- Any solution *tn* contains only one executable linearization. Each such linearization ω contains some ω' twice, with $\omega' \in L(G)$ and $\omega' \in L(G')$.

HTN Planning is not in R

We reduce from the (undecidable) grammar intersection problem. (Chap.9, slide 31)

More specifically, deploy HTN planning to solve the following problem: Given context-free grammars G and G', is $L(G) \cap L(G') = \emptyset$?

Decision procedure:

- Construct an HTN planning problem $\mathcal{P}$ that has a solution if and only if the correct answer is yes.
- Translate the production rules to decomposition methods in a way that only words in both $L(G)$ and $L(G')$ can be produced.
- Any solution *tn* contains only one executable linearization. Each such linearization ω contains some ω' twice, with $\omega' \in L(G)$ and $\omega' \in L(G')$.

We show the encoding using an example. (And skip the proof that it's a reduction.)
Proof by Erol et al., 1994.

HTN Planning is not in R, cont'd

$$\text{Let } G = (\overbrace{\Gamma = \{H, Q\}}^{\text{non-terminal symbols}}, \overbrace{\Sigma = \{a, b\}}^{\text{terminal symbols}}, \overbrace{R}^{\text{production rules}}, \overbrace{H}^{\text{start symbol}}) \text{ and}$$
$$G' = (\Gamma' = \{D, F\}, \Sigma' = \{a, b\}, R', D).$$

Production rules R: $H \mapsto aQb$ $Q \mapsto aQ \mid bQ \mid a \mid b$

Production rules R': $D \mapsto aFD \mid ab$ $F \mapsto a \mid b$

HTN Planning is not in R, cont'd

$$\text{Let } G = (\overbrace{\Gamma = \{H, Q\}}^{\text{non-terminal symbols}}, \overbrace{\Sigma = \{a, b\}}^{\text{terminal symbols}}, \overbrace{R}^{\text{production rules}}, \overbrace{H}^{\text{start symbol}}) \text{ and}$$
$$G' = (\Gamma' = \{D, F\}, \Sigma' = \{a, b\}, R', D).$$

Production rules R: $H \mapsto aQb$ $Q \mapsto aQ \mid bQ \mid a \mid b$

Production rules R': $D \mapsto aFD \mid ab$ $F \mapsto a \mid b$

$$\mathcal{P} = (V, \overbrace{\{H, Q, D, F\}}^{C}, \overbrace{\{a, b, a', b'\}}^{P}, \delta, M, \overbrace{\{v_{turn:G}\}}^{\text{initial state}}, tn_I, \overbrace{\{v_{turn:G}\}}^{\text{goal description}})$$

HTN Planning is not in R, cont'd

$$\text{Let } G = (\overbrace{\Gamma = \{H, Q\}}^{\text{non-terminal symbols}}, \overbrace{\Sigma = \{a, b\}}^{\text{terminal symbols}}, \overbrace{R}^{\text{production rules}}, \overbrace{H}^{\text{start symbol}}) \text{ and}$$
$$G' = (\Gamma' = \{D, F\}, \Sigma' = \{a, b\}, R', D).$$

Production rules R:	$H \mapsto aQb$	$Q \mapsto aQ \mid bQ \mid a \mid b$
Production rules R':	$D \mapsto aFD \mid ab$	$F \mapsto a \mid b$

$$\mathcal{P} = (V, \overbrace{\{H, Q, D, F\}}^{C}, \overbrace{\{a, b, a', b'\}}^{P}, \delta, M, \overbrace{\{v_{turn:G}\}}^{\text{initial state}}, tn_I, \overbrace{\{v_{turn:G}\}}^{\text{goal description}})$$
$$V = \{v_{turn:G}, v_{turn:G'}\} \cup \{v_a, v_b\}$$

HTN Planning is not in R, cont'd

$$\text{Let } G = (\overbrace{\Gamma = \{H, Q\}}^{\text{non-terminal symbols}}, \overbrace{\Sigma = \{a, b\}}^{\text{terminal symbols}}, \overbrace{R}^{\text{production rules}}, \overbrace{H}^{\text{start symbol}}) \text{ and}$$
$$G' = (\Gamma' = \{D, F\}, \Sigma' = \{a, b\}, R', D).$$

Production rules R: $H \mapsto aQb$ $\quad Q \mapsto aQ \mid bQ \mid a \mid b$

Production rules R': $D \mapsto aFD \mid ab$ $\quad F \mapsto a \mid b$

$$\mathcal{P} = (V, \overbrace{\{H, Q, D, F\}}^{C}, \overbrace{\{a, b, a', b'\}}^{P}, \delta, M, \overbrace{\{v_{turn:G}\}}^{\text{initial state}}, tn_I, \overbrace{\{v_{turn:G}\}}^{\text{goal description}})$$

$$V = \{v_{turn:G}, v_{turn:G'}\} \cup \{v_a, v_b\}$$

$$\begin{aligned}\delta = \{\, a &\mapsto (\{v_{turn:G}\}, \{v_{turn:G'}, v_a\}, \{v_{turn:G}\}),\\ b &\mapsto (\{v_{turn:G}\}, \{v_{turn:G'}, v_b\}, \{v_{turn:G}\}),\\ a' &\mapsto (\{v_{turn:G'}, v_a\}, \{v_{turn:G}\}, \{v_{turn:G'}, v_a\}),\\ b' &\mapsto (\{v_{turn:G'}, v_b\}, \{v_{turn:G}\}, \{v_{turn:G'}, v_b\})\}\end{aligned}$$

HTN Planning is not in R, cont'd

$$\text{Let } G = (\overbrace{\Gamma = \{H, Q\}}^{\text{non-terminal symbols}}, \overbrace{\Sigma = \{a, b\}}^{\text{terminal symbols}}, \overbrace{R}^{\text{production rules}}, \overbrace{H}^{\text{start symbol}}) \text{ and}$$

$$G' = (\Gamma' = \{D, F\}, \Sigma' = \{a, b\}, R', D).$$

Production rules R: $H \mapsto aQb$ $\quad Q \mapsto aQ \mid bQ \mid a \mid b$

Production rules R': $D \mapsto aFD \mid ab$ $\quad F \mapsto a \mid b$

$$\mathcal{P} = (V, \overbrace{\{H, Q, D, F\}}^{C}, \overbrace{\{a, b, a', b'\}}^{P}, \delta, M, \overbrace{\{v_{turn:G}\}}^{\text{initial state}}, tn_I, \overbrace{\{v_{turn:G}\}}^{\text{goal description}})$$

$$V = \{v_{turn:G}, v_{turn:G'}\} \cup \{v_a, v_b\}$$

$$\begin{aligned}\delta = \{\ a &\mapsto (\{v_{turn:G}\}, \{v_{turn:G'}, v_a\}, \{v_{turn:G}\}),\\ b &\mapsto (\{v_{turn:G}\}, \{v_{turn:G'}, v_b\}, \{v_{turn:G}\}),\\ a' &\mapsto (\{v_{turn:G'}, v_a\}, \{v_{turn:G}\}, \{v_{turn:G'}, v_a\}),\\ b' &\mapsto (\{v_{turn:G'}, v_b\}, \{v_{turn:G}\}, \{v_{turn:G'}, v_b\})\}\end{aligned}$$

$M = M(G) \cup M(G')$ (translated production rules of G and G')

$$tn_I = (\underbrace{\{t, t'\}}_{T}, \underbrace{\emptyset}_{\prec}, \underbrace{\{t \mapsto H, t' \mapsto D\}}_{\alpha})$$

Decidable Subcases

We only list some special cases that make HTN planning decidable.

- Acyclicity of Tasks. (Finitely many plans.)
- Total Order. (Among all the tasks.)
- Delete Relaxation.
- Regularity. (Only the last task in each method can be compound.)
- Tail-recursivity. (Generalization of Regularity.)
- Task insertion. (If we can also insert tasks anywhere.)
- Many more (possibly).

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 in HTN planning we had $\delta : P \to (2^V)^3$, so every action had a unique name. We thus assume this for classical planning as well. (So solutions are sequences of task names.)

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 in HTN planning we had $\delta : P \rightarrow (2^V)^3$, so every action had a unique name. We thus assume this for classical planning as well. (So solutions are sequences of task names.)
- So we can define:
  - $L(\mathcal{P}) = sol(\mathcal{P})$ if $\mathcal{P}$ is a classical problem.

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 in HTN planning we had $\delta : P \rightarrow (2^V)^3$, so every action had a unique name. We thus assume this for classical planning as well. (So solutions are sequences of task names.)
- So we can define:
  - $L(\mathcal{P}) = sol(\mathcal{P})$ if $\mathcal{P}$ is a classical problem.
  - $L(\mathcal{P}) = \{\bar{p} : tn \in sol(\mathcal{P})$ and $\bar{p}$ is an executable linearization of tn that makes g true.$\}$

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 in HTN planning we had $\delta : P \to (2^V)^3$, so every action had a unique name. We thus assume this for classical planning as well. (So solutions are sequences of task names.)
- So we can define:
  - $L(\mathcal{P}) = sol(\mathcal{P})$ if $\mathcal{P}$ is a classical problem.
  - $L(\mathcal{P}) = \{\bar{p} : tn \in sol(\mathcal{P})$ and $\bar{p}$ is an executable linearization of tn that makes g true.$\}$
- We can now compare planning problems (and their special cases) with regard to the Chomsky Hierarchy (i.e., the standard language classes, like regular and context-free languages).

HTN problems, defined differently

Let $\mathcal{P}$ be a hierarchical planning problem.

- Let $L_H(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ when disregarding executability. $\}$

HTN problems, defined differently

Let $\mathcal{P}$ be a hierarchical planning problem.

- Let $L_H(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ when disregarding executability. $\}$
- Let $L_C(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ the induced classical problem while disregarding the initial task network. $\}$

HTN problems, defined differently

Let $\mathcal{P}$ be a hierarchical planning problem.

- Let $L_H(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ when disregarding executability. $\}$
- Let $L_C(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ the induced classical problem while disregarding the initial task network. $\}$
- Thus, L_H just looks at the 'words' produced by the hierarchy, whereas L_C just looks at the executable words that produce the goal.
- Now, $L(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$.

HTN problems, defined differently

Let $\mathcal{P}$ be a hierarchical planning problem.

- Let $L_H(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ when disregarding executability. $\}$
- Let $L_C(\mathcal{P}) = \{\bar{p} : \bar{p}$ is the set of solutions to $\mathcal{P}$ the induced classical problem while disregarding the initial task network. $\}$
- Thus, L_H just looks at the 'words' produced by the hierarchy, whereas L_C just looks at the executable words that produce the goal.
- Now, $L(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$.

This obseration/proposition gives a new/simplified view on HTN planning:

HTN planning extends classical planning by adding a grammar to filter solutions.

Classes of Planning Problems

We can define the following Language classes:

- Let $\mathcal{HTN} = \{L(\mathcal{P}) : \mathcal{P}$ is an HTN planning problem.$\}$
- Let $\mathcal{CLASSIC} = \{L(\mathcal{P}) : \mathcal{P}$ is a classical planning problem.$\}$
- We can do the same for any restriction on planning problems:
  - $\mathcal{TOHTN} = \{L(\mathcal{P}) : \mathcal{P}$ is a total-order HTN planning problem.$\}$
  - and for any other restriction!

Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathit{REG}$

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathit{REG}$.

Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathit{REG}$

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathit{REG}$.
  - For this, notice that each planning problem encodes an (exponentially larger) DFA. Thus, given a classical planning problem, we can create its DFA. Each node is a state, each edge is an action.
  - We know that each DFA is regular, thus showing the claim.

Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathit{REG}$

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathit{REG}$.
  - For this, notice that each planning problem encodes an (exponentially larger) DFA. Thus, given a classical planning problem, we can create its DFA. Each node is a state, each edge is an action.
  - We know that each DFA is regular, thus showing the claim.
- We now show $\mathcal{CLASSIC} \subsetneq \mathit{REG}$.

Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathit{REG}$

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathit{REG}$.
  - For this, notice that each planning problem encodes an (exponentially larger) DFA. Thus, given a classical planning problem, we can create its DFA. Each node is a state, each edge is an action.
  - We know that each DFA is regular, thus showing the claim.
- We now show $\mathcal{CLASSIC} \subsetneq \mathit{REG}$.
  - We prove that $\{aa\} \in \mathit{REG}$ is not the language of any classical problem $\mathcal{P}$, $L(\mathcal{P}) \neq \{aa\}$ for all $\mathcal{P}$.

Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq REG$

Proof:

- We first show $\mathcal{CLASSIC} \subseteq REG$.
  - For this, notice that each planning problem encodes an (exponentially larger) DFA. Thus, given a classical planning problem, we can create its DFA. Each node is a state, each edge is an action.
  - We know that each DFA is regular, thus showing the claim.
- We now show $\mathcal{CLASSIC} \subsetneq REG$.
  - We prove that $\{aa\} \in REG$ is not the language of any classical problem $\mathcal{P}$, $L(\mathcal{P}) \neq \{aa\}$ for all $\mathcal{P}$.
  - Assume $aa \in L(\mathcal{P})$ for some classical problem $\mathcal{P}$. Then, we can show that $aaa \in L(\mathcal{P})$. (We skip the proof here, just write down the sets to show that $pre(a)$ must be contained in the state resulting from aa – because a was applicable in the state after executing a in s_I.)

Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathit{CF}$

Proof:

- We first show $\mathcal{TOHTN} \supseteq \mathit{CF}$.

Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathit{CF}$

Proof:

- We first show $\mathcal{TOHTN} \supseteq \mathit{CF}$.
  - Let G be a CF grammar. Use rules as methods, compound task names as terminal symbols, and primitive task names as terminal symbols.
  - For each terminal symbol define a no-operation (i.e., empty preconditions and effects). Set $g = \emptyset$.

Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = CF$

Proof:

- We first show $\mathcal{TOHTN} \supseteq CF$.
  - Let G be a CF grammar. Use rules as methods, compound task names as terminal symbols, and primitive task names as terminal symbols.
  - For each terminal symbol define a no-operation (i.e., empty preconditions and effects). Set $g = \emptyset$.
- Now we show $\mathcal{TOHTN} \subseteq CF$.

Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = CF$

Proof:

- We first show $\mathcal{TOHTN} \supseteq CF$.
  - Let G be a CF grammar. Use rules as methods, compound task names as terminal symbols, and primitive task names as terminal symbols.
  - For each terminal symbol define a no-operation (i.e., empty preconditions and effects). Set $g = \emptyset$.
- Now we show $\mathcal{TOHTN} \subseteq CF$.
  - We know that $L(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$ for all HTN problems $\mathcal{P}$.
  - We know that $L_H(\mathcal{P})$ is context-free and that $L_C(\mathcal{P})$ is regular.
  - It is known that the intersection of a context-free and regular language is context-free.

Conclusion

Some final remarks.

- Don't forget that:
  - We have almost 10 (internationally known) AI Planning experts at the ANU. (In case you want to do a PhD or research project.)
  - Many (most?) in the Foundations group (might) have theory-heavy research projects to offer – assuming they have any project at all! (They/we might not have any at the moment, or no capacities left.)

Conclusion

Some final remarks.

- Don't forget that:
  - We have almost 10 (internationally known) AI Planning experts at the ANU. (In case you want to do a PhD or research project.)
  - Many (most?) in the Foundations group (might) have theory-heavy research projects to offer – assuming they have any project at all! (They/we might not have any at the moment, or no capacities left.)
- Please take part in SELT. (No matter whether you liked it or not.)

Conclusion

Some final remarks.

- Don't forget that:
  - We have almost 10 (internationally known) AI Planning experts at the ANU. (In case you want to do a PhD or research project.)
  - Many (most?) in the Foundations group (might) have theory-heavy research projects to offer – assuming they have any project at all! (They/we might not have any at the moment, or no capacities left.)
- Please take part in SELT. (No matter whether you liked it or not.)
- I hope you enjoyed the course!
- Good luck in the exam! (And your other exams.)

Thank you for taking this course!