COMP3630 / COMP6363

# *week 12:* **Examples from Hierarchical 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

Hierarchical Task Network (HTN) Planning:

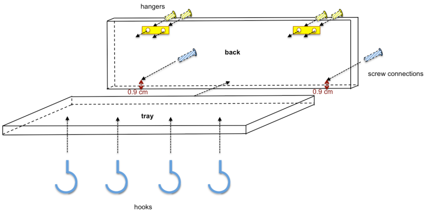
- Examples
- Formal Problem Definition(s)
- Expressivity Analysis
- Complexity Analysis

# Examples

# 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? (Like a production rule!)
- Why defining/solving a hierarchical 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 exclude (more) undesired plans! (Exactly how formal grammars do!)
  - 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

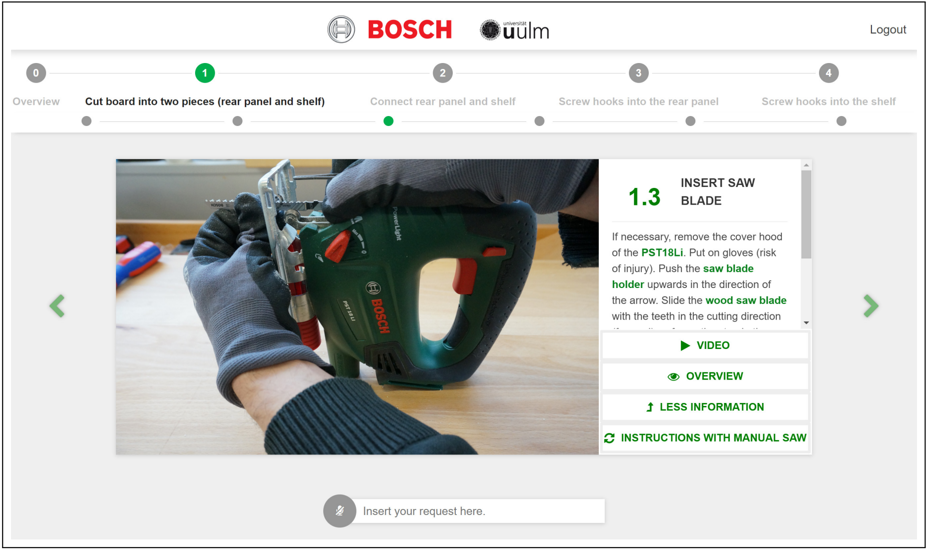



The material:

- 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.

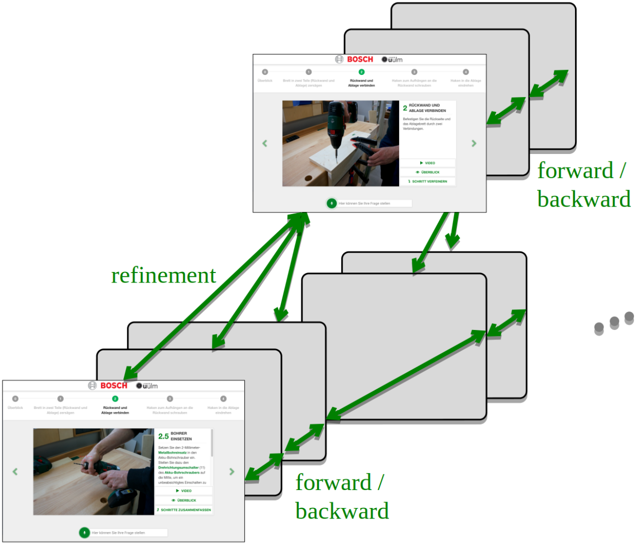
# Example: Do-It-Yourself (DIY) Assistant, User Interface



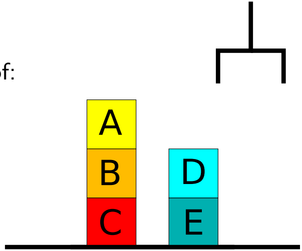
# Example: Do-It-Yourself (DIY) Assistant, Task Hierarchy

Abstract Level

Detailed Level



# Recap: Blocksworld via Classical Planning

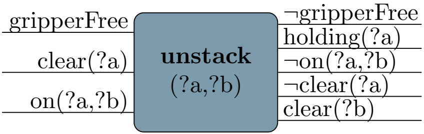



We consider classical planning problems, which consist of:

- All existing state variables $V$.
- An initial state $s_I \in 2^V$.
- A set of available actions $A$.
- A goal description $g \subseteq V$.

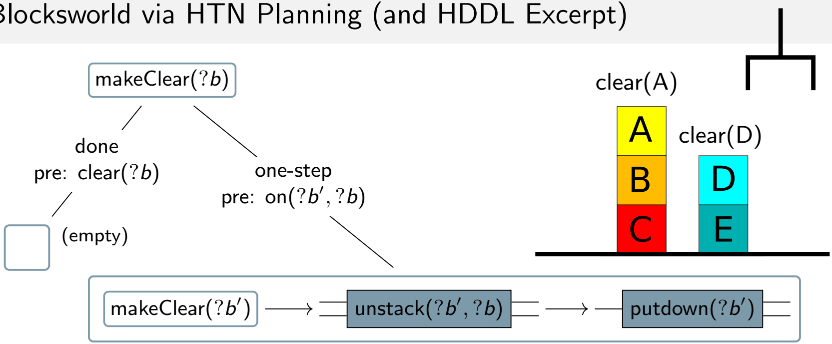
$\rightarrow$ Find an action sequence (i.e., a plan) that transforms $s_I$ into a state $s \supseteq g$.

For example, one of the available actions is:




- For an action to be executable, all preconditions must hold.
- Actions change states by adding or deleting their effects.

# Blocksworld via HTN Planning (and HDDL Excerpt)



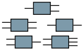

```
(:task makeClear :parameters (?b - block))

(:method one-step
 :parameters (?b1 ?b2 - block)
 :task (makeClear ?b1)
 :precondition (and (on ?b2 ?b1))
 :ordered-tasks (and (makeClear ?b2)
                     (unstack ?b2 ?b1)
                     (putdown ?b2)))
```

~~makeClear(A)~~
unstack(A,B)
putdown(A)
~~makeClear(B)~~
unstack(B,C)
putdown(B)
~~makeClear(C)~~

# Formalism

# Introduction to HTN Planning

primitive
tasks

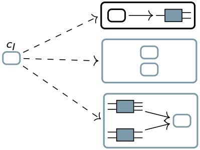


compound
tasks

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

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

# Introduction to HTN Planning



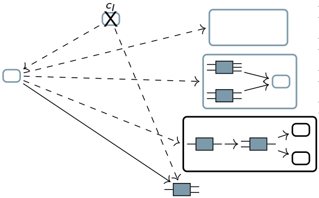

$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- $V$ a set of facts
- $P$ 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

We must find a task network *tn*, such that:

- it is a refinement of $c_I$,
- only contains primitive tasks, and

# Introduction to HTN Planning



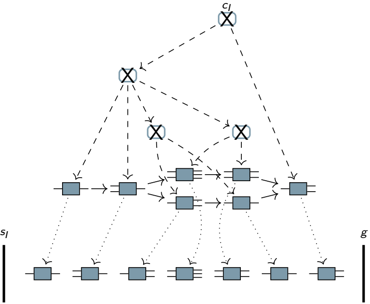

$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I, g)$

- $V$ a set of facts
- $P$ 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

We must find a task network *tn*, such that:

- it is a refinement of $c_I$,
- only contains primitive tasks, and

# Introduction to HTN Planning




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

- $V$ a set of facts
- $P$ 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
- $s_I \in 2^V$ the initial state
- $g \subseteq V$ the (optional) goal description

We must find a task network *tn*, such that:

- it is a refinement of $c_I$,
- only contains primitive tasks, and
- has an executable linearization that makes the goals in $g$ true.

# Decomposition, formally

- Task networks are tuples $(T, \prec, \alpha)$ consisting of a set of task IDs (labels) $T$ and a strict partial order $\prec \subseteq T \times T$. I.e., $\prec$ is irreflexive and transitive (and hence asymmetric). $\alpha : T \to P \cup C$ maps task IDs to the actual tasks (i.e., their names).
- 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 $\alpha(t) = c$ is 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:

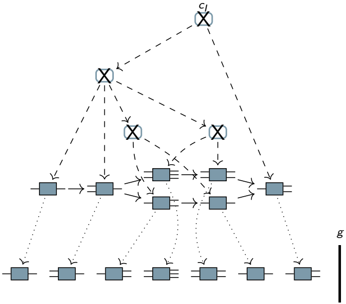
$$\prec_X := \{(t', t'') \mid (t', t) \in \prec, t'' \in T_m\} \cup \{(t'', t') \mid (t, t') \in \prec, t'' \in T_m\}$$

  where $(X_1, \ldots, X_n)|_Y$ restricts the sets $X_i$ to elements in $Y$
- Note that this definition becomes trivial if all methods are totally ordered.
  It then perfectly coincides with the definition of using a production rule.

# HTN Planning: Solution Criteria in more Detail

An action sequence $\bar{p} \in P^*$ is a solution if and only if:

- There is a sequence of decomposition methods $\overline{m}$ that transforms $c_I$ into some *tn*,
- *tn* contains only primitive tasks (those in $\bar{p}$), and
- *tn* admits $\bar{p}$ as linearization, is executable, leads to a goal state $s \supseteq g$.




An action sequence is called executable if every action is executable in its state:

- Let $s \in 2^V$ be a state, $p \in P$, and $\delta(p)$ an action with $\delta(p) = (pre, add, del)$ and $pre, add, del \subseteq V$.
- Then, $p$ is executable in $s$ iff $pre \subseteq s$.
- Then, $p$ executed in $s$ leads to new state $s' = (s \setminus del) \cup add$.

# Alternative Definition of HTN Planning

- Actions were defined by their name: $\delta : P \rightarrow 2^V \times 2^V \times 2^V$.
  Thus, solutions are (the same as) sequences of task names.
- Thus, any solution set $sol(\mathcal{P})$ is a language. Let:
  - $L_H(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}')$, where $\mathcal{P}'$ ignores all facts $\}$
  - $L_C(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}')$, where $\mathcal{P}'$ is the induced classical problem $\}$
- This means:
  - $L_H$ just looks at the words produced by the hierarchy, (ignores executability)
  - $L_C$ just looks at the executable words that produce the goal. (ignores hierarchy)

$\rightarrow$ Thus: $sol(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$.

This observation gives a new/simplified view on HTN planning:

**HTN planning = classical planning + grammar to filter solutions**

**!!** Maybe the most important interpretation of knowledge about HTN Planning **!!**

# Expressivity Analysis

# Recap: Chomsky Hierarchy, extended

We can define the following Language classes:

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

We know that $\mathcal{CLASSIC} \subsetneq \mathcal{R}eg \subsetneq \mathcal{CF} \subsetneq \mathcal{CSL} \subsetneq \mathcal{A}ll$.

Now, we also have:

- $\mathcal{HTN} = \{L(\mathcal{P}) \mid \mathcal{P} \text{ is an HTN planning problem.}\}$
- $\mathcal{TOHTN} = \{L(\mathcal{P}) \mid \mathcal{P} \text{ is a total-order HTN planning problem.}\}$

# Expressivity of HTN Problems

**Theorem w12.1 (Höller et al. (2014), Thm. 6)**

$\mathcal{TOHTN} = \mathcal{CF}$

**Proof.**

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

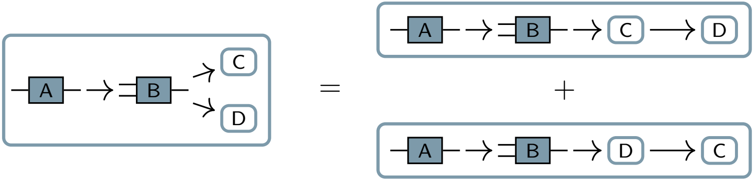
- Let $G$ be a CF grammar. Use rules as methods, compound task names as terminal symbols, and primitive task names as terminal symbols.
- For each terminal symbol define a no-operation. Set $g = \emptyset$.
- With this, every CF grammar is a TO HTN problem!

Now we show $\mathcal{TOHTN} \subseteq \mathcal{CF}$.

- 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 (we established that above)
  - $L_C(\mathcal{P})$ is regular (we established that last week).
- It is known that the intersection of a context-free and regular language is context-free. (We didn't prove that yet, but the idea is a product automaton of PDA and DFA.) □

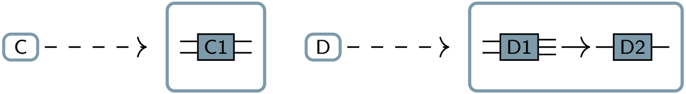
# On the (Non?)equivalence of PO and TO HTN models

Each partially ordered task network is just a compact representation of its linearizations:

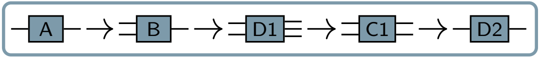



, right?

Let:




Can we create the following task network?




**No!** Not anymore...

# More Expressivity Results

Recall: (the last two are new)

- $\mathcal{HTN} = \{\mathbf{L}(\mathcal{P}) \mid \mathcal{P}$ is an HTN planning problem.$\}$
- $\mathcal{TOHTN} = \{\mathbf{L}(\mathcal{P}) \mid \mathcal{P}$ is a total-order HTN planning problem.$\}$
- $\mathcal{ACYC}\text{-}\mathcal{HTN} = \{\mathbf{L}(\mathcal{P}) \mid \mathcal{P}$ is an acyclic HTN planning problem.$\}$
- $\mathcal{NOOP}\text{-}\mathcal{HTN} = \{\mathbf{L}_H(\mathcal{P}) \mid \mathcal{P}$ is an HTN planning problem.$\}$

An excerpt of other expressivity results (not shown here):

- $\mathcal{ACYC}\text{-}\mathcal{HTN} \subsetneq \mathcal{R}eg$ Because their languages are finite!
- $\mathcal{CF} = \mathcal{TOHTN} \subsetneq \mathcal{NOOP}\text{-}\mathcal{HTN} \subsetneq \mathcal{HTN} \subsetneq \mathcal{CSL}$

Note: There are more special cases that were considered in literature.

# Complexities

# Complexity of HTN Planning (General Case)

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

**Theorem w12.1 (Erol et al. (1996), Thm. 1)**

*$PLANEX_{HTN}$ is undecidable*

**Proof.**

We reduce from the (undecidable) CF grammar intersection problem.

Given the CF grammars $G$ and $G'$, construct HTN problem to answer $L(G) \cap L(G') \overset{?}{\neq} \emptyset$ using the following decision procedure:

- Construct an HTN planning problem $\mathcal{P}$ that has a solution if and only if the correct answer to the grammar/language intersection problem 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.
- Our desired primitive task network *tn* contains only one executable linearization $\omega = \omega_1, \omega_2, \ldots, \omega_{2n-1}, \omega_{2n}$:
  - $\omega^1 = \omega_1, \omega_3, \ldots, \omega_{2n-1}$, $|\omega^1| = n$, and $\omega^1 \in L(G)$
  - $\omega^2 = \omega_2, \omega_4, \ldots, \omega_{2n}$, $|\omega^2| = |\omega^1|$, and $\omega^2 \in L(G')$
- Encoding given in next slide!

□

# Reduction, Shown by Example

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

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

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

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

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

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

# Complexity of HTN Planning

$\text{PLANEX}_{TOHTN} = \{\langle\mathcal{P}\rangle \mid \mathcal{P}$ is a solvable TO HTN planning problem.$\}$

**Theorem w12.2 (Erol et al. (1996), Thm. 4, Alford et al. (2016), Thm. 5.1)**

*$\text{PLANEX}_{TOHTN}$ is* **EXPTIME***-complete.*

**Proof.**

Membership:

- Dynamic programming procedure that does a bottom-up analysis.
- Check all pairs of (state,task,state) for executability/decomposability.
- Details covered in the tutorial!

Hardness:

- Reduction from a polyspace-bounded ATM.
  (We know **AP** = **PSPACE**, but it also holds: **APSPACE** = **EXPTIME**)
- Proof skipped.

□

# Conclusion

# Conclusion on HTN planning

- HTN planning is classical planning plus a grammar to filter solutions.
- HTN planning is both more expressive and more complex than classical planning.
- HTN planning is undecidable in general, but restrictions on the hierarchy or ordering make it simpler.
- Recall one interesting result from week 10: Delete-relaxed HTN planning is **NP**-complete, although even the shortest solution may be exponential.

# Conclusion of the Course

- The first few weeks we were investigating the “expressivity” of various kinds of machine models. Noteworthy are:
  - the languages from the Chomsky Hierarchy (and the Pumping lemmas)
  - the classes $\mathcal{R}$, $\mathcal{RE}$, non-$\mathcal{RE}$
- The last weeks we were investigating the “computational complexity” of various languages. Noteworthy are:
  - The investigated complexity classes and their relationship. (Which are known?)
  - The difference between **NP** and **co-NP**.
  - The difference of membership, hardness, completeness – and reductions.

# Final Remarks

- Don't forget that:
  - We have about 8 (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.
- **Please take part in SELT.** (No matter whether you liked it or not.)
- Keep monitoring the forum! Read all questions/answers !!
- I hope you enjoyed the course!
- Good luck in the exam! (And your other exams.)

**Thank you** for taking this course!

A **special Thank you** to those who attended in person! :)