Hierarchical Planning:
Introduction and Modeling Support

Pascal Bercher

School of Computing
College of Engineering, Computer Science and Cybernetics
The Australian National University (ANU)

3 July 2023

Introduction

About the Speaker

Who am I? Why am I here?

- Did my PhD in Ulm, at the Institute of Artificial Intelligence
  - Under the supervision of Prof. Biundo (now retired)
  - In the context of Hierarchical Planning, applied to Companion Technology (cognitive systems), specifically to provide planning-based user assistance
- Worked closely with Prof. Glimm (13 publications! – and counting)
- Bio:
  - 2002–2009: Studied Computer Science in Freiburg im Breisgau
  - 2009–2017: Doctoral Studies in Ulm
  - 2017–2019: Post-Doc – still in Ulm
  - 2019–2021: Lecturer (Assistant/Junior Professor) at the ANU
  - 2022–. . . : Senior Lecturer (Associate/W2 Professor) at the ANU

Planning in a Nutshell: Main Ingredients and Purpose of AI Planning

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?

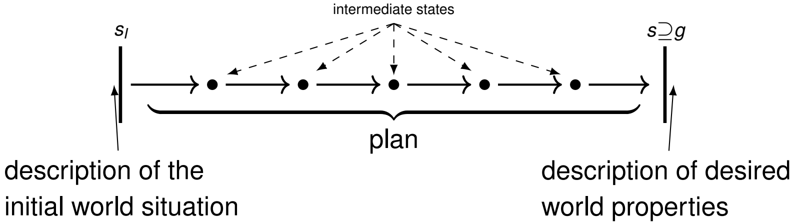
Planning in a Nutshell: Main Ingredients and Purpose of AI Planning

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.



Planning in a Nutshell: Example: Home Theater Assembly Assistant

Sink devices:

- Television (requires video)
- Amplifier (requires audio)

Source devices:

- Blu-ray player
- Satellite receiver
  (both produce audio & video)

Planning in a Nutshell: Definitions, Examples

- Planning problems are usually defined by a description language (e.g., PDDL / HDDL) based on a first-order predicate logic.
  - Predicates, like $\mathit{HasPort}(?\mathit{device}, ?\mathit{port})$, express relationships between variables representing objects.
  - Constants, like *AMPLIFIER* and *CABLE_HDMI*, represent objects.
- States are sets of (ground) propositions, e.g.,

$$s \supseteq \{\mathit{HasPort}(\mathit{AMPLIFIER}, \mathit{HDMI}),\\ \mathit{HasPort}(\mathit{AMPLIFIER}, \mathit{CINCH}),\\ \mathit{HasPort}(\mathit{CABLE\_HDMI}, \mathit{HDMI}),\\ \mathit{IsConnected}(\mathit{AMPLIFIER}, \mathit{CABLE\_HDMI}, \mathit{HDMI})\}$$

(devices are connected to each other)

Planning in a Nutshell: Definitions, Examples

- Actions are defined by their preconditions and effects, e.g.,

plugIn(*?cable, ?device, ?port* ***)***

precondition: $HasPort(?device, ?port) \wedge$
$HasPort(?cable, ?port) \wedge$
$\nexists ?cable' : IsConnected(?device, ?cable', ?port)$
$\nexists ?device' : IsConnected(?device', ?cable, ?port)$

effect: $IsConnected(?device, ?cable, ?port)$

(Signal flow not shown for the sake of simplicity)

Planning in a Nutshell: Planning Problem Definition in the Home Theater Domain

Initial state:

- HasPort(. . . , . . .) // which device has which ports?
- IsConnected(. . . , . . . , . . .) // how are the connections initially?
- HasSignal(. . . , . . . , . . .) // which device has which signals?

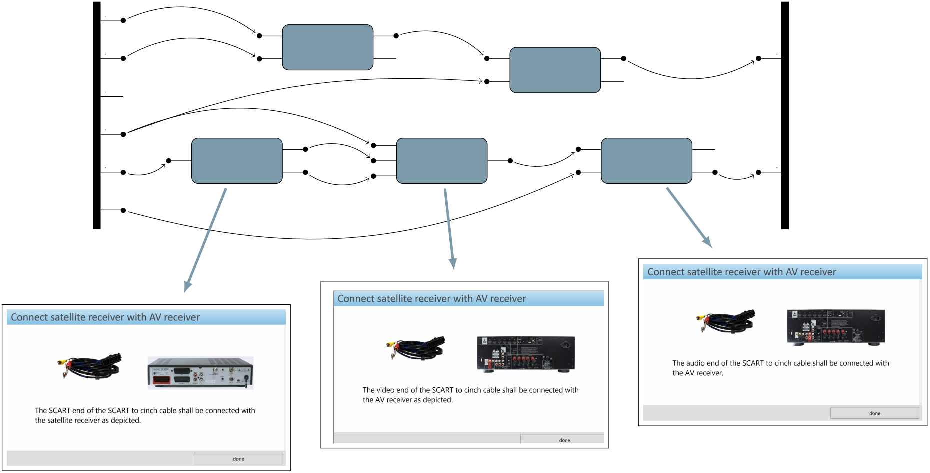
Action portfolio:

- ***plugIn(****?cable*, ?*device*, ?*port****)*** // plugging in a cable
- ***plugOut(****?cable*, ?*device*, ?*port****)*** // in case plugging out is allowed

Goal description:

- HasSignal(. . . , . . . , . . .) // e.g., HasSignal(TV, VIDEO, BR) denoting
- . . . that the TV has the video signal of the blu-ray player

Planning in a Nutshell: The Assistant: Communicating Solution Plans



Examples of Planning Problems: Games (e.g., Rush Hour)

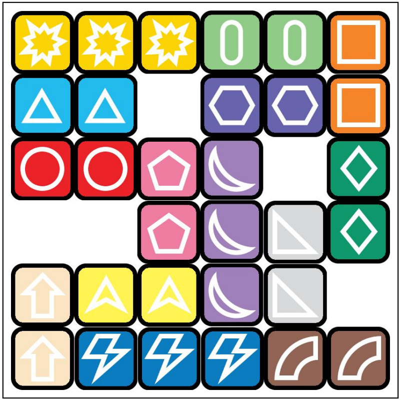
Photo made out of Hanna Neumann of the ANU (between HN, Birch, and CSIT; December 2020).

Examples of Planning Problems: Games (e.g., Rush Hour)

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

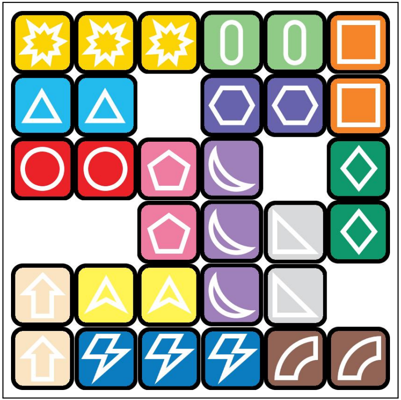
Examples of Planning Problems: Games (e.g., Rush Hour)

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



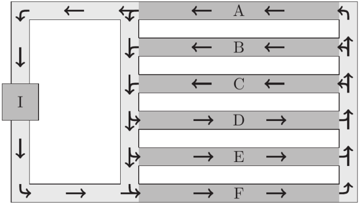
Examples of Planning Problems: Games (e.g., Rush Hour)

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



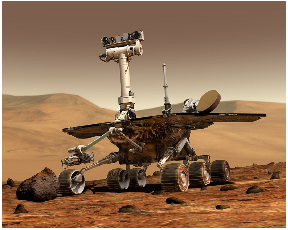
Modeling this, including the automated video creation was a 6 pt. project in S1 2023 (= effort of one course).

Examples of Planning Problems: Automated Factories (e.g., a Greenhouse)



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

Examples of Planning Problems: Robotics (e.g., Mars Rovers)



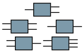
- MAPGEN (Mixed Initiative Activity Planning Generator) is a ground-based decision support system for Mars Exploration Rover mission operations and science teams that begins to give content to the notion of autonomous planetary exploration.
- The paradigm is to enable the person using the software to critique a plan that the system automatically produces and ensure that resulting plans are viable within mission and flight rules.

from *https://www.nasa.gov/*

Hierarchical Planning

Formalism: HTN Planning: Problem Definition & Solution Criteria

primitive tasks



compound tasks

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

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

Formalism: HTN Planning: Problem Definition & Solution Criteria

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

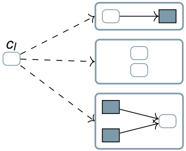
- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task

A solution task network *tn* must:

- be a refinement of c_I,

Formalism: HTN Planning: Problem Definition & Solution Criteria

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



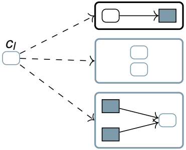
- F a set of facts
- P a set of primitive task names
- $\delta : P \rightarrow (2^F)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

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

Formalism: HTN Planning: Problem Definition & Solution Criteria

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

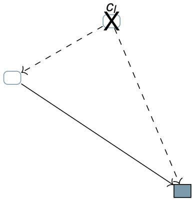


- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ the task name mapping
- C a set of compound task names
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods

A solution task network *tn* must:

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

Formalism: HTN Planning: Problem Definition & Solution Criteria



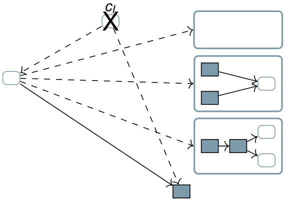
$$\mathcal{P} = (F, P, \delta, C, M, s_I, c_I, g)$$

- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ 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

Formalism: HTN Planning: Problem Definition & Solution Criteria



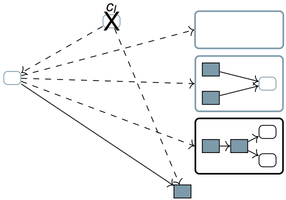
$$\mathcal{P} = (F, P, \delta, C, M, s_I, c_I, g)$$

- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ 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

Formalism: HTN Planning: Problem Definition & Solution Criteria



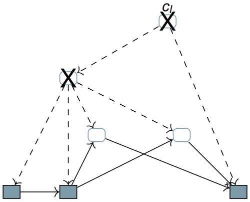
$$\mathcal{P} = (F, P, \delta, C, M, s_I, c_I, g)$$

- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ 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

Formalism: HTN Planning: Problem Definition & Solution Criteria



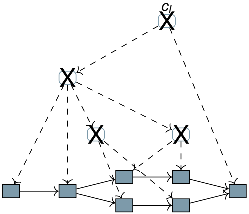
$$\mathcal{P} = (F, P, \delta, C, M, s_I, c_I, g)$$

- F a set of facts
- P a set of primitive task names
- $\delta : P \rightarrow (2^F)^3$ 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

Formalism: HTN Planning: Problem Definition & Solution Criteria



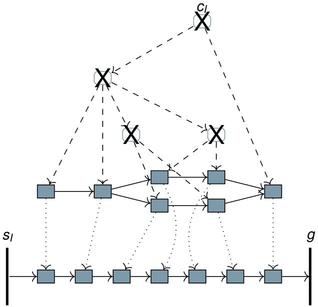
$$\mathcal{P} = (F, P, \delta, C, M, s_I, c_I, g)$$

- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ 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

Formalism: HTN Planning: Problem Definition & Solution Criteria



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

- F a set of facts
- P a set of primitive task names
- $\delta : P \to (2^F)^3$ 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
- $s_I \in 2^F$ the initial state
- $g \subseteq F$ 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.

Formalism: HTN Planning: Solution Criteria in more Detail

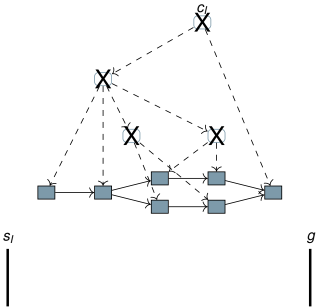
A task network *tn* is a solution if and only if:

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

Formalism: HTN Planning: Solution Criteria in more Detail

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

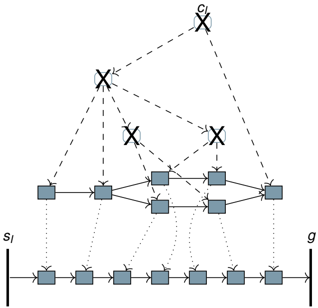
- There is a sequence of decomposition methods $\overline{m}$ that transforms c_I into *tn*,
- *tn* contains only primitive tasks, and



Formalism: HTN Planning: Solution Criteria in more Detail

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

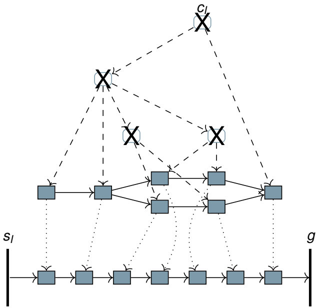
- There is a sequence of decomposition methods $\overline{m}$ that transforms c_I into *tn*,
- *tn* contains only primitive tasks, and
- the (still partially ordered) task network *tn* admits an executable linearization $\overline{t}$ of its tasks leading to some state $s \supseteq g$.



Formalism: HTN Planning: Solution Criteria in more Detail

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 $\overline{t}$ of its tasks leading to some state $s \supseteq g$.



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

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

Motivation: Comparison to Classical Planning

- HTN planning differs from classical planning in:
  - We don't plan to achieve some state features at the end of the plan, but to find some plan that's a refinement of some initial task(s).
  - We also can't insert actions anywhere (as long as they are executable), but we need to adhere replacement rules (called methods).

Motivation: Comparison to Classical Planning

- HTN planning differs from classical planning in:
  - We don't plan to achieve some state features at the end of the plan, but to find some plan that's a refinement of some initial task(s).
  - We also can't insert actions anywhere (as long as they are executable), but we need to adhere replacement rules (called methods).
- Why defining/solving a *hierarchical* problem?
  - In many real-world applications, knowledge is given in form of control rules: we know the steps required to perform some task.

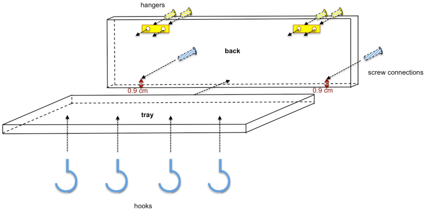
Motivation: Comparison to Classical Planning

- HTN planning differs from classical planning in:
  - We don't plan to achieve some state features at the end of the plan, but to find some plan that's a refinement of some initial task(s).
  - We also can't insert actions anywhere (as long as they are executable), but we need to adhere replacement rules (called methods).
- Why defining/solving a *hierarchical* problem?
  - 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" (methods) need to be adhered. We can *exclude* (more) undesired plans! (Leads to higher expressivity, see later!)

Motivation: Comparison to Classical Planning

- HTN planning differs from classical planning in:
  - We don't plan to achieve some state features at the end of the plan, but to find some plan that's a refinement of some initial task(s).
  - We also can't insert actions anywhere (as long as they are executable), but we need to adhere replacement rules (called methods).
- Why defining/solving a *hierarchical* problem?
  - 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" (methods) need to be adhered. We can *exclude* (more) undesired plans! (Leads to higher expressivity, see later!)
  - Plans can be presented more compactly/abstractly to users.

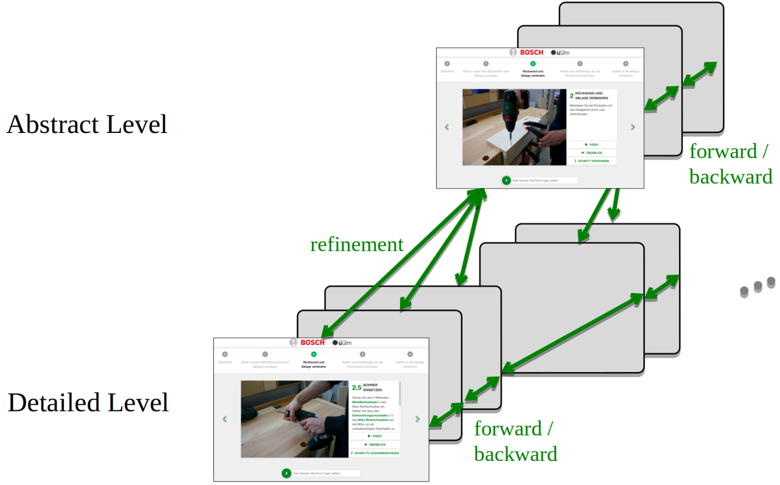
Motivation: Example: Do-It-Yourself (DIY) Assistant



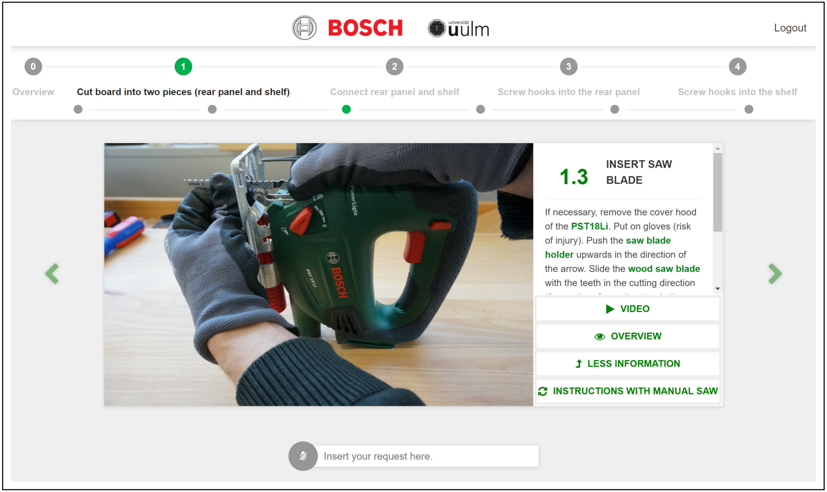
The material:

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

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



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



Formal Grammars and Expressivity

Formal Grammars and Languages: Definition

Recap from Theoretical Computer Science:

A *context-free grammar* G is a tuple $\langle N, \Sigma, S, R \rangle$ where

- N is a finite set of *non-terminal symbols*,
- Σ, disjoint from N, is a finite set of *terminal symbols* (Σ is also called *alphabet*),
- $S \in N$ is the *start symbol*,
- $R \subseteq N \times (N \cup \Sigma)^*$ is a finite set of *production rules*.

Formal Grammars and Languages: Definition

Recap from Theoretical Computer Science:

A *context-free grammar* G is a tuple $\langle N, \Sigma, S, R \rangle$ where

- N is a finite set of *non-terminal symbols*,
- Σ, disjoint from N, is a finite set of *terminal symbols* (Σ is also called *alphabet*),
- $S \in N$ is the *start symbol*,
- $R \subseteq N \times (N \cup \Sigma)^*$ is a finite set of *production rules*.

Languages:

- A language L is any (possibly infinite) set of words (sequences of symbols). E.g., the sets $\emptyset$, $\{a, b, \ldots, z\}$, and $\mathbb{N}$ are languages.
- The *language of a grammar*, $L(G) \subseteq \Sigma^*$, is the set of terminal words obtainable by refining S by only using production rules.

Formal Grammars and Languages: Example

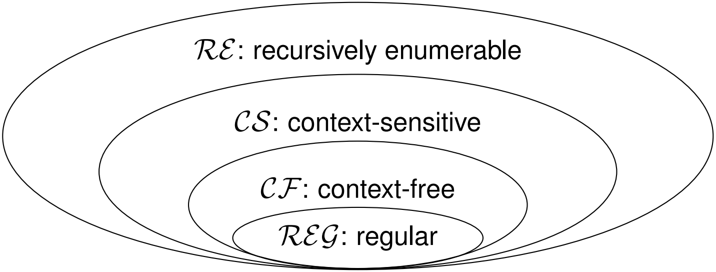
- Let $G = \langle \{a, b\}, \{S, A, B\}, S, \{S \rightarrow aB, B \rightarrow Ab, A \rightarrow S, A \rightarrow \epsilon\} \rangle$, so we have:
  - Terminal symbols: $\{a, b\}$
  - Non-terminals: $\{S, A, B\}$
  - Start symbol: S
  - Production rules:
    - $S \rightarrow aB$
    - $B \rightarrow Ab$
    - $A \rightarrow S \mid \epsilon$
- So, the language is $L(G) =$

Formal Grammars and Languages: Example

- Let $G = \langle \{a, b\}, \{S, A, B\}, S, \{S \rightarrow aB, B \rightarrow Ab, A \rightarrow S, A \rightarrow \epsilon\} \rangle$, so we have:
  - Terminal symbols: $\{a, b\}$
  - Non-terminals: $\{S, A, B\}$
  - Start symbol: S
  - Production rules:
    - $S \rightarrow aB$
    - $B \rightarrow Ab$
    - $A \rightarrow S \mid \epsilon$
- Some example derivations:
  - $S \longrightarrow aB \longrightarrow aAb \longrightarrow ab$
  - $S \longrightarrow aB \longrightarrow aAb \longrightarrow aSb \longrightarrow \cdots \longrightarrow aabb$
- So, the language is $L(G) = \{a^n b^n \mid n \geq 1\}$

Formal Grammars and Languages: Chomsky Hierarchy

The Chomsky Hierarchy defines a hierarchy of expressiveness.



For example, we know that:

- The context-free languages are exactly those for which there is a context-free grammar. $\rightarrow$ Thus $\{a^n b^n \mid n \geq 1\}$ is context-free.
- Regular languages are exactly those for which there exists a finite automaton. $\rightarrow$ Thus $\{a^n b^n \mid n \geq 1\}$ is *not* regular.

Expressiveness of Planning Formalisms: Language of a Planning Problem

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

- Actions were defined by their name: $\delta : P \rightarrow 2^F \times 2^F \times 2^F$.
  So solutions are (the same as) sequences of task names.

Expressiveness of Planning Formalisms: Language of a Planning Problem

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

- Actions were defined by their name: $\delta : P \rightarrow 2^F \times 2^F \times 2^F$. So solutions are (the same as) sequences of task names.
- Let $\mathcal{P}$ be a (classical) planning problem and $sol(\mathcal{P})$ its set of solutions. If we interpret any action as a symbol (or just use its name instead), then $sol(\mathcal{P})$ is a language!

Expressiveness of Planning Formalisms: Language of a Planning Problem

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

- Actions were defined by their name: $\delta : P \to 2^F \times 2^F \times 2^F$. So solutions are (the same as) sequences of task names.
- Let $\mathcal{P}$ be a (classical) planning problem and $sol(\mathcal{P})$ its set of solutions. If we interpret any action as a symbol (or just use its name instead), then $sol(\mathcal{P})$ is a language!
- So we can define:
  - $L_{\mathcal{CLASSIC}}(\mathcal{P}) = sol(\mathcal{P})$
  - $L_{\mathcal{HTN}}(\mathcal{P}) = \{\bar{p} \mid tn \in sol(\mathcal{P})$ and $\bar{p}$ is an executable linearization of tn that makes g true.$\}$

If $\mathcal{P}$ is a classical or HTN planning problem, respectively.

Expressiveness of Planning Formalisms: Language of a Planning Problem

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

- Actions were defined by their name: $\delta : P \rightarrow 2^F \times 2^F \times 2^F$.
  So solutions are (the same as) sequences of task names.
- Let $\mathcal{P}$ be a (classical) planning problem and $sol(\mathcal{P})$ its set of solutions. If we interpret any action as a symbol (or just use its name instead), then $sol(\mathcal{P})$ is a language!
- So we can define:
  - $L_{\mathcal{CLASSIC}}(\mathcal{P}) = sol(\mathcal{P})$
  - $L_{\mathcal{HTN}}(\mathcal{P}) = \{\bar{p} \mid tn \in sol(\mathcal{P})$ and $\bar{p}$ is an executable linearization of tn that makes g true.$\}$

  If $\mathcal{P}$ is a classical or HTN planning problem, respectively.
- We can ask for possible plan structures depending on problem classes! E.g., can we define a problem where every solution has the form $\langle action_1^n, action_2, action_2, action_3^n \rangle$ (for any n)?

Expressiveness of Planning Formalisms: Let's redefine HTN Problems!

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

Expressiveness of Planning Formalisms: Let's redefine HTN Problems!

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

- Let $L_H(\mathcal{P}) = \{\bar{p} \mid tn \in sol(\mathcal{P})$ and $\bar{p}$ is an "executable" linearization of tn – when ignoring executability, i.e., preconditions.$\}$
  - $\rightarrow$ Now, we only have a grammar left!

Expressiveness of Planning Formalisms: Let's redefine HTN Problems!

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

- Let $L_H(\mathcal{P}) = \{\bar{p} \mid tn \in sol(\mathcal{P})$ and $\bar{p}$ is an "executable" linearization of tn – when ignoring executability, i.e., preconditions.$\}$
  - $\rightarrow$ Now, we only have a grammar left!
- Let $L_C(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}')$ with $\mathcal{P}'$ the classical problem induced by $\mathcal{P}'$ (by disregarding tn_I and allowing action insertion).$\}$
  - $\rightarrow$ Now, we only have a classical problem left! No restriction by the hierarchy.

Expressiveness of Planning Formalisms: Let's redefine HTN Problems!

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

- Let $L_H(\mathcal{P}) = \{\bar{p} \mid tn \in sol(\mathcal{P})$ and $\bar{p}$ is an "executable" linearization of tn – when ignoring executability, i.e., preconditions.$\}$

  → Now, we only have a grammar left!

- Let $L_C(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}')$ with $\mathcal{P}'$ the classical problem induced by $\mathcal{P}'$ (by disregarding tn_I and allowing action insertion).$\}$

  → Now, we only have a classical problem left! No restriction by the hierarchy.

- Thus:
  - L_H just looks at the 'words' produced by the hierarchy,
  - L_C just looks at the executable words that produce the goal.

  → $L_{\mathcal{HTN}}(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$.

Expressiveness of Planning Formalisms: Let's redefine HTN Problems!

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

- Let $L_H(\mathcal{P}) = \{\bar{p} \mid tn \in sol(\mathcal{P})$ and $\bar{p}$ is an "executable" linearization of tn – when ignoring executability, i.e., preconditions.$\}$
  - → Now, we only have a grammar left!
- Let $L_C(\mathcal{P}) = \{\bar{p} \mid \bar{p} \in sol(\mathcal{P}')$ with $\mathcal{P}'$ the classical problem induced by $\mathcal{P}'$ (by disregarding tn_I and allowing action insertion).$\}$
  - → Now, we only have a classical problem left! No restriction by the hierarchy.
- Thus:
  - L_H just looks at the 'words' produced by the hierarchy,
  - L_C just looks at the executable words that produce the goal.
  - → $L_{\mathcal{HTN}}(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$.

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

HTN planning = classical planning + grammar to filter solutions

Expressiveness of Planning Formalisms: Classes of Planning Problems

We can define the following Language classes:

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

Expressiveness of Planning Formalisms: Classes of Planning Problems

We can define the following Language classes:

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

What's the idea/purpose?

- Recall why we defined $\mathcal{REG}, \mathcal{CF}$, etc.: To check which languages can be expressed by the respective structures, e.g., Automata for $\mathcal{REG}$ or context-free grammars for $\mathcal{CF}$.
- Now we can also classify $\mathcal{HTN}$, $\mathcal{TOHTN}$, etc. within the Chomsky Hierarchy. E.g., can $\mathcal{CLASSIC}$ express solutions of the form $\langle action_1^n, action_2, action_2, action_3^n \rangle$ (for any n)?

Expressivity of Classical Problems: Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$ – by Höller, Behnke, Bercher, Biundo, ’14

Expressivity of Classical Problems: Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$ – by Höller, Behnke, Bercher, Biundo, ’14

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathcal{REG}$
  - 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: Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$ – by Höller, Behnke, Bercher, Biundo, '14

Proof:

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

Expressivity of Classical Problems: Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$ – by Höller, Behnke, Bercher, Biundo, '14

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathcal{REG}$
  - 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 $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$.
  - We prove that $\{aa\} \in \mathcal{REG}$ is not the language of any classical problem $\mathcal{P}$, $L(\mathcal{P}) \neq \{aa\}$ for all $\mathcal{P}$.

Expressivity of Classical Problems: Expressivity of Classical Problems

Theorem: $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$ – by Höller, Behnke, Bercher, Biundo, '14

Proof:

- We first show $\mathcal{CLASSIC} \subseteq \mathcal{REG}$
  - 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 $\mathcal{CLASSIC} \subsetneq \mathcal{REG}$.
  - We prove that $\{aa\} \in \mathcal{REG}$ 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 $\mathcal{P}$. We know that a leads into a state in which a is executable (since aa is executable). Hence, aaa must be executable as well! But then $aaa \in L(\mathcal{P})$, so $\{aa, aaa\} \subseteq L(\mathcal{P})$, and hence $L(\mathcal{P}) \neq \{aa\}$.

Expressivity of Classical Problems: Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathcal{CF}$ – by Höller, Behnke, Bercher, Biundo, '14

Expressivity of Classical Problems: Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathcal{CF}$ – by Höller, Behnke, Bercher, Biundo, '14

Proof:

- We first show $\mathcal{TOHTN} \supseteq \mathcal{CF}$.
  - In a nutshell, any context-free grammar *is* an HTN problem:
    - Let G be a context-free grammar. Use rules as methods, compound task names as non-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 Classical Problems: Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathcal{CF}$ – by Höller, Behnke, Bercher, Biundo, '14

Proof:

- We first show $\mathcal{TOHTN} \supseteq \mathcal{CF}$.
  - In a nutshell, any context-free grammar *is* an HTN problem:
    - Let G be a context-free grammar. Use rules as methods, compound task names as non-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 $\mathcal{TOHTN} \subseteq \mathcal{CF}$.

Expressivity of Classical Problems: Expressivity of HTN Problems

Theorem: $\mathcal{TOHTN} = \mathcal{CF}$ – by Höller, Behnke, Bercher, Biundo, '14

Proof:

- We first show $\mathcal{TOHTN} \supseteq \mathcal{CF}$.
  - In a nutshell, any context-free grammar *is* an HTN problem:
    - Let G be a context-free grammar. Use rules as methods, compound task names as non-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 $\mathcal{TOHTN} \subseteq \mathcal{CF}$.
  - We know that $L_{\mathcal{HTN}}(\mathcal{P}) = L_H(\mathcal{P}) \cap L_C(\mathcal{P})$ 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.

Complexity

Complexity of General Case: Complexity of HTN Planning

Theorem: HTN Planning is undecidable. – by Erol et al., ’94

We reduce from the (undecidable) grammar intersection problem.
Given context-free grammars G and G', construct HTN problem to answer $L(G) \cap L(G') \stackrel{?}{=} \emptyset$.

Complexity of General Case: Complexity of HTN Planning

Theorem: HTN Planning is undecidable. – by Erol et al., '94

We reduce from the (undecidable) grammar intersection problem. Given context-free grammars G and G', construct HTN problem to answer $L(G) \cap L(G') \stackrel{?}{=} \emptyset$.

Decision procedure:

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

Complexity of General Case: Complexity of HTN Planning

Theorem: HTN Planning is undecidable. – by Erol et al., '94

We reduce from the (undecidable) grammar intersection problem. Given context-free grammars G and G', construct HTN problem to answer $L(G) \cap L(G') \stackrel{?}{=} \emptyset$.

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.

Complexity of General Case: Complexity of HTN Planning

Theorem: HTN Planning is undecidable. – by Erol et al., '94

We reduce from the (undecidable) grammar intersection problem. Given context-free grammars G and G', construct HTN problem to answer $L(G) \cap L(G') \stackrel{?}{=} \emptyset$.

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 $\omega' \in L(G')$.

Complexity of General Case: Complexity of HTN Planning

Theorem: HTN Planning is undecidable. – by Erol et al., '94

We reduce from the (undecidable) grammar intersection problem. Given context-free grammars G and G', construct HTN problem to answer $L(G) \cap L(G') \stackrel{?}{=} \emptyset$.

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.

Complexity of General Case: Example Reduction

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

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

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

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

Complexity of General Case: Example Reduction

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

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

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

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

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

Complexity of General Case: Example Reduction

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

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

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

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

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

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

Complexity of General Case: Example Reduction

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

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

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

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

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

$M = M(G) \cup M(G')$ (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 Special Cases: Decidable Subclasses

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. (I.e., ignore all delete effects.)
- 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).

Modeling Support

Motivation

Why do we need modeling support?

- Writing planning models (e.g., in PDDL/HDDL) is challenging!
- Typical problems that can occur:
  - Problem is unsolvable, $L(\mathcal{P}) = \emptyset$
  - We have an undesired solution, $\bar{a} \in L(\mathcal{P})$
  - A desired solution can't be found, $\bar{a} \notin L(\mathcal{P})$

Motivation

Why do we need modeling support?

- Writing planning models (e.g., in PDDL/HDDL) is challenging!
- Typical problems that can occur:
  - Problem is unsolvable, $L(\mathcal{P}) = \emptyset$
  - We have an undesired solution, $\bar{a} \in L(\mathcal{P})$
  - A desired solution can't be found, $\bar{a} \notin L(\mathcal{P})$
- Generalizations of the above:
  - Provide language(s) L (e.g., via regular expression), such that:
    - $L \subseteq L(\mathcal{P})$, $L \supseteq L(\mathcal{P})$, or both
    - $L \cap L(\mathcal{P}) = \emptyset$ or $L \cap L(\mathcal{P}) \neq \emptyset$
    - Demand that solutions of certain size exist.

How to support?

How to support with the desired properties? (Or beyond?)

1. Simply *check* properties.
   - Plan existence problem. (Is there a solution? Of certain length?)
   - Plan verification. (Is the given plan a solution?)
2. *Establish* properties by changing the model automatically.
   - Change action specifications.
   - Change, add, or delete methods.

How to support?

How to support with the desired properties? (Or beyond?)

1. Simply *check* properties.
   - Plan existence problem. (Is there a solution? Of certain length?)
   - Plan verification. (Is the given plan a solution?)
2. *Establish* properties by changing the model automatically.
   - Change action specifications.
   - Change, add, or delete methods.

Further important questions:

- Which corrections do we prefer?
  - Minimize number of changes?
  - Minimize $||L(\mathcal{P}')| - |L(\mathcal{P})||$?
  - Or aim at $|L(\mathcal{P}')| > |L(\mathcal{P})|$? Or the other way round?
  - Follow a user-specific preference? How would that look?
- Many more! (Almost no research up to date!)

My Research up to Date

Complexity investigations:

- Given a non-solution plan, how hard is it to:
  - Change action preconditions/effects to turn it into solution
  - Change methods to make plan a refinement of tn_I.

My Research up to Date

Complexity investigations:

- Given a non-solution plan, how hard is it to:
  - Change action preconditions/effects to turn it into solution
  - Change methods to make plan a refinement of tn_I.

Empirical work:

- Given a non-solution plan,
  - Change action preconditions/effects to turn it into solution.
    - → By exploiting duality, encoding into hitting sets.
    - → Encoding into SAT. (Future work)
  - Add actions into methods to make plan a refinement of tn_I.
    - → Encoding into HTN problem. (Under review)
    - → Encoding into SAT. (Future work)

Summary

Summary

Today we've seen:

- Introduction to and applications of:
  - Classical (= non-hierarchical) planning.
  - Hierarchical Task Network (HTN) planning.
- The close relationship of HTN planning and formal grammars.
  - For expressivity analysis.
  - For computational complexity investigations.
- Short except to modeling support.
  - Which problems would we like to solve?
  - (What to do about it?)

Summary

Today we've seen:

- Introduction to and applications of:
  - Classical (= non-hierarchical) planning.
  - Hierarchical Task Network (HTN) planning.
- The close relationship of HTN planning and formal grammars.
  - For expressivity analysis.
  - For computational complexity investigations.
- Short except to modeling support.
  - Which problems would we like to solve?
  - (What to do about it?)

Thank you for listening!