Automated Planning and Scheduling

Lecture 7: SAT-based Planning, Hierarchical Planning

Tomáš Balyo, Dominik Schreiber | November 27, 2019
Last week: Brief introduction to SAT-based planning

Today: **Hierarchical Planning**

Next week: Advanced SAT-based planning, SAT-based hierarchical planning
Motivation

Share your domain-specific knowledge with your planner.
Motivation

Share your domain-specific knowledge with your planner.

Example: Cooking

- Suppose we program a robot which cooks meals autonomously
Motivation

Share your domain-specific knowledge with your planner.

Example: Cooking

- Suppose we program a robot which cooks meals autonomously
- Plan the cooking using classical planning
  - Objects: All kinds of ingredients, tools, pans, ...
  - Actions: Put object from $x$ to $y$, turn on stove, chop stuff, ...
  - Goal: Have a finished meal on the table
Motivation

Share your domain-specific knowledge with your planner.

Example: Cooking

- Suppose we program a robot which cooks meals autonomously
- Plan the cooking using classical planning
  - Objects: All kinds of ingredients, tools, pans, . . .
  - Actions: Put object from $x$ to $y$, turn on stove, chop stuff, . . .
  - Goal: Have a finished meal on the table
- Get some plan fulfilling your problem model
  - Unstructured: Order of done tasks more or less arbitrary
  - Can be sub-optimal w.r.t. actual real-world actions the robot does
  - Finding a plan can be very expensive
Motivation

Share your domain-specific knowledge with your planner.

Example: Cooking

- Suppose we program a robot which cooks meals autonomously
- Plan the cooking using classical planning
  - Objects: All kinds of ingredients, tools, pans, . . .
  - Actions: Put object from x to y, turn on stove, chop stuff, . . .
  - Goal: Have a finished meal on the table
- Get some plan fulfilling your problem model
  - Unstructured: Order of done tasks more or less arbitrary
  - Can be sub-optimal w.r.t. actual real-world actions the robot does
  - Finding a plan can be very expensive
- Better idea: Tell the planner what you know about cooking
Motivation: A Cooking Task Network (1)

- Ovals: **non-primitive tasks** (expand to sequences of **subtasks**)
- Rectangles: **primitive tasks** (correspond to classical actions)
After expanding all tasks in the network exhaustively,

... all leaves correspond to classical actions
... traversing all leaves from left to right yields a plan
Motivation: A Cooking Task Network (2)

- After expanding all tasks in the network exhaustively,
  - all leaves correspond to classical actions
  - traversing all leaves from left to right yields a plan
- How does this structure differ from classical planning?
  - Plan is structured in a natural way (intuitive subtasks can be identified)
  - Search space of planner is drastically reduced
Towards Hierarchical Task Networks

What do we keep from classical planning?

- States: Consistent sets of atoms
- Goal: Atoms which need to be achieved (now: **optional**)
- Actions: Only way to **alter states**
  - Will be “leaves” in our hierarchy

What is new?

- Tasks: What needs to be achieved not just in the end, but during the entire procedure
- Methods: How a task may be achieved
- Constraints: What the application of some method enforces on the world state before / during / after achieving the task on the spawned subtasks
- Top-level objective: Initial task network to achieve
Towards Hierarchical Task Networks

What do we keep from classical planning?

- **States**: Consistent sets of atoms
- **Goal**: Atoms which need to be achieved (now: optional)
- **Actions**: Only way to alter states
  - Will be “leaves” in our hierarchy

What is new?

- **Tasks**: *What* needs to be achieved
  - Not just in the end, but during the entire procedure
- **Methods**: *How* a task may be achieved
- **Constraints**: What the application of some method enforces
  - on the world state before / during / after achieving the task
  - on the spawned subtasks
- **Top-level objective**: Initial task network to achieve
Hierarchical Planning: Tasks

**Task**

A *task* is a syntactical expression which represents a certain operational objective of the problem at hand. A task is either *primitive* or *compound*. If it is primitive, it can be achieved by applying a certain action. If it is compound, it can be achieved by applying certain *methods*.

Examples:

- Task `put(plate, table)`: *primitive*, achieved by action `put(plate, table)` (Primitive task and its action are *interchangeable*)
Hierarchical Planning: Tasks

**Task**

A *task* is a syntactical expression which represents a certain operational objective of the problem at hand. A task is either *primitive* or *compound*. If it is primitive, it can be achieved by applying a certain action. If it is compound, it can be achieved by applying certain *methods*.

Examples:

- **Task put(plate, table):**
  - *primitive*, achieved by action `put(plate, table)`
  (Primitive task and its action are interchangeable)

- **Task chop_stuff():**
  - *compound*, achieved by multiple *subtasks*
    `⟨grab(knife), chop(onions, table)⟩`
Hierarchical Planning: Task Networks

**Task Network**

A task network \((T, C)\) is a set of tasks \(T\) with a set of constraints \(C\). A constraint has one of the following forms:

1. \( t_1 \prec t_2 \): Ordering constraint; \(t_1\) must be achieved before \(t_2\)
2. \((p, t)\): “Before” constraint; atom \(p\) holds before \(t\) is achieved
3. \((t, p)\): “After” constraint; atom \(p\) holds after \(t\) is achieved
4. \((t_1, p, t_2)\): “Between” constraint; atom \(p\) holds between achieving \(t_1\) and achieving \(t_2\)
A task network \((T, C)\) is a set of tasks \(T\) with a set of constraints \(C\). A constraint has one of the following forms:

1. \(t_1 \prec t_2\): Ordering constraint; \(t_1\) must be achieved before \(t_2\)
2. \((p, t)\): “Before” constraint; atom \(p\) holds before \(t\) is achieved
3. \((t, p)\): “After” constraint; atom \(p\) holds after \(t\) is achieved
4. \((t_1, p, t_2)\): “Between” constraint; atom \(p\) holds between achieving \(t_1\) and achieving \(t_2\)

Example constraints in a task network for \(chop\_stuff\):

1. \((\text{grab(knife)} \prec \text{chop(onions, table)})\)
2. \((\text{at(onions, table)}, \text{chop(onions, table)})\)
3. \((\text{chop(onions, table)}, \text{chopped(onions)})\)
4. \((\text{grab(knife)}, \text{at(knife, hand)}, \text{chop(onions, table)})\)
A method \( m \) is a “recipe” for how to achieve a certain compound task \( t \). It contains a task network \( (subtasks(m), constraints(m)) \).
Hierarchical Planning: Methods

Method

A method $m$ is a “recipe” for how to achieve a certain compound task $t$. It contains a task network ($\text{subtasks}(m)$, $\text{constraints}(m)$).

- Method $m$ for task $t = \text{chop\_stuff}$:

  $\text{subtasks}(m) = \{t_1: \text{grab(knife)}, t_2: \text{chop(onions, table)}\}$

  $\text{constraints}(m) = \{(t_1 \prec t_2), (\text{at(onions, table)}, t_2), (t_2, \text{chopped(onions)}), (t_1, \text{at(knife, hand)}, t_2)\}$
Hierarchical Planning: Methods

Method

A method \( m \) is a “recipe” for how to achieve a certain compound task \( t \).
It contains a task network \((\text{subtasks}(m), \text{constraints}(m))\).

- Method \( m \) for task \( t = \text{chop\_stuff} \):
  \[
  \text{subtasks}(m) = \{t_1 : \text{grab(knife)}, t_2 : \text{chop(onions, table)}\}
  \]
  \[
  \text{constraints}(m) = \{(t_1 \prec t_2), (\text{at(onions, table)}, t_2),
  (t_2, \text{chopped(onions)}), (t_1, \text{at(knife, hand)}, t_2)\}
  \]

- Workings of methods in general:
  - One task can have multiple methods with different task networks
    (planner needs to pick one!)
  - A method for a task \( t \) may feature \( t \) in its task network (recursion)
  - Constraints of a task network may be empty
  - If subtasks are totally ordered: Just write \( \langle \ldots \rangle \) instead of \( \{\ldots\} \),
    keep needed ordering constraints implicit
Task Network Example: Navigation

- Agent at some position $x$, walks on a map of waypoints
- In classical planning: operator $\text{move}(x, y)$, atoms $\text{at}(x)$, $\text{road}(x, y)$ and $\text{visited}(x)$
  - “Random” chaining of $\text{move}$ actions until goal is reached

$\text{HTN}: \text{Define task } \text{navigate}(x, z)$ with two different methods:

Method $m_1$:
- subtasks ($m_1$) = {$t_1$: $\text{nop}()$} (do nothing),
- constraints ($m_1$) = {$(\text{at}(x), t_1)$, $(x = z, t_1)$}

Method $m_2$:
- subtasks ($m_2$) = {$t_1$: $\text{move}(x, y)$, $t_2$: $\text{navigate}(y, z)$} (direct move + recursion),
- constraints ($m_2$) = {$(\text{at}(x), t_1)$, $(\text{road}(x, y), t_1)$, $(t_1 \prec t_2)$, $(t_1, \text{at}(y), t_2)$}

(Note we could add an after constraint $\text{at}(z)$ to each of the methods)
Task Network Example: Navigation

- Agent at some position $x$, walks on a map of waypoints
- In classical planning: operator $\text{move}(x, y)$, atoms $\text{at}(x), \text{road}(x, y)$ and $\text{visited}(x)$
  - “Random” chaining of $\text{move}$ actions until goal is reached
- HTN: Define task $\text{navigate}(x, z)$ with two different methods: Base case, and recursion
  - Method $m_1$: $\text{subtasks}(m_1) = \{t_1 : \text{nop}()\}$ (do nothing), $\text{constraints}(m_1) = \{(\text{at}(x), t_1), (x = z, t_1)\}$
Task Network Example: Navigation

- Agent at some position \( x \), walks on a map of waypoints
- In classical planning: operator \( \text{move}(x, y) \),
  atoms \( \text{at}(x) \), \( \text{road}(x, y) \) and \( \text{visited}(x) \)
- “Random” chaining of \( \text{move} \) actions until goal is reached
- HTN: Define task \( \text{navigate}(x, z) \) with two different methods:
  Base case, and recursion
  - Method \( m_1 \): \( \text{subtasks}(m_1) = \{ t_1 : \text{nop}() \} \) (do nothing),
    \( \text{constraints}(m_1) = \{ (\text{at}(x), t_1), (x = z, t_1) \} \)
  - Method \( m_2 \): \( \text{subtasks}(m_2) = \{ t_1 : \text{move}(x,y), t_2 : \text{navigate}(y,z) \} \)
    (direct move + recursion),
    \( \text{constraints}(m_2) = \{ (\text{at}(x), t_1), (\text{road}(x,y), t_1), (t_1 \prec t_2), (t_1, \text{at}(y), t_2) \} \)
  - (Note we could add an \( \text{after} \) constraint \( \text{at}(z) \) to each of the methods)
Hierarchical Navigation: Example

Graph of waypoints:

Initial position 1, goal position 5

navigate(1,5)
m2
move(1,2)
navigate(2,5)
m2
move(2,3)
navigate(3,5)
m2
move(3,4)
navigate(4,5)
m2
move(4,5)
navigate(5,5)
m1
nop()
An HTN planning problem $\mathcal{P} = (P, O, M, s_0, T_0)$ is a tuple where:

- $P$, $O$, and $M$ are the sets of atoms, operators, and methods;
- $s_0$ is the initial state; and
- $T_0$ is the initial task network.
### HTN Planning: Problems

#### HTN (Hierarchical Task Network) Planning Problem

An **HTN planning problem** $\mathcal{P} = (P, O, M, s_0, T_0)$ is a tuple where:

- $P$, $O$, and $M$ are the sets of atoms, operators, and methods;
- $s_0$ is the initial state; and
- $T_0$ is the *initial task network*.

What is a **ground** HTN planning problem in that sense?

- Replace operators $O$ with their actions $A$
- Similarly, replace methods $M$ with **ground methods** $D$
  (sometimes called **reductions** or **decompositions**)
- Initial task network is always ground (just like initial state)
Formal definition of HTN solutions?

- HTN planning is **structurally complex**: Completely formal definition of HTN solution plans uses operational semantics.
Formal definition of HTN solutions?

- HTN planning is **structurally complex**: Completely formal definition of HTN solution plans uses operational semantics
- Idea of finding a plan to an HTN planning problem:
  - Resolve initial task network step by step in chronological order
    (constraint $t_1 \prec t_2 \Rightarrow$ process $t_1$ before $t_2$)
Formal definition of HTN solutions?

- HTN planning is **structurally complex**: Completely formal definition of HTN solution plans uses operational semantics.
- Idea of finding a plan to an HTN planning problem:
  - Resolve initial task network step by step in **chronological order** (constraint $t_1 \prec t_2 \implies$ process $t_1$ before $t_2$)
  - If a **primitive task** can be achieved regarding the current constraints, remove it, apply corresponding action and append it to the plan.
Formal definition of HTN solutions?

- HTN planning is **structurally complex**: Completely formal definition of HTN solution plans uses operational semantics.
- Idea of finding a plan to an HTN planning problem:
  - Resolve initial task network step by step in chronological order (constraint $t_1 \prec t_2 \Rightarrow$ process $t_1$ before $t_2$).
  - If a **primitive task** can be achieved regarding the current constraints, remove it, apply corresponding action and append it to the plan.
  - If a **compound task** can be achieved by some method, apply it and update task network with new subtasks and new constraints.
Formal definition of HTN solutions?

- HTN planning is **structurally complex**: Completely formal definition of HTN solution plans uses operational semantics
- Idea of finding a plan to an HTN planning problem:
  - Resolve initial task network step by step in **chronological order** (constraint $t_1 \prec t_2 \Rightarrow$ process $t_1$ before $t_2$)
  - If a **primitive task** can be achieved regarding the current constraints, remove it, apply corresponding action and append it to the plan
  - If a **compound task** can be achieved by some method, apply it and update task network with new subtasks and new constraints
  - If all tasks have been **achieved** and all constraints have been **satisfied**, return the plan
HTN planning: Solutions

HTN Solution Plan (semi-formal)

A sequence of actions \( \pi = \langle a_1, \ldots, a_n \rangle \) is a solution plan for a ground HTN planning problem \( \mathcal{P} = (P, A, D, s_0, T_0) \), \( T_0 = (T, C) \), if one of the following alternatives holds:

1. \( T \) is empty, and \( n = 0 \).

2. Achieving a primitive task \( t \in T \) meets the constraints \( C \) in \( s_0 \), its corresponding action \( a_1 \) is applicable in \( s_0 \), and \( \pi' = \langle a_2, \ldots, a_n \rangle \) is a solution plan for \( \mathcal{P}' = (P, A, D, \gamma(s_0, a_1), (T \setminus \{t\}, C)) \).

3. Applying a ground method \( m \) of a compound task \( t \in T \) meets the constraints \( C \) in \( s_0 \), and \( \pi \) is a solution plan for \( \mathcal{P}' = (P, A, D, s_0, (T \setminus \{t\} \cup \text{subtasks}(m), C \cup \text{constraints}(m))) \).
Example of deriving a plan in an HTN planning problem:

- Task network $T_0 = (\{t_1 : \text{cook spaghetti}\},
\{(t_1, \text{at(table,plate)})\})$, partial plan $\pi = \langle \rangle$

Finally:

$T_0 = \emptyset$, $\pi = \langle \text{put(noodles,cupboard,table), put(tomatoes,fridge,table), put(onions,can,table),... put(noodles,plate),put(sauce,plate),put(plate,table)} \rangle$. 
Example of deriving a plan in an HTN planning problem:

- Task network $T_0 = (\{t_1: \text{cook\_spaghetti}\},$
  
  $\{(t_1, \text{at\(table\),plate})\})$, partial plan $\pi = \langle \rangle$

- Reduce $t_1$ (with some method):
  
  $T_0 = (\{t_1: \text{prepare\_ingredients}, t_2: \text{cook\_noodles},$
  
  $t_3: \text{cook\_sauce}, t_4: \text{serve}\},$
  
  $\{(t_4, \text{at\(table\),plate}), t_1 < t_2, t_2 < t_3, \ldots \}), \pi = \langle \rangle$

- ...
Example of deriving a plan in an HTN planning problem:

- Task network $T_0 = (\{t_1 : \text{cook spaghetti}\},\{(t_1, \text{at}(\text{table}, \text{plate}))\})$, partial plan $\pi = \emptyset$

- Reduce $t_1$ (with some method):
  $T_0 = (\{t_1 : \text{prepare ingredients}, t_2 : \text{cook noodles}, t_3 : \text{cook sauce}, t_4 : \text{serve}\},\{(t_4, \text{at}(\text{table}, \text{plate})), t_1 < t_2, t_2 < t_3, \ldots \}), \pi = \emptyset$

- ...  

- Finally: $T_0 = \emptyset$, $\pi = \langle \text{put(noodles, cupboard, table)}, \text{put(tomatoes, fridge, table)}, \text{put(onions, can, table)}, \ldots, \text{put(noodles, plate)}, \text{put(sauce, plate)}, \text{put(plate, table)} \rangle$. 
Trucking domain in HTN (any number of trucks, packages):

- Task: deliver-package(p, loc)
- Method: m
  - subtasks(m) = ⟨navigate(truck,x), load(truck,p), navigate(truck,loc), drop(truck,p)⟩
  - constraints(m) = {at(p,x), t₁, empty(t), t₁, ...}

Note: The truck to use is not part of the task definition!
⇒ Implicit parameter of the method
⇒ Truck is picked when planner picks a (ground) method
Trucking domain in HTN (any number of trucks, packages):

- Task deliver-package(p, loc)
  - Method \( m \): \( \text{subtasks}(m) = \langle \text{navigate(truck,x)}, \text{load(truck,p)}, \text{navigate(truck,loc)}, \text{drop(truck,p)} \rangle \),
  \( \text{constraints}(m) = \{(\text{at(p,x)}, t_1), (\text{empty(t)}, t_1), \ldots \} \)
HTN Trucking

Trucking domain in HTN (any number of trucks, packages):

- Task \text{deliver-package}(p, \text{loc})
- Method \textit{m}: \textit{subtasks}(m) = \langle \text{navigate(truck,x)}, \text{load(truck,p)}, \text{navigate(truck,loc)}, \text{drop(truck,p)} \rangle,
  \textit{constraints}(m) = \{ (\text{at}(p,x), t_1), (\text{empty}(t), t_1), \ldots \}\)
- Note: The truck to use is not part of the \textit{task definition}!
  \Rightarrow \text{Implicit parameter} of the method
  \Rightarrow Truck is picked when planner picks a (ground) method
Trucking domain in HTN (any number of trucks, packages):

- Task deliver-package(p, loc)
  - Method m: subtasks(m) = ⟨ navigate(truck,x), load(truck,p), navigate(truck,loc), drop(truck,p) ⟩,
    constraints(m) = {(at(p,x), t₁), (empty(t), t₁),...}
  - Note: The truck to use is not part of the task definition!
    ⇒ Implicit parameter of the method
    ⇒ Truck is picked when planner picks a (ground) method

- Task navigate(truck,x): Navigation procedure as described

- Tasks load(t,p) and drop(t,p): primitive
Trucking domain in HTN (any number of trucks, packages):

- Task deliver-package(p, loc)
- Method $m$: $\text{subtasks}(m) = \langle \text{navigate}(\text{truck},x), \text{load}(\text{truck},p),$ $\text{navigate}(\text{truck},\text{loc}), \text{drop}(\text{truck},p) \rangle$, $\text{constraints}(m) = \{(\text{at}(p,x), t_1), (\text{empty}(t), t_1), \ldots\}$
- Note: The truck to use is not part of the task definition!
  ⇒ Implicit parameter of the method
  ⇒ Truck is picked when planner picks a (ground) method

- Task navigate(truck,x): Navigation procedure as described
- Tasks load(t,p) and drop(t,p): primitive
- Initial tasks deliver-package(p,loc) for each package $p$ with destination loc
How does HTN practically help model planning domains?

- Express \textit{stepwise refinement} of abstract tasks (very intuitive)
- Supply methods with \textit{expressive constraints}, focusing search of a planner
How does HTN practically help model planning domains?

- Express stepwise refinement of abstract tasks (very intuitive)
- Supply methods with expressive constraints, focusing search of a planner

Is HTN planning theoretically more powerful than STRIPS planning?

- Can we simulate a classical planning problem with HTN?
HTN vs. Classical Planning

How does HTN practically help model planning domains?

- Express stepwise refinement of abstract tasks (very intuitive)
- Supply methods with expressive constraints, focusing search of a planner

Is HTN planning theoretically more powerful than STRIPS planning?

- Can we simulate a classical planning problem with HTN?
  - Sure: See exercises
- Can we simulate an HTN planning problem with classical planning?
How does HTN practically help model planning domains?

- Express stepwise refinement of abstract tasks (very intuitive)
- Supply methods with expressive constraints, focusing search of a planner

Is HTN planning theoretically more powerful than STRIPS planning?

- Can we simulate a classical planning problem with HTN?
  - Sure: See exercises
- Can we simulate an HTN planning problem with classical planning?
  - No!
Undecidability of HTN Planning

Theorem. [EHN94]
Given an HTN planning problem $\mathcal{P}$, it is generally undecidable whether $\mathcal{P}$ is solvable (i.e. has a solution plan).

- Consequence: Can only have semi-decidable planning procedures
  - If a plan exists, it will eventually be found
  - If no plan exists, you may never know
**Theorem. [EHN94]**

Given an HTN planning problem $\mathcal{P}$, it is generally **undecidable** whether $\mathcal{P}$ is solvable (i.e. has a solution plan).

- **Consequence:** Can only have **semi-decidable** planning procedures
  - If a plan exists, it will eventually be found
  - If no plan exists, you may never know
- **Proof:** Model an undecidable problem as an HTN planning problem
  $\Rightarrow$ HTN planning cannot be decidable then!
- **Possible candidates?**
Undecidability of HTN Planning

Theorem. [EHN94]
Given an HTN planning problem $\mathcal{P}$, it is generally undecidable whether $\mathcal{P}$ is solvable (i.e. has a solution plan).

- Consequence: Can only have semi-decidable planning procedures
  - If a plan exists, it will eventually be found
  - If no plan exists, you may never know

- Proof: Model an undecidable problem as an HTN planning problem
  $\Rightarrow$ HTN planning cannot be decidable then!

- Possible candidates?
  - Halting problem: quite cumbersome and technical
Undecidability of HTN Planning

Theorem. [EHN94]

Given an HTN planning problem $\mathcal{P}$, it is generally undecidable whether $\mathcal{P}$ is solvable (i.e. has a solution plan).

- Consequence: Can only have semi-decidable planning procedures
  - If a plan exists, it will eventually be found
  - If no plan exists, you may never know

- Proof: Model an undecidable problem as an HTN planning problem
  $\Rightarrow$ HTN planning cannot be decidable then!

- Possible candidates?
  - Halting problem: quite cumbersome and technical
  - Undecidable problems from formal languages: Much better suited
  - Post Correspondence Problem: Simpler, more intuitive
The Post Correspondence Problem (PCP) [Pos46]

Let $\Sigma$ an alphabet with $|\Sigma| \geq 2$, and $A := \langle a_1, \ldots, a_k \rangle$ and $B := \langle b_1, \ldots, b_k \rangle$ two sequences of words from $\Sigma$. $(a_i, b_i \in \Sigma^*)$

Is there a sequence of indices $\langle i_1, i_2, \ldots, i_N \rangle$, $N \geq 1$, such that

$$a_{i_1} a_{i_2} \ldots a_{i_N} = b_{i_1} b_{i_2} \ldots b_{i_N},$$

i.e. the concatenations of chosen words from $A$ and $B$ perfectly match?

Objective: Model PCP as an HTN planning problem
The Post Correspondence Problem (PCP) [Pos46]

Let \( \Sigma \) an alphabet with \(|\Sigma| \geq 2\), and \( A := \langle a_1, \ldots, a_k \rangle \) and \( B := \langle b_1, \ldots, b_k \rangle \) two sequences of words from \( \Sigma \). \((a_i, b_i \in \Sigma^*)\)

Is there a sequence of indices \( \langle i_1, i_2, \ldots, i_N \rangle \), \( N \geq 1 \), such that

\[
    a_{i_1} a_{i_2} \ldots a_{i_N} = b_{i_1} b_{i_2} \ldots b_{i_N},
\]

i.e. the concatenations of chosen words from \( A \) and \( B \) perfectly match?

Objective: Model PCP as an HTN planning problem

- Make the planner pick a sequence of indices
- Symbol for symbol, match the resulting strings against each other
An HTN planning domain for PCP (1)

- Atoms: turnA and turnB (whose turn is it to add a symbol?), symbol(x) for all x ∈ Σ (which symbol is added?), picked(i) for all i ∈ 1, . . . , N (what’s the “current” picked index i?)
An HTN planning domain for PCP (1)

- Atoms: $\text{turnA}$ and $\text{turnB}$ (whose turn is it to add a symbol?), $\text{symbol}(x)$ for all $x \in \Sigma$ (which symbol is added?), $\text{picked}(i)$ for all $i \in 1, \ldots, N$ (what’s the “current” picked index $i$?)

- All actions either (require $\text{turnA}$, delete $\text{turnA}$ and add $\text{turnB}$), or (require $\text{turnB}$, delete $\text{turnB}$ and add $\text{turnA}$)
  - Strict alternation between actions concerning $A$ and actions concerning $B$
  - Action $\text{pickIndex}(i)$ sets atom $\text{picked}(i)$, action $\text{matchIndex}(i)$ requires and deletes $\text{picked}(i)$
  - Action $\text{print}(x)$ sets atom $\text{symbol}(x)$, action $\text{match}(x)$ requires and deletes $\text{symbol}(x)$

- Initial tasks: $T_0 = \{\text{startA()}, \text{startB()}\}$
An HTN planning domain for PCP (2)

- Initial tasks: \( T_0 = \{ \text{startA()}, \text{startB()}) \)
- For each word in \( A (B) \), add a method for \( \text{startA} (\text{startB}) \)
- Say, \( \Sigma := \{0, 1\} \), \( A \supset a_1 := 001 \) and \( B \supset b_1 := 0 \)
  - Add possible method for \( \text{startA}() \) with ordered subtasks \( \langle \text{pickIndex(1)}, \text{startA()}, \text{print(0)}, \text{print(0)}, \text{print(1)} \rangle \)
  - Add possible method for \( \text{startB}() \) with ordered subtasks \( \langle \text{matchIndex(1)}, \text{startB()}, \text{match(0)} \rangle \)
An HTN planning domain for PCP (2)

- Initial tasks: $T_0 = \{\text{startA()}, \text{startB()}\}$
- For each word in $A$ ($B$), add a method for $\text{startA}$ ($\text{startB}$)
- Say, $\Sigma := \{0, 1\}$, $A \supseteq a_1 := 001$ and $B \supseteq b_1 := 0$
  - Add possible method for $\text{startA}()$ with ordered subtasks
    $\langle \text{pickIndex}(1), \text{startA}(), \text{print}(0), \text{print}(0), \text{print}(1) \rangle$
  - Add possible method for $\text{startB}()$ with ordered subtasks
    $\langle \text{matchIndex}(1), \text{startB}(), \text{match}(0) \rangle$
- Structure of derivable plans:
  1. First part: $\langle \text{pickIndex}(iN), \text{matchIndex}(iN), \ldots \text{pickIndex}(i2),\text{matchIndex}(i2), \text{pickIndex}(i1), \text{matchIndex}(i1) \rangle$
  2. Second part: $\langle \text{print}(x1), \text{match}(x1), \text{print}(x2), \text{match}(x2), \ldots, \text{print}(xZ), \text{match}(xZ) \rangle$
- Such a plan can be derived from the HTN problem if and only if there is such a matching in the original PCP instance
HTN Undecidability: Remarks

- As PCP is undecidable, HTN planning is undecidable as well
- Why can’t we model PCP in classical planning?

Direct consequence:

Expressiveness of HTN planning

There are planning problems which can be solved with HTN formalisms, but not with classical planning formalisms.
HTN Undecidability: Remarks

- As PCP is undecidable, HTN planning is undecidable as well
- Why can’t we model PCP in classical planning?
  - Needed sequence of indices may be arbitrarily long
  - Need to pick same indices for both A and B, but the corresponding words may occur at very different times in the plan
  - “Memory” of any given state bounded by $2^{|P|}$ in classical planning
HTN Undecidability: Remarks

- As PCP is undecidable, HTN planning is undecidable as well
- Why can’t we model PCP in classical planning?
  - Needed sequence of indices may be arbitrarily long
  - Need to pick same indices for both A and B, but the corresponding words may occur at very different times in the plan
  - “Memory” of any given state bounded by $2^{|P|}$ in classical planning
- Direct consequence:

Expressiveness of HTN planning

There are planning problems which can be solved with HTN formalisms, but not with classical planning formalisms.
HTN Expressiveness

Where does the power of HTN come from?
HTN Expressiveness

Where does the power of HTN come from?

- Hierarchical search states of **arbitrary depth** (Recursive tasks)
- Interleaving of initial tasks
  (complex interaction between tasks possible)
HTN Expressiveness

Where does the power of HTN come from?

- Hierarchical search states of arbitrary depth (Recursive tasks)
- Interleaving of initial tasks
  (complex interaction between tasks possible)

Which features of HTN are (theoretically) unimportant?
HTN Expressiveness

Where does the power of HTN come from?

- Hierarchical search states of arbitrary depth \textit{(Recursive tasks)}
- Interleaving of initial tasks
  (complex interaction between tasks possible)

Which features of HTN are (theoretically) unimportant?

- Causal constraints (other than ordering)
- (Partial order and interleaving) of non-initial tasks
HTN Expressiveness

Where does the power of HTN come from?
- Hierarchical search states of arbitrary depth (Recursive tasks)
- Interleaving of initial tasks
  (complex interaction between tasks possible)

Which features of HTN are (theoretically) unimportant?
- Causal constraints (other than ordering)
- (Partial order and interleaving) of non-initial tasks

Decidable subclasses:
HTN Expressiveness

Where does the power of HTN come from?

- Hierarchical search states of arbitrary depth (Recursive tasks)
- Interleaving of initial tasks
  (complex interaction between tasks possible)

Which features of HTN are (theoretically) unimportant?

- Causal constraints (other than ordering)
- (Partial order and interleaving) of non-initial tasks

Decidable subclasses:

- Task networks restricted to regular structure
  - Each task network is an ordered sequence $\langle a, t \rangle$ or $\langle a \rangle$
  - Same expressive power as classical planning
HTN Expressiveness

Where does the power of HTN come from?

- Hierarchical search states of arbitrary depth (Recursive tasks)
- Interleaving of initial tasks
  (complex interaction between tasks possible)

Which features of HTN are (theoretically) unimportant?

- Causal constraints (other than ordering)
- (Partial order and interleaving) of non-initial tasks

Decidable subclasses:

- Task networks restricted to regular structure
  - Each task network is an ordered sequence $\langle a, t \rangle$ or $\langle a \rangle$
  - Same expressive power as classical planning
- Totally ordered tasks on all levels
  - Disallows interleaving $\Rightarrow$ no complex interactions between tasks
Our definition of HTN solutions is already an abstract algorithm
- Chronological processing of tasks and actions
- Maintain a state during search (as in classical forward search)
- Non-deterministic choice of tasks and methods to use
Our definition of HTN solutions is already an abstract algorithm
- **Chronological processing** of tasks and actions
- **Maintain a state** during search (as in classical forward search)
- Non-deterministic choice of tasks and methods to use

Mainly three approaches:
1. State space planning ("chronological", as described above)
2. Plan space planning (also "non-linear planning")
3. SAT-based HTN planning

Current mainstream approach: State space planning
Algorithm 1 SHOP2 Planning procedure [NAI⁺03] (simplified, abstract)

1: \( \pi = \langle \rangle; \ (T, C) := \) initial task network
2: while TRUE do
3: \quad if \( T = \emptyset \) then return \( \pi \) // everything achieved
4: \quad \quad \quad T' := \{ t \in T : \text{there is no } t' \text{ such that } t' < t \in C \}
5: \quad if \( T' = \emptyset \) then return FAILURE // no valid tasks to pick from
6: \quad if \( \exists t \in T' : t \text{ is primitive and its action } a \text{ is applicable in } s \) then
7: \quad \quad T := T \setminus \{ t \}; \quad \pi := \pi \circ a; \quad s := \gamma(s, a)
8: \quad else if \( \exists t \in T' : t \text{ is compound and one of its methods } m \text{ is applicable in } s \) then
9: \quad \quad T := T \setminus \{ t \} \cup \{ \text{subtasks}(m) \}
10: \quad \quad C := C \cup \{ \text{constraints}(m) \}
11: \quad else return FAILURE
12: end if
13: end while
SHOP2 Planner

Which simplifications are assumed in Alg. 1?
SHOP2 Planner

Which simplifications are assumed in Alg. 1?

- Check constraints of a method’s task network before its application
  - Problem with general constraints (especially between, after)
  - In SHOP2: Only before constraints for an entire method and ordering constraints supported
- Alternative: Add “markers” to check causal constraints later, even if parent task has been removed
SHOP2 Planner

Which simplifications are assumed in Alg. 1?

- Check constraints of a method’s task network before its application
  - Problem with general constraints (especially between, after)
  - In SHOP2: Only before constraints for an entire method and ordering constraints supported
  - Alternative: Add “markers” to check causal constraints later, even if parent task has been removed
- Constraints involving removed tasks are not removed from $C$
  - Remove concerned constraints, add updated ordering constraints
  - Alternative: Keep tasks in network as long as they are unfinished
Which simplifications are assumed in Alg. 1?

- Check constraints of a method’s task network before its application
  - Problem with general constraints (especially between, after)
  - In SHOP2: Only before constraints for an entire method and ordering constraints supported
  - Alternative: Add “markers” to check causal constraints later, even if parent task has been removed

- Constraints involving removed tasks are not removed from $C$
  - Remove concerned constraints, add updated ordering constraints
  - Alternative: Keep tasks in network as long as they are unfinished

- Non-deterministic choice of tasks and methods
  - Heuristics needed!
SHOP2 Planner: Remarks

- Predecessor SHOP (Simple Hierarchical Ordered Planner) [NCLMA99] only for totally ordered subtasks.
- SHOP2 also features partial orders, axioms, numerical and temporal planning, external procedure calls, . . .
SHOP2 Planner: Remarks

- Predecessor SHOP (Simple Hierarchical Ordered Planner) [NCLM99] only for totally ordered subtasks
- SHOP2 also features partial orders, axioms, numerical and temporal planning, external procedure calls, . . .
- Optional branch-and-bound optimization after finding initial solution
  - Keep searching for better plans
  - In general, no guarantees for optimal plans
SHOP2 Planner: Remarks

- Predecessor SHOP (Simple Hierarchical Ordered Planner) [NCLMA99] only for totally ordered subtasks
- SHOP2 also features partial orders, axioms, numerical and temporal planning, external procedure calls, ...
- Optional **branch-and-bound optimization** after finding initial solution
  - Keep searching for better plans
  - In general, no guarantees for optimal plans
- SHOP and SHOP2 operate on **lifted problem**, instantiation and unification just as needed
- Motivation for SHOP, SHOP2: Solving **practical problems** from industry, other research domains [NAI⁺05]
HDDL: HTN Planning in PDDL (1/3)

- No standardized PDDL extension to hierarchical planning
- But: Effort by different researchers at ICAPS 2019 leading to de facto standard HDDL for HTN planning [HBB+19]

Task definitions:

(task deliver :parameters (?p - pkg ?l - loc))
(task get-to :parameters (?l - loc))

Method definition (simple, totally ordered):

(method m-deliver :parameters (?p - pkg ?lp ?ld - loc)
  :task (deliver ?p ?ld)
  :ordered-subtasks (and
  )
)

Tomáš Balyo, Dominik Schreiber – Planning and Scheduling
November 27, 2019 28/34
Method definition (simple, partially ordered):
(:method m-drive-to-via :parameters (?li ?ld - loc)
 :task (get-to ?ld)
 :subtasks (and (t1 (get-to ?li)) (t2 (drive ?li ?ld)))
 :ordering (and (t1 < t2))
)

Method definitions (with preconditions and constraints):
(:method m-already-there :parameters (?l - loc)
 :task (get-to ?l)
 :precondition (tAt ?l) :subtasks ()
)
(:method m-direct :parameters (?ls ?ld - loc
 :task (get-to ?ld) :constraints (not (= ?li ?ld))
 :subtasks (drive ?ls ?ld))
)
Problem definition:
(define (problem p)
 (:domain transport)
 (:objects city-loc-0 city-loc-1 city-loc-2 - loc
 package-0 package-1 - pkg)
 (:htn
 :tasks (and
   (deliver package-0 city-loc-0)
   (deliver package-1 city-loc-2)
 )
 :ordering ()
 :constraints ()
)
 (:init (road city-loc-0 city-loc-1) ...))
Modeling HTN Domains: Some Notes

- Find right balance of **which method constraints to add**
  - State space HTN planners do not profit a lot from *after* constraints
  - Missing important method preconditions (*before*) can lead to terrible performance
  - Sometimes, can propagate action preconditions up to method level

Tomáš Balyo, Dominik Schreiber – Planning and Scheduling
November 27, 2019
Modeling HTN Domains: Some Notes

- Find right balance of **which method constraints to add**
  - State space HTN planners do not profit a lot from *after* constraints
  - Missing important method preconditions (*before*) can lead to terrible performance
  - Sometimes, can *propagate action preconditions* up to method level

- **Total order of subtasks** whenever it makes sense
  - Takes away choices the planner would need to do
  - Allowing interleaving of actions may just be a design decision
Modeling HTN Domains: Some Notes

- Find right balance of **which method constraints to add**
  - State space HTN planners do not profit a lot from *after* constraints
  - Missing important method preconditions (*before*) can lead to terrible performance
  - Sometimes, can **propagate action preconditions** up to method level

- **Total order of subtasks** whenever it makes sense
  - Takes away choices the planner would need to do
  - Allowing interleaving of actions may just be a design decision

- Make hierarchy **as shallow as possible**
  - Especially for SAT-based planners: iterative deepening!
    Deep hierarchies imply many iterations
Modeling HTN Domains: Some Notes

- Find right balance of which method constraints to add
  - State space HTN planners do not profit a lot from after constraints
  - Missing important method preconditions (before) can lead to terrible performance
  - Sometimes, can propagate action preconditions up to method level
- Total order of subtasks whenever it makes sense
  - Takes away choices the planner would need to do
  - Allowing interleaving of actions may just be a design decision
- Make hierarchy as shallow as possible
  - Especially for SAT-based planners: iterative deepening!
  - Deep hierarchies imply many iterations
- Branching factor of methods matters
  - How many methods per task to pick from?
  - What ratio of these methods actually leads to a plan?
Stay tuned!

Next lecture: Advanced SAT-based planning: classical planning and hierarchical planning via SAT

Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford, *Hddl-a language to describe hierarchical planning problems*.

