Lecture *Hierarchical Planning*

Chapter: *Introduction to HTN Planning*

Dr. Pascal Bercher

Institute of Artificial Intelligence,
Ulm University, Germany

Winter Term 2018/2019

(Compiled on: February 19, 2019)

ulm university universität uulm

Overview:

What is Planning?

- Concerning the *problem class*, so far, we considered only *classical planning* (and, as related work, also *various* extensions thereof).
- Thus: What is the *primary*[1] *goal* of planning?

  To find a sequence of actions that reaches some state in which the desired properties hold.

→ That's only the case for *non-hierarchical* planning and different from *hierarchical* planning!

[1] Please don't forget to check out the first lecture for an overview of some of the other interesting goals and research topics.

What is Hierarchical Planning?

"[Hierarchical] planners differ from classical planners in what they plan for and how they plan for it. In [a hierarchical] planner, the objective is not to achieve a set of goals but instead to perform some set of tasks."

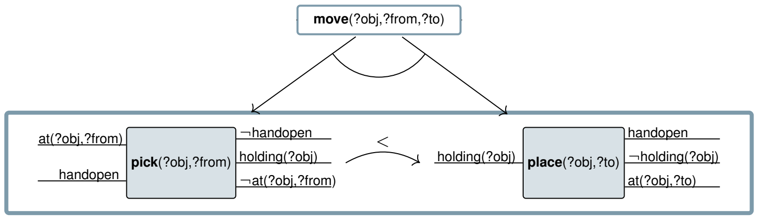
Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Ed. by Denise E. M. Penrose. Morgan Kaufmann, 2004

Main differences to classical planning problems:

- It's not about generating some goal state! The goal is find a refinement of the initial task(s), not to satisfy some goal description.
- There is no arbitrary task insertion: to alter task networks, we need to decompose compound tasks using their pre-defined methods (see next slide).

What is Hierarchical Planning? . . . More Precisely? Example?

The model specifies a ***task hierarchy***: *compound* (or *complex*, *abstract*, *high-level*) tasks need to be decomposed into *primitive tasks*.



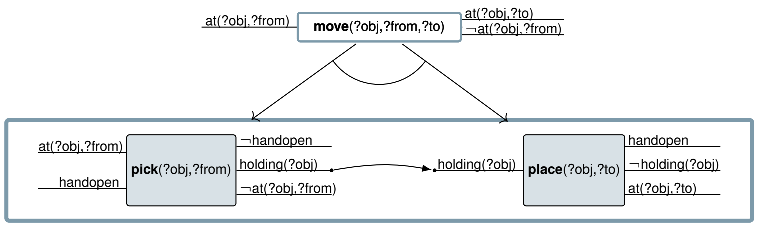
Goal: Find a (primitive) executable *refinement* of an initial hierarchical task network (HTN) or partial plan.

- Top: A compound task.
- Bottom: A task network.
- Together: A (decomposition) method.

} shown above

What is Hierarchical Planning? . . . More Precisely? Example?

The model specifies a ***task hierarchy***: *compound* (or *complex*, *abstract*, *high-level*) tasks need to be decomposed into *primitive tasks*.



Goal: Find a (primitive) executable *refinement* of an initial hierarchical task network (HTN) or partial plan.

- Top: A compound task (with precs/effs).
- Bottom: A partial (POCL) plan.
- Together: A (decomposition) method.

} shown above

Differences to Non-Hierarchical Planning

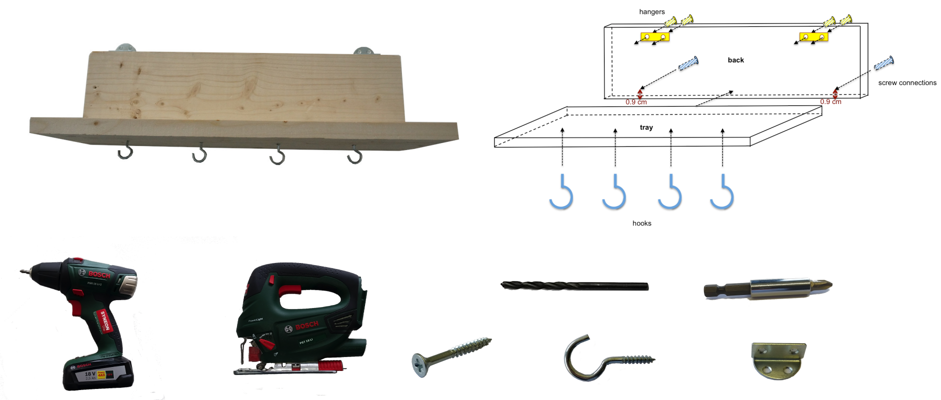
- Hierarchical Planning is often – *wrongly* – mistaken for a planning technique. This is *not* true. Its a different problem class with different properties.
- But of course we *also* need new/adapted planning techniques...
- *It's not about generating some goal state!* The goal is find a refinement of the initial compound task(s), *not* to satisfy some goal description.
- There is (normally) no arbitrary task insertion: To alter task networks/partial plans, we need to decompose compound tasks using their pre-defined methods. ("Task insertion" is an additional feature (actually: solution criteria!) that has to be provided/allowed in addition.)

Motivation

Why relying on a hierarchical model?

- More flexibility with regard to modeling approach: incorporate procedural expert knowledge (just as a modeling means, or to speed up search).
- Describe more complex behavior (i.e., pose complex restrictions on the desired solutions).
- Allow easier user integration in the plan generation process (mixed initiative planning; MIP).
- Use hierarchy as plan libraries (describing possible user intent) for plan recognition.
- Communicate plans on different levels of abstraction.
- Incorporate task abstraction in plan explanations.

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

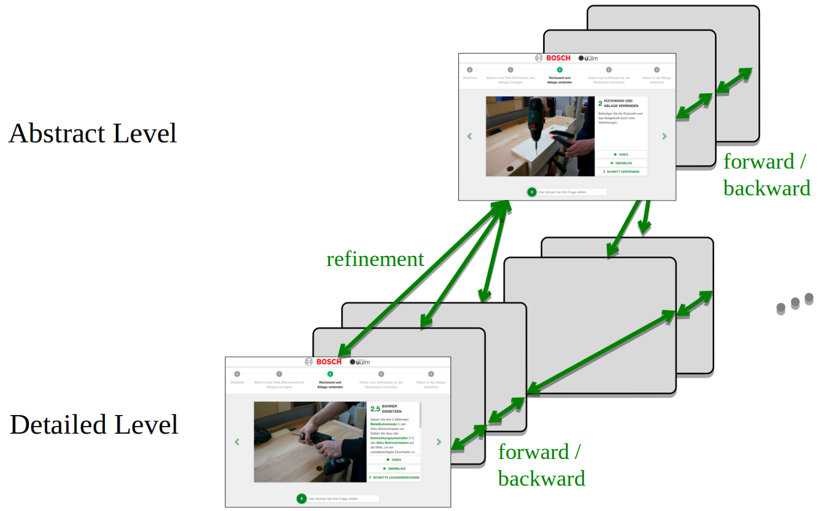


The material:

- 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, cond't

Presentation of instructions on different levels of abstraction:



Which Formalism?

- Hierarchical planning describes a *range* of hierarchical *problem classes* or *planning approaches* (solving techniques) that share the idea of *problem decomposition*.
- One of the best-known formalizations is called *hierarchical task network* (HTN) planning, so it is often used as a synonym to hierarchical planning although the latter can be regarded as the more general expression/field.
- Since the HTN formalism can be regarded a standard and the most simplistic one, we will start with that. Later, we extend it in several directions.

HTNs vs. HTN Problems vs. HTN Planning

- HTN is short for *(hierarchical) task network*.
  Thus, it is a *data structure*, not a formalism or an approach.
  → Never write/say something like: "In HTNs, we have to/aim at …".
  Correct would be: "In HTN planning, we …".
- The term "HTN planning" can still refer to either the problem class or an (HTN) planning approach (similar to classical planning).
- In the context of the HTN planning framework, we use *HTNs* as basic data structure, i.e., partially ordered tasks.
  If we also use causal links there, we refer to these data structures as *partial plans* instead.

Actions vs. Tasks

- We will use the terms *abstract*, *compound*, *complex*, and *high-level* tasks synonymously.
- *Actions* known from classical planning are the same as *primitive tasks* in hierarchical planning.
- In hierarchical planning, the term *task* is used to refer to either *actions* (i.e., primitive tasks) or *abstract tasks*.

Introduction

- In HTN planning (and all of its extensions that we will discuss in later lectures) we rely on the same basic assumptions as in classical planning (cf. first lecture).
- Just like in classical planning, problems are – in practice – *not* defined in a propositional way, but lifted.
- The basic formalism, is again defined in a *propositional* fashion.
- Note: While hierarchical planning, in principle, only extends non-hierarchical planning via a task hierarchy, we now also have some syntactical changes:
  - Rather than defining an action as 4-tuple $a = (pre, add, del, c)$ (and use a as its name, although not formally being defined), we have a designated set of *primitive task names* P, and a mapping δ to obtain their tuples (see next slide).
  - Rather than plan steps being 2-tuples $l{:}a$, we have another mapping α to map l to a (see later).

Literature

Most definitions (in particular: the ground HTN and TIHTN formalisms, and the decomposition tree) are taken from:

- Thomas Geier and Pascal Bercher. "On the Decidability of HTN Planning with Task Insertion". In: *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*. AAAI Press, 2011, pp. 1955–1961

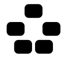
Definitions of the lifted HTN and TIHTN formalisms can be found in:

HTN Ron Alford, Pascal Bercher, and David Aha. "Tight Bounds for HTN Planning". In: *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS 2015)*. AAAI Press, 2015, pp. 7–15

TIHTN Ron Alford, Pascal Bercher, and David Aha. "Tight Bounds for HTN planning with Task Insertion". In: *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 2015, pp. 1502–1508

Problem Definition, Example Search Process

primitive tasks

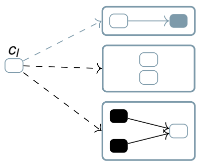


compound tasks

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

- V, a set of state variables.
- P, a set of primitive task names.
- $\delta : P \to (2^V)^3 \times \mathbb{R} \cup \{\infty\}$, the task name mapping.
- C, a set of compound task names.

Problem Definition, Example Search Process



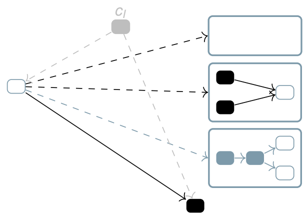
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$ with:

- V, a set of state variables.
- P, a set of primitive task names.
- $\delta : P \rightarrow (2^V)^3 \times \mathbb{R} \cup \{\infty\}$, the task name mapping.
- C, a set of compound task names.
- $c_I \in C$, the initial task.
- $M \subseteq C \times TN_{P \cup C}$, the (decomposition) methods.

A solution task network tn must:

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

Problem Definition, Example Search Process



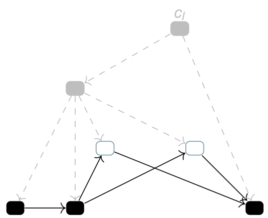
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$ with:

- V, a set of state variables.
- P, a set of primitive task names.
- $\delta : P \rightarrow (2^V)^3 \times \mathbb{R} \cup \{\infty\}$, the task name mapping.
- C, a set of compound task names.
- $c_I \in C$, the initial task.
- $M \subseteq C \times TN_{P \cup C}$, the (decomposition) methods.

A solution task network tn must:

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

Problem Definition, Example Search Process



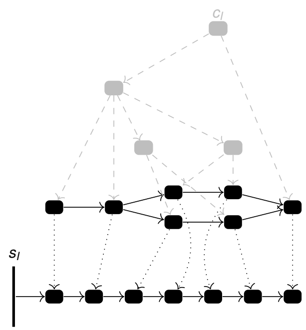
$\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$ with:

- V, a set of state variables.
- P, a set of primitive task names.
- $\delta : P \rightarrow (2^V)^3 \times \mathbb{R} \cup \{\infty\}$, the task name mapping.
- C, a set of compound task names.
- $c_I \in C$, the initial task.
- $M \subseteq C \times TN_{P \cup C}$, the (decomposition) methods.

A solution task network tn must:

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

Problem Definition, Example Search Process



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

- V, a set of state variables.
- P, a set of primitive task names.
- $\delta : P \rightarrow (2^V)^3 \times \mathbb{R} \cup \{\infty\}$, the task name mapping.
- C, a set of compound task names.
- $c_I \in C$, the initial task.
- $M \subseteq C \times TN_{P \cup C}$, the (decomposition) methods.
- $s_I \in 2^V$ the initial state.

A solution task network tn must:

- be a refinement of c_I,
- only contain primitive tasks, and
- have an executable linearization.

(Hierarchical) Task Networks

- A *task network* $tn = (T, \prec, \alpha)$ consists of:
  - T, a possibly empty set of *tasks* or *task identifier symbols*.
  - $\prec$, a strict partial order on the tasks.
  - $\alpha : T \to P \dot{\cup} C$, the task mapping function.

  Primitive task names are mapped to their tuples by the task name mapping $\delta : P \to (2^V)^3 \times \mathbb{R} \cup \{\infty\}$.
- Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are called *isomorphic* (written $tn \cong tn'$) if they differ solely in their task identifier symbols, i.e. there is a bijection $\sigma : T \to T'$ so that:
  - For all task identifiers $t \in T$ holds $\alpha(t) = \alpha'(\sigma(t))$.
  - For all task identifiers $t_1, t_2 \in T$ holds that $(t_1, t_2) \in \prec$ if and only if $(\sigma(t_1), \sigma(t_2)) \in \prec'$.
- A task network is called *executable* if it is primitive and there exists an executable linearization of its tasks (actions). Executability of action sequences is defined as usual.
- TN_X refers to the set of all task networks using only task names in X.

Decomposition Methods

- A *(decomposition) method* $m \in M$ is a tuple $m = (c, tn_m)$ with a compound task c and task network $tn_m = (T_m, \prec_m, \alpha_m)$.
- Let $tn = (T, \prec, \alpha)$ be a task network, $t \in T$ a task identifier, and $\alpha(t) = c$ a compound task to be decomposed by $m = (c, tn_m)$. We assume $T \cap T_m = \emptyset$.

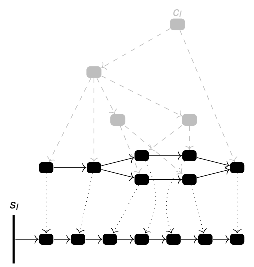
  Then, the application of m to tn results in the task network $tn' = ((T \setminus \{t\}) \cup T_m, \prec \cup \prec_m \cup \prec_X, \alpha \cup \alpha_m)|_{(T \setminus \{t\}) \cup T_m}$ with:

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

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

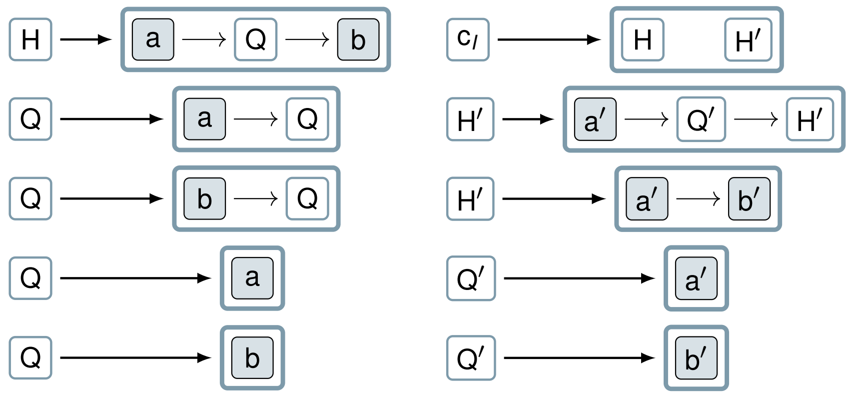
Solution Criteria

- A task network tn is a solution if and only if:
  - There is a sequence of decomposition methods $\overline{m}$ that transforms c_I into tn (written $tn_I \to^*_{TD} tn$, where tn_I denotes the initial task network consisting only of c_I) and
  - tn is executable, i.e.,
    - it contains only primitive tasks, and
    - the (still partially ordered) task network tn admits an executable linearization $\overline{t}$ of its tasks.



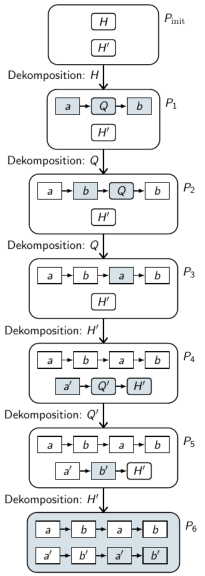
How to Represent Decomposition?

Consider the following decomposition methods:

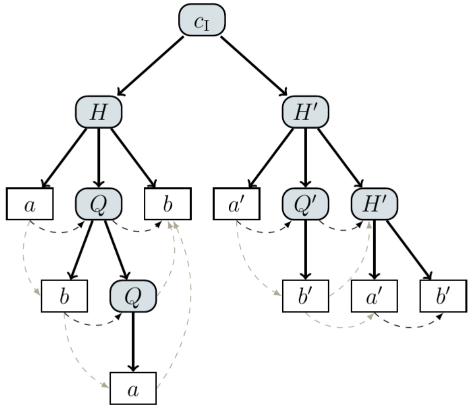


(Preconditions and effects don't matter for now.)

Decomposition Tree: Example



Representation as a tree:



Decomposition Tree: Definition

Definition (Decomposition Tree)

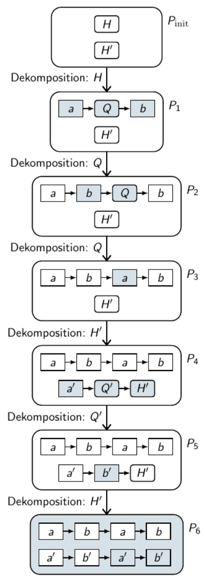
A *decomposition tree* $dt = (T, E, \prec, \alpha, \beta)$ is a five-tuple with the following properties:

- (T, E) is a tree with task identifier symbols T (the nodes of the tree) and directed edges $E \subseteq T \times T$ pointing towards the leafs,
- $\prec \subseteq T \times T$ is a strict partial order,
- $\alpha : T \to C \cup P$ is a task instance mapping that maps inner nodes to compound task names C and non-inner nodes to compound or primitive task names $C \cup P$, and
- $\beta : T' \to M \times Iso$, with $T' \subseteq T$, is a function mapping each node out of a (possibly strict) superset of the inner nodes to a tuple consisting of a method $m \in M$ and an isomorphism $\sigma \in Iso$, Iso denoting the set of all isomorphisms over the task instances in T.

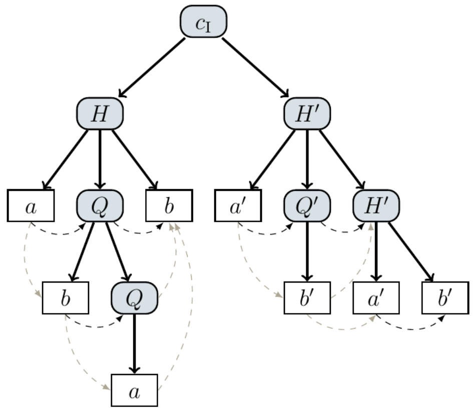
Decomposition Tree: Definition, cont'd

- We refer to the task instances T of dt by $T(dt)$ and
- to the direct children of $t \in T(dt)$ by $ch(dt, t)$.
- By $dt[t]$ we refer to the subtree of dt that is rooted in t.
- A task instance $t' \in T$ is called an ancestor of t if $t \in T(dt[t'])$.

Decomposition Tree: Example – Are We There Yet?



Representation as a tree:



Valid Decomposition Tree: Definition

Definition (Valid Decomposition Tree)

A decomposition tree $dt = (T, E, \prec, \alpha, \beta)$ is *valid* with respect to a planning problem $\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$ if and only if:

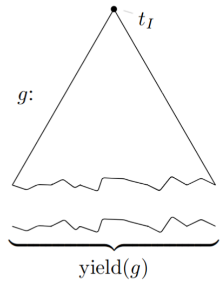
1. The root node of dt is labeled with the initial task name c_I.
2. If $t \in T$, $\alpha(t) = c$, and β is defined for t, then β maps t to a method $(c, tn_m) \in M$ and to an isomorphism σ that decomposes c, i.e., $\beta(t) = ((c, tn_m), \sigma)$, such that
   a. σ is an isomorphism for the children of t in dt and the tasks in tn_m. That is, let $(T', \prec', \alpha') \cong_\sigma tn_m$, then
      - $T' \subseteq T$ and $\{(t, t') \mid t' \in T'\} \subseteq E$
      - $(T', \prec', \alpha') = (ch(dt, t), \prec|_{ch(dt,t)}, \alpha|_{ch(dt,t)})$
   b. The ordering constraints imposed on t are correctly inherited. That is, for all $t' \in T$ and $t'' \in ch(dt, t)$ it holds that
      - if $(t, t') \in \prec$, then $(t'', t') \in \prec$
      - if $(t', t) \in \prec$, then $(t', t'') \in \prec$
3. There are no other ordering constraints than those demanded by Criterion 2 or those required by the definition of decomposition trees.

Yield of a Decomposition Tree

Definition (Yield of a Decomposition Tree)

The yield of a decomposition tree dt, *yield*(dt), is the following task network.

- Let $dt = (T, E, \prec, \alpha, \beta)$ and $T' \subseteq T$ be the set of all leaf nodes of dt for which β is not defined.
- Then, $yield(dt) := (T', \alpha|_{T'}, \prec|_{T'})$.



Properties of Decomposition Trees

Theorem

Given a planning problem $\mathcal{P}$, then for any task network $tn \in TN_{C \cup P}$ there exists a valid decomposition tree dt with $yield(dt) = tn$ if and only if $tn_I \rightarrow^*_{TD} tn$.

Proof:
Straight-forward.

Overview

Which formalization choices and extension to standard HTN planning do exist? Which impact do they have?

- Separation into problem and domain.
- Initial task network vs. a single initial task.
- Adding a goal description.
- Alternative definition of executability.
- Allowing to insert tasks.
- Adding state constraints.

Separating Between Domain and Problem

- So far, the problem was given as one single tuple $\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$.
- Similar to the planning domain description language (PDDL – see chapter on problem compilations), here we can also separate the problem into its *domain* and *problem (instance)*.
- Then, $\mathcal{D} = (V, P, \delta, C, M)$ is the *domain* and $\mathcal{P} = (\mathcal{D}, s_I, c_I)$ (or $\mathcal{P} = (\mathcal{D}, s_I, tn_I)$) is the *problem (instance)*.
- Then, the domain $\mathcal{D}$ describes the world's "physics", whereas the problem $\mathcal{P}$ describes the current task to solve.
- That way, we can also define several problems for the same domain.

Impact of Initial Task Network

Recap: $\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$ describes an HTN planning problem as described before.

Let $\mathcal{P}^\star = (V, P, \delta, C, M, s_I, \boldsymbol{tn_I})$ be an **HTN planning problem with initial task network $\boldsymbol{tn_I}$.**

Then, a task network tn is a solution if and only if:

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

Impact of Initial Task Network, cont'd

Theorem: **Initial task networks can be compiled away.**

Proof:

Let $\mathcal{P}^\star = (V, P, \delta, C, M, s_I, \boldsymbol{tn_I})$ be an HTN planning problem with initial task network tn_I.

Then, there is an HTN planning problem $\mathcal{P}' = (V, P, \delta, \boldsymbol{C'}, \boldsymbol{M'}, s_I, \boldsymbol{c_I})$ with the same set of solutions:

Let $C' := C \,\dot\cup\, \{c_I\}$ and $M' := M \cup \{(c_I, tn_I)\}$.

Identical solution set is obvious.

Impact of Goal Description

Recap: $\mathcal{P} = (V, P, \delta, C, M, s_I, c_I)$ describes an HTN planning problem as described before.

Let $\mathcal{P}^\star = (V, P, \delta, C, M, s_I, c_I, \boldsymbol{g})$ be an **HTN planning problem with goal description $\boldsymbol{g} \subseteq V$.**

Then, 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,
- the (still partially ordered) task network tn admits an executable linearization $\overline{t}$ of its tasks, and
- **the task sequence $\boldsymbol{\overline{t}}$ generates a goal state $\boldsymbol{s \supseteq g}$.**

Impact of Goal Description, cont'd

Theorem: **Goal descriptions can be compiled away.**

Proof:

Let $\mathcal{P}^\star = (V, P, \delta, C, M, s_I, c_I, \boldsymbol{g})$ be an HTN planning problem with goal description.

Then, there is an HTN planning problem $\mathcal{P}' = (V, \boldsymbol{P'}, \boldsymbol{\delta'}, C, M, s_I, \boldsymbol{tn_I})$ with the same set of solutions:

Here, tn_I contains two tasks: c_I followed by a new primitive task p with no effects and g as precondition, $\delta(p) = (g, \emptyset, \emptyset)$.

Then, the initial task network in $\mathcal{P}'$ can be compiled away as before.

Identical solution set is obvious.

Definition of Executability in HTN Planning

- So far, executability is defined as:
  There must *exist* an executabile linearization.
- What (happens and do we have to change) if we demand that *all* linearizations must be executable?

→ The altered (but non-standard) criterion is more practical, since it's the executable action sequence is, what we are usually interested in. "Finding" one from a solution is now trivial, otherwise hard (see later chapter).

→ Plan verification becomes easier (see later chapter).

→ For this criterion, we must allow ordering insertion, as otherwise solutions with the demanded properties might not exist.

Motivation

Benefits of allowing task insertion:

- Task insertion plus goal description fully subsumes classical planning (while allowing task hierarchies as well).
- Task insertion makes the modeling process easier: certain parts can be left to the planner.
- Task insertion makes the problem computationally easier (can be exploited for heuristics).

Problem Definition

In *HTN planning with task insertion*, *TIHTN planning*, tasks may be added arbitrarily to task networks (not just via decomposition):

Let $\mathcal{P}^\star = (V, P, \delta, C, M, s_I, c_I)$ be a **TIHTN planning problem**.

Then, a task network tn is a solution if and only if:

- There is a sequence of decomposition methods $\overline{m}$ **and task insertions** 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.

Problem Definition

In *HTN planning with task insertion*, *TIHTN planning*, tasks may be added arbitrarily to task networks (not just via decomposition):

Let $\mathcal{P}^\star = (V, P, \delta, C, M, s_I, c_I)$ be a **TIHTN planning problem**.

Then, 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 $\boldsymbol{tn'}$,
- $\boldsymbol{tn \supseteq tn'}$ **contains all tasks and orderings of** $\boldsymbol{tn'}$,
  (Note: allowing $\prec \supseteq \prec'$ would imply that we allow ordering insertion, which would, similar to HTN planning, be required if we demand all linearizations to be executable.)
- *tn* contains only primitive tasks, and
- the (still partially ordered) task network *tn* admits an executable linearization $\overline{t}$ of its tasks.

Problem Formalization

State constraints have been introduced in the HTN formalization by Erol et al. (1994):

- (l, t), the literal l holds immediately before task t.
- (t, l), the literal l holds immediately after task t.
- (t, l, t'), the literal l holds in all states between t and t'.

In case t, resp. t', are compound, a constraint (l, t) is, upon decomposition, translated to $(l, \mathit{first}[t_1, \ldots, t_n])$, where the t_i are all sub tasks of t. ((t, l) and (t, l, t') are handled analogously.)

Notably: Erol et al.'s formalization specifies a boolean constraint formula, in which *state*, *variable*, and *ordering constraints* can be specified with negations and disjunctions.

No compilation known yet.

Summary

- Hierarchical planning is *not* about generating a goal state (i.e., about finding a plan that generates a goal state) but about achieving a set of tasks.
- There are various different hierarchical planning formalisms (some of them covered later) with different theoretical properties.
- HTN planning is *the* standard hierarchical planning formalism.
- Also for HTN planning there are various formalization choices with differing impact on theoretical properties.