Hierarchical Planning: Limits, Extensions, and Model Repair

Pascal Bercher

School of Computing
College of Systems and Society
The Australian National University

18 December 2025

NATURAM PRIMUM COGNOSCERE RERUM

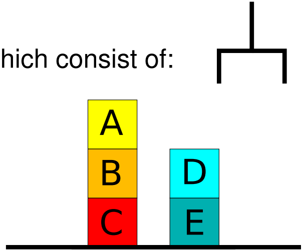
Australian
National
University

Planning Formalism

Classical Planning

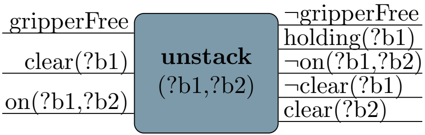
Let's start with *classical planning problems*, which consist of:

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



→ Find an action sequence (i.e., a *plan*) that transforms s_I into g.

For example, one of the available actions is:


gripperFree

clear(?b1)

on(?b1,?b2)

unstack (?b1,?b2)

¬gripperFree

holding(?b1)

¬on(?b1,?b2)

¬clear(?b1)

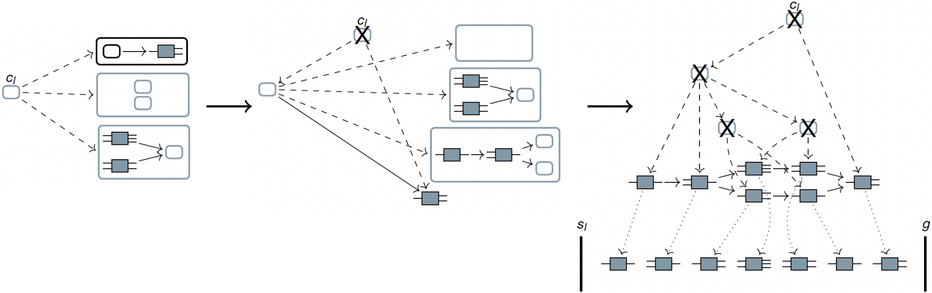
clear(?b2)

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

Hierarchical Task Network (HTN) Planning

In HTN Planning,

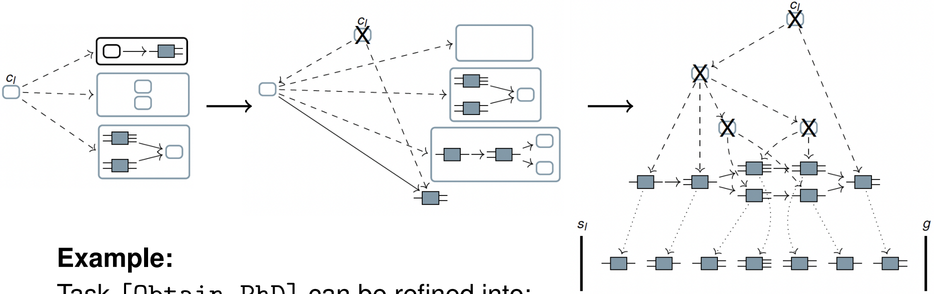
- we do not (only) plan for state-based variables; instead,
- we have initial *compound* tasks that need to be refined
  for which the model contains “methods”, the refinement rules.
- The solution is an executable, primitive task network (refinement).



Hierarchical Task Network (HTN) Planning

In HTN Planning,

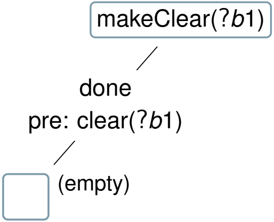
- we do not (only) plan for state-based variables; instead,
- we have initial *compound* tasks that need to be refined
  for which the model contains “methods”, the refinement rules.
- The solution is an executable, primitive task network (refinement).



Example:

Task `[Obtain PhD]` can be refined into:
`[Do Research]` → `[Write Thesis]` → `[Defend Thesis]`

HTN Planning by Example (Special Case: Totally Ordered)

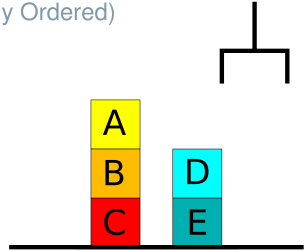


makeClear(?b1)

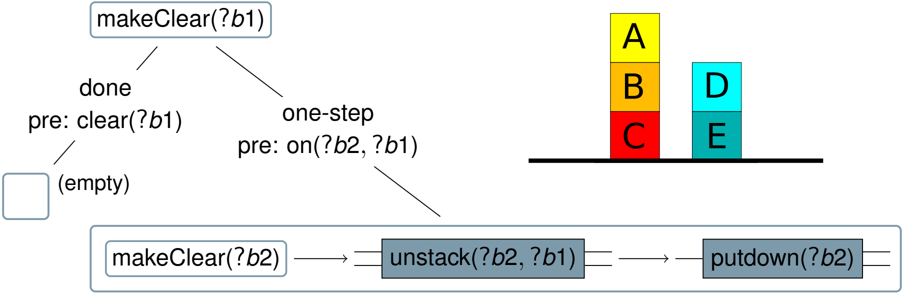
done

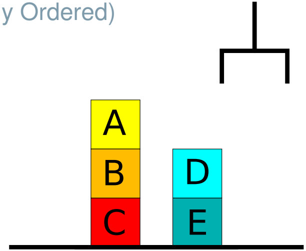
pre: clear(?b1)

(empty)

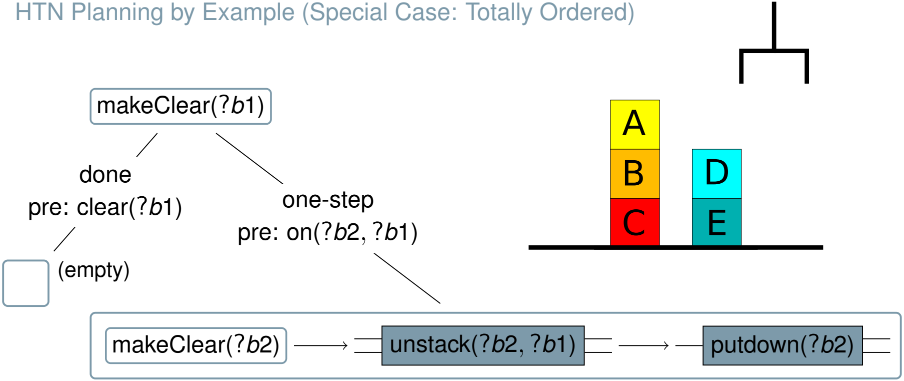


HTN Planning by Example (Special Case: Totally Ordered)



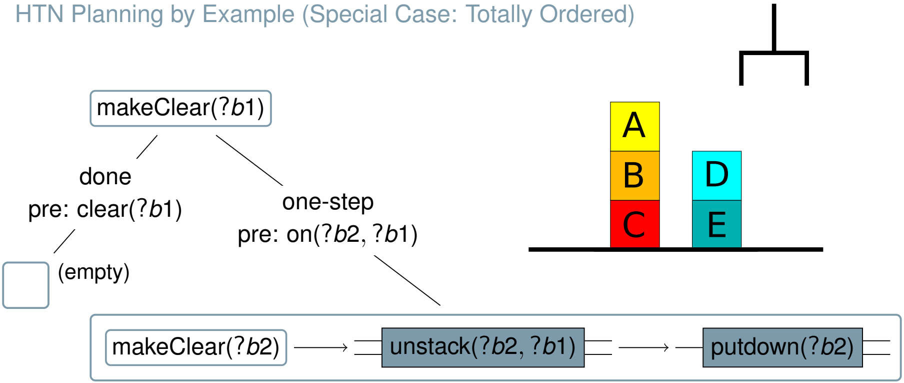


HTN Planning by Example (Special Case: Totally Ordered)



makeClear(C)

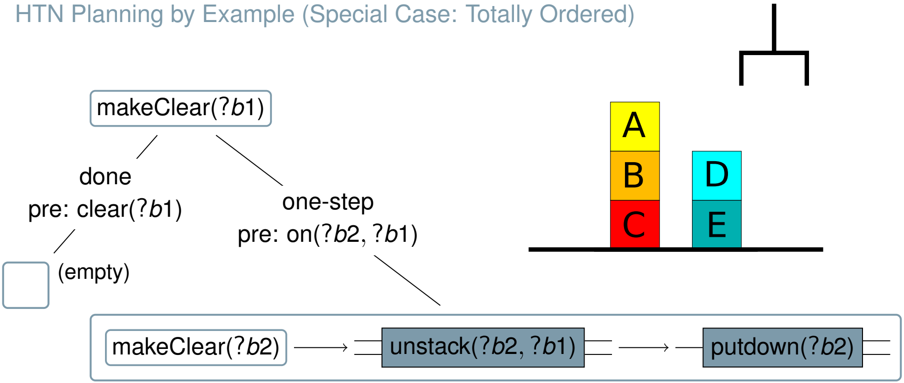
HTN Planning by Example (Special Case: Totally Ordered)



makeClear(C)

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

HTN Planning by Example (Special Case: Totally Ordered)



makeClear(?b1)

done
pre: clear(?b1)

(empty)

one-step
pre: on(?b2, ?b1)

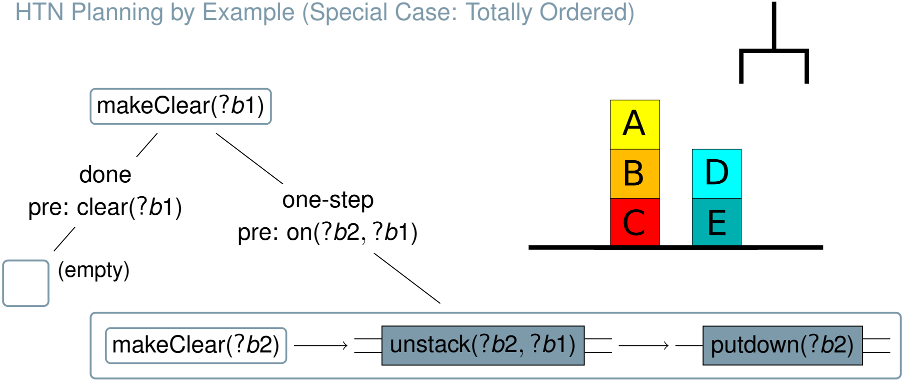
makeClear(?b2) → unstack(?b2, ?b1) → putdown(?b2)

makeClear(C)

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

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

HTN Planning by Example (Special Case: Totally Ordered)



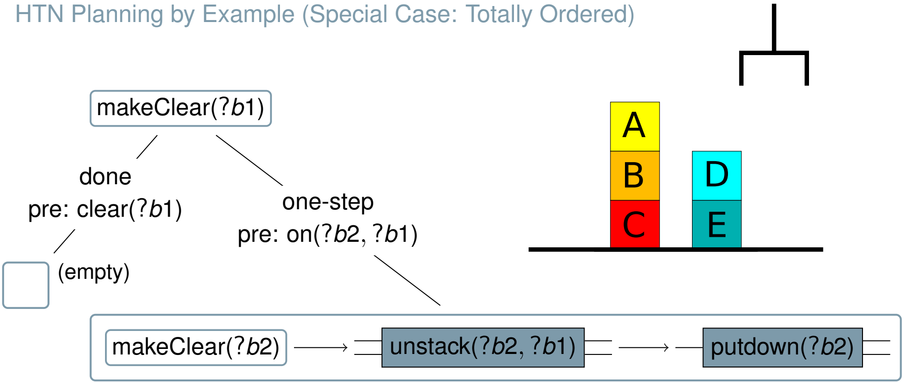
makeClear(C)

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

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

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

HTN Planning by Example (Special Case: Totally Ordered)



makeClear(C)

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

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

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

Note that in this model, we don’t need the predicates *gripperFree* and *holding(?b)* – since their “logic” is encoded into the task hierarchy.

Recap: What is HTN Planning?

Let’s provide a more theoretical (but “cleaner”) viewpoint.

- Let $\mathcal{P}$ be an HTN problem and let’s define:
  - $\mathit{Sol}_{\mathit{classic}}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in \mathit{sol}(\mathcal{P}')$, where $\mathcal{P}'$ is the induced classical problem $\}$, where $\mathit{sol}(\cdot)$ is the solution set of its classical planning problem

Recap: What is HTN Planning?

Let's provide a more theoretical (but "cleaner") viewpoint.

- Let $\mathcal{P}$ be an HTN problem and let's define:
  - $Sol_{classic}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in sol(\mathcal{P}')$, where $\mathcal{P}'$ is the induced classical problem $\}$, where $sol(\cdot)$ is the solution set of its classical planning problem
  - $Sol_{hierarchical}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in L(\mathcal{P}')$, where $\mathcal{P}'$ ignores all facts $\}$, where $L(\cdot)$ is the language of its formal grammar

Recap: What is HTN Planning?

Let's provide a more theoretical (but "cleaner") viewpoint.

- Let $\mathcal{P}$ be an HTN problem and let's define:
  - $Sol_{classic}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in sol(\mathcal{P}')$, where $\mathcal{P}'$ is the induced classical problem $\}$, where $sol(\cdot)$ is the solution set of its classical planning problem
  - $Sol_{hierarchical}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in L(\mathcal{P}')$, where $\mathcal{P}'$ ignores all facts $\}$, where $L(\cdot)$ is the language of its formal grammar
- Now, we can decompose the solution criteria:
  - $Sol_{classic}$ just looks at the executable action sequences that produce the goal,
  - $Sol_{hierarchical}$ just looks at the action sequences produced by the hierarchy.

Recap: What is HTN Planning?

Let's provide a more theoretical (but "cleaner") viewpoint.

- Let $\mathcal{P}$ be an HTN problem and let's define:
  - $Sol_{classic}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in sol(\mathcal{P}')$, where $\mathcal{P}'$ is the induced classical problem $\}$, where $sol(\cdot)$ is the solution set of its classical planning problem
  - $Sol_{hierarchical}(\mathcal{P}) = \{\bar{a} \mid \bar{a} \in L(\mathcal{P}')$, where $\mathcal{P}'$ ignores all facts $\}$, where $L(\cdot)$ is the language of its formal grammar
- Now, we can decompose the solution criteria:
  - $Sol_{classic}$ just looks at the executable action sequences that produce the goal,
  - $Sol_{hierarchical}$ just looks at the action sequences produced by the hierarchy.

→ $Sol(\mathcal{P}) = Sol_{classic}(\mathcal{P}) \cap Sol_{hierarchical}(\mathcal{P})$.

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

HTN planning = classical planning + grammar to filter solutions

Outline

Research Expertise and Interests

My Research Directions:

(1) Algorithm and heuristic design (solve problems quickly)

(2) Computational Complexity Analyses

times (cross product)

(a) Solving (mostly hierarchical) planning problems

(b) Repairing flawed (both classical and hierarchical) models

Research Expertise and Interests

My Research Directions:

(1) Algorithm and heuristic design (solve problems quickly)

~~(2) Computational Complexity Analyses~~

times (cross product)

(a) Solving (mostly hierarchical) planning problems

(b) Repairing flawed (both classical and hierarchical) models

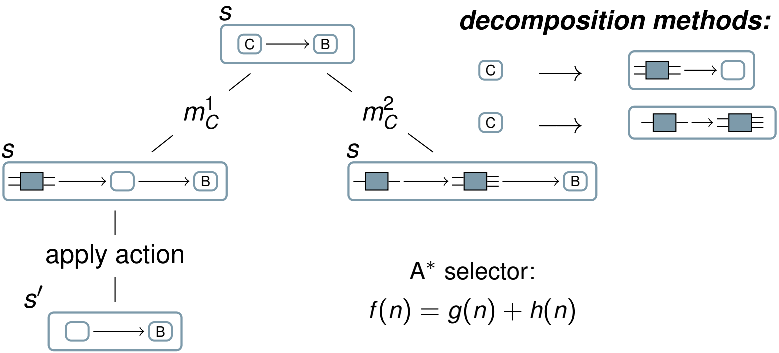
Today / Talk Outline:

- What's HTN planning? (Check!)
- Solving HTN problems via A* search.
- Extension of HTNs to uncertainty.
- Model Repair: some overview.

Progression Search to Solve HTN Problems

HTN Progression Search via A^*

- **node selection:** Select a node with minimal f value.
  - g: cost incurred so far (number of progressed/applied actions)
  - h: estimate of number of actions to still be applied
- **node expansion:**
  - primitive? Progress it! (Update the state.)
  - compound? Apply all its decomposition methods!
- **solution:** The action sequences encoded in the search path.

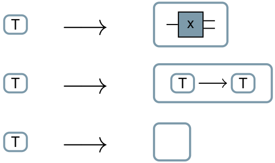


A^* selector:

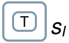
$$f(n) = g(n) + h(n)$$

A^* is Incomplete: We have infinitely many nodes with perfect f-value!

decomposition methods:

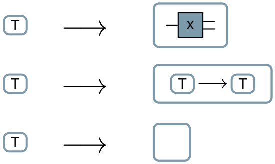


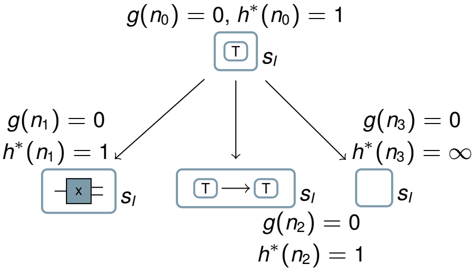
$g(n_0) = 0$, $h^*(n_0) = 1$



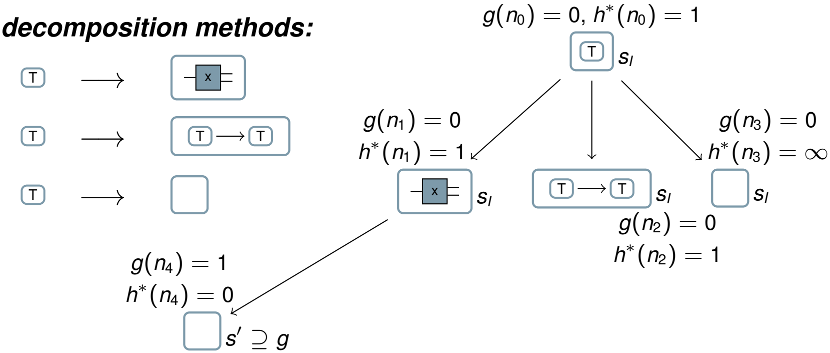
A^* is Incomplete: We have infinitely many nodes with perfect f-value!

decomposition methods:

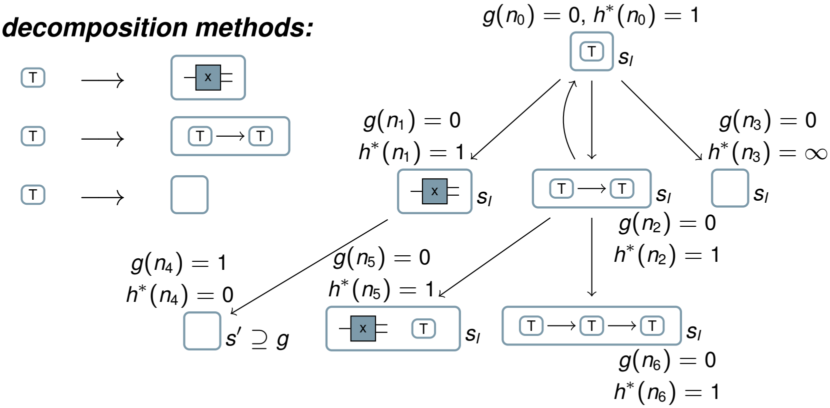




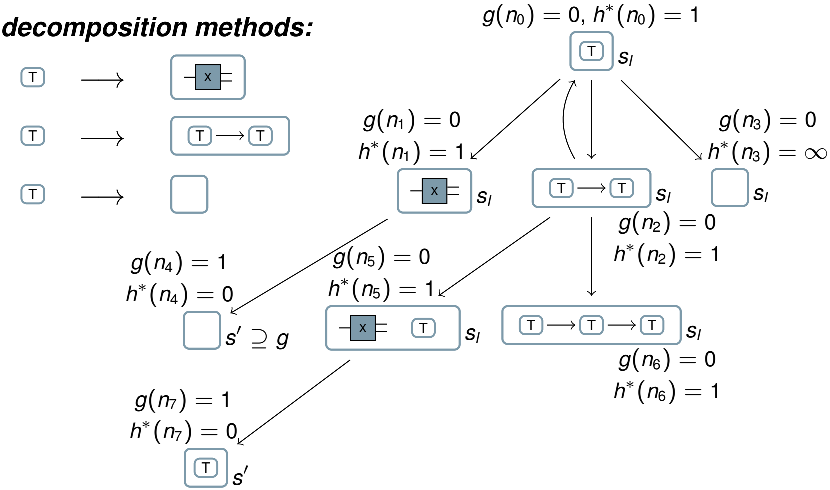
A^* is Incomplete: We have infinitely many nodes with perfect f-value!



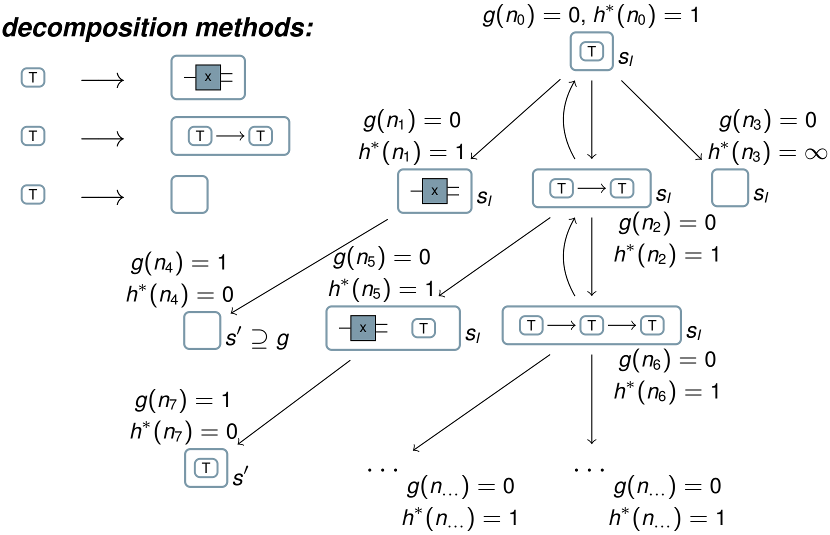
A^* is Incomplete: We have infinitely many nodes with perfect f-value!



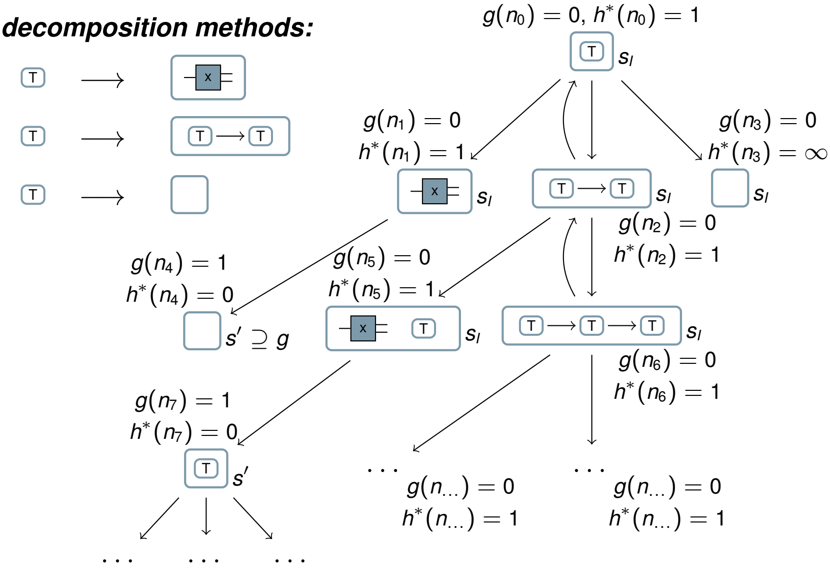
A^* is Incomplete: We have infinitely many nodes with perfect f-value!



A^* is Incomplete: We have infinitely many nodes with perfect f-value!



A^* is Incomplete: We have infinitely many nodes with perfect f-value!



Key Messages

So, what about progression search?

- Progression search is a state-of-the-art approach.
  (There are several heuristics and planners by multiple teams.)

(I have several works on HTN progression)

Key Messages

So, what about progression search?

- Progression search is a state-of-the-art approach.
  (There are several heuristics and planners by multiple teams.)
- However, even with *perfect heuristic* and total-order HTN problems, and if there’s a solution, search might get stuck in an infinite loop!
- The issue can be solved by problem compilations.

(I have several works on HTN progression)
(This particular result is from ICAPS’25)

Key Messages

So, what about progression search?

- Progression search is a state-of-the-art approach.
  (There are several heuristics and planners by multiple teams.)
- However, even with *perfect heuristic* and total-order HTN problems, and if there's a solution, search might get stuck in an infinite loop!
- The issue can be solved by problem compilations.

What about other algorithms?

- There's also planning as SAT: **see tomorrow's PhD defense!**
- Currently, that is the only approach that guarantees termination!

(I have several works on HTN progression)
(This particular result is from ICAPS'25)

FOND HTN Planning

Deterministic vs. Non-Deterministic HTN Planning

The solution criteria of HTN planning so far, informally:

Given an initial compound task,

- find a refinement task network (according to methods/rules),
- such that it does possess an executable action linearization.

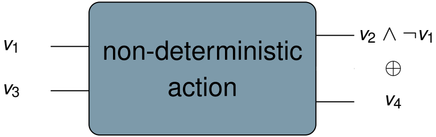
Deterministic vs. Non-Deterministic HTN Planning

The solution criteria of HTN planning so far, informally:

Given an initial compound task,

- find a refinement task network (according to methods/rules),
- such that it does possess an executable action linearization.

Now, we have actions with effect uncertainty!



Two pairs of effects:

- **Either** add v_2 and delete v_1,
- **or** add v_4

We don't know in advance which effect will happen (until observed)

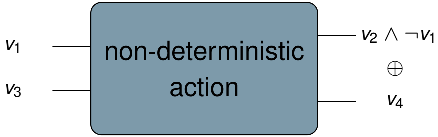
Deterministic vs. Non-Deterministic HTN Planning

The solution criteria of HTN planning so far, informally:

Given an initial compound task,

- find a refinement task network (according to methods/rules),
- such that it does possess an executable action linearization.

Now, we have actions with effect uncertainty!



v_1

v_3

non-deterministic action

$v_2 \wedge \neg v_1$

$\oplus$

v_4

Two pairs of effects:

- **Either** add v_2 and delete v_1,
- **or** add v_4

We don't know in advance which effect will happen (until observed)

New solution criteria:

- Find a refinement task network (as before!),
- such that *we can execute it no matter what happens*.

Different Solution Concepts

We developed two main categories of solution concepts:

1. Solutions are still (partially ordered) task networks (we called them fixed-method policies)
   - without likelihoods; plans “have to work”, ICAPS’21
   - with probability distribution, KR’25

Different Solution Concepts

We developed two main categories of solution concepts:

1. Solutions are still (partially ordered) task networks (we called them fixed-method policies)
   - without likelihoods; plans “have to work”, ICAPS’21
   - with probability distribution, KR’25
2. Solutions are policies that can pick methods during execution (we called them method-based policies, ICAPS’22)

Different Solution Concepts

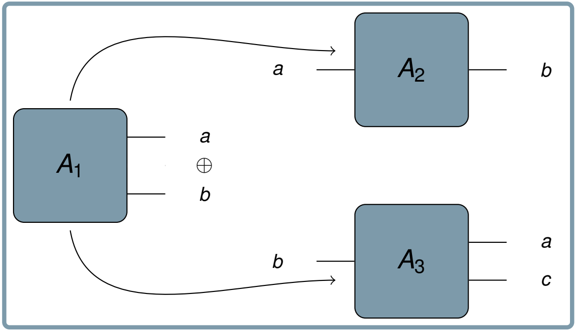
We developed two main categories of solution concepts:

1. Solutions are still (partially ordered) task networks (we called them fixed-method policies)
   - without likelihoods; plans “have to work”, ICAPS’21
   - with probability distribution, KR’25
2. Solutions are policies that can pick methods during execution (we called them method-based policies, ICAPS’22)

We now give short examples, current developments, and some interesting properties for each!

Fixed-Method Solutions: Strong Solutions

Assume we found this plan:

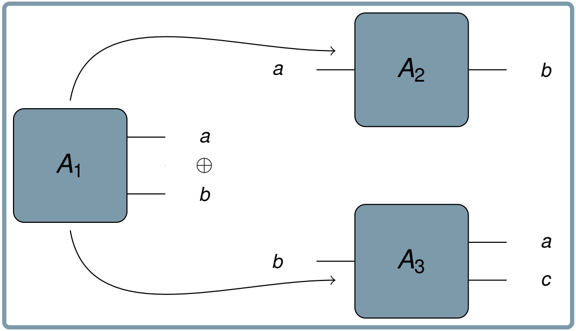


(To turn the empty initial state into one that makes c true.)

Does it do its job?

Fixed-Method Solutions: Strong Solutions

Assume we found this plan:



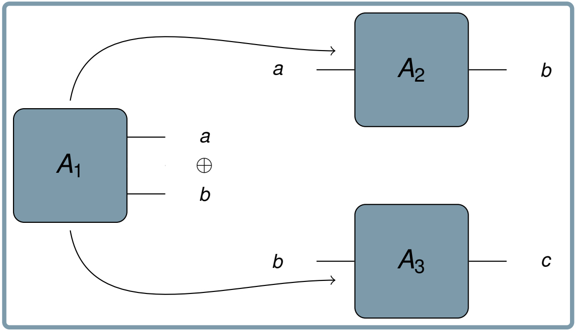
(To turn the empty initial state into one that makes c true.)

Does it do its job? **Yes!** *With a policy:*

- If A_1 produces a, then execute A_2 next (and then A_3)
- If A_1 produces b, then execute A_3 next (and then A_2)

Fixed-Method Solutions: Weak Solutions

Assume we found *this* plan:

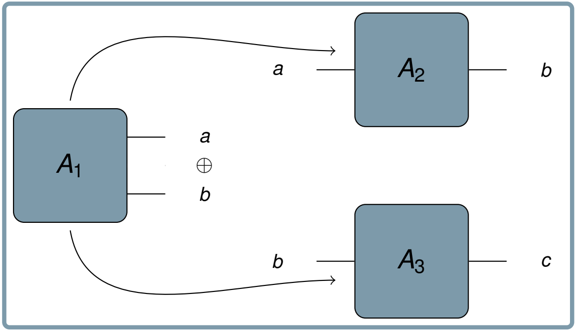


(To turn the empty initial state into one that makes c true.)

What about now?

Fixed-Method Solutions: Weak Solutions

Assume we found *this* plan:



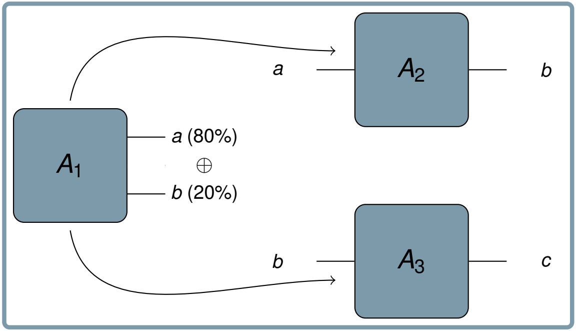
(To turn the empty initial state into one that makes c true.)

What about now? *Kinda.* But only one branch succeeds:

- If A_1 produces a, then execute A_2 next (and then A_3)
- If A_1 produces b, A_2 can't be executed, but must! :(

Fixed-Method Solutions: What if we have effect likelihoods?

Assume we found *this* plan:

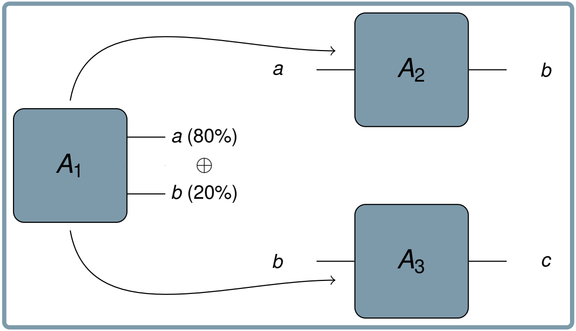


(To turn the empty initial state into one that makes c true.)

What about now?

Fixed-Method Solutions: What if we have effect likelihoods?

Assume we found *this* plan:



(To turn the empty initial state into one that makes c true.)

What about now? *Yes,* and we can quantify its success probability:

- It succeeds with 80% likelihood (and fails with 20%).
- Thus, we can ask for a plan that succeeds with $\mathbb{P} \geq p$

Fixed-Method Solutions: Overview

Trivia / work so far:

- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’21
  - Sadly, *no planner exists yet!*
- Extension to effect likelihoods:
  - Formalism and complexity results (again, “plan existence”), KR’25
  - Again, no planner, yet!

Fixed-Method Solutions: Overview

Trivia / work so far:

- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’21
  - Sadly, *no planner exists yet!*
- Extension to effect likelihoods:
  - Formalism and complexity results (again, “plan existence”), KR’25
  - Again, no planner, yet!

Some pro’s and con’s:

- **Pro:** Solution concept is very close to deterministic HTNs!
- **Con:** Extremely limited “repair” capabilities! Why?

Fixed-Method Solutions: Overview

Trivia / work so far:

- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’21
  - Sadly, *no planner exists yet!*
- Extension to effect likelihoods:
  - Formalism and complexity results (again, “plan existence”), KR’25
  - Again, no planner, yet!

Some pro’s and con’s:

- **Pro:** Solution concept is very close to deterministic HTNs!
- **Con:** Extremely limited “repair” capabilities! Why?
  - Because solutions are fixed sets of actions!
  - Prevents solutions such as *“(re)submit paper until accepted”* :(

Fixed-Method Solutions: Overview

Trivia / work so far:

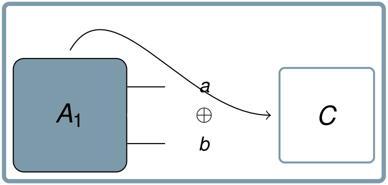
- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’21
  - Sadly, *no planner exists yet!*
- Extension to effect likelihoods:
  - Formalism and complexity results (again, “plan existence”), KR’25
  - Again, no planner, yet!

Some pro’s and con’s:

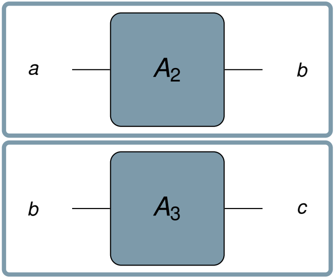
- **Pro:** Solution concept is very close to deterministic HTNs!
- **Con:** Extremely limited “repair” capabilities! Why?
  - Because solutions are fixed sets of actions!
  - Prevents solutions such as *“(re)submit paper until accepted”* :(
  - Formalism with likelihoods remains undecidable even under *significant restrictions*: total order + tail-recursiveness

Method-based Solutions: Limitations of Fixed-method Solutions

Initial task network:



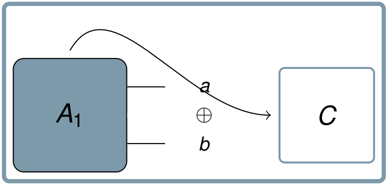
Decomposition methods for C:



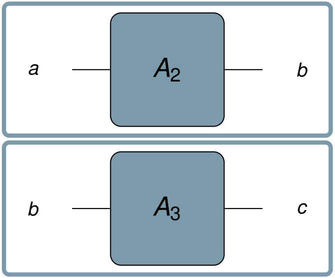
Is there a fixed-method solution?

Method-based Solutions: Limitations of Fixed-method Solutions

Initial task network:



Decomposition methods for C:



Is there a fixed-method solution?

- No, because no single primitive plan always works!
- But there is if we can delay the method choice!

→ Solution definition is a policy $\pi : s, tn \mapsto [a \text{ or } (c, m)]$

Method-based Solutions: Overview

Trivia / work so far:

- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’22
  - One planner, grounder, and heuristics, IJCAI’24
- Extension to effect likelihoods:
  - Nothing, yet

Method-based Solutions: Overview

Trivia / work so far:

- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’22
  - One planner, grounder, and heuristics, IJCAI’24
- Extension to effect likelihoods:
  - Nothing, yet

Some pro’s and con’s:

- **Pro:** We finally can: *“(re)submit until accepted”*! :)
  That’s since a policy can recursively re-invoke a compound task.

Method-based Solutions: Overview

Trivia / work so far:

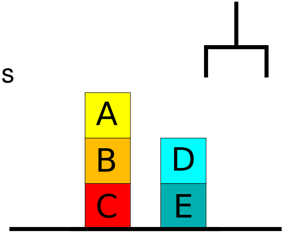
- For non-deterministic effects:
  - Formalism and complexity results (“plan existence”), ICAPS’22
  - One planner, grounder, and heuristics, IJCAI’24
- Extension to effect likelihoods:
  - Nothing, yet

Some pro’s and con’s:

- **Pro:** We finally can: *“(re)submit until accepted”*! :)
  That’s since a policy can recursively re-invoke a compound task.
- **Con:** Computationally even harder than fixed-method ones.

Modeling Support

Blocksworld Revisited



We want automated support in creating actions (and their interactions) like the one below:

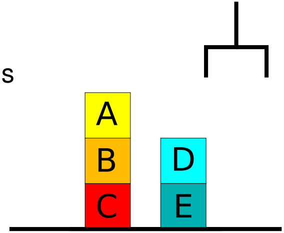
Warm-up: ***Is the action correctly modeled?***

```
(:action unstack
   :parameters (?b1 ?b2 - block)
   :precondition (and (gripperFree)
                      (on ?b1 ?b2) (clear ?b1))
   :effect (and (not (gripperFree)) (holding ?b1)
                (not (on ?b1 ?b2)) (not (clear ?b1))
                (clear ?b2)))
```

Blocksworld Revisited

We want automated support in creating actions (and their interactions) like the one below:

Warm-up: ***Is the action correctly modeled?***



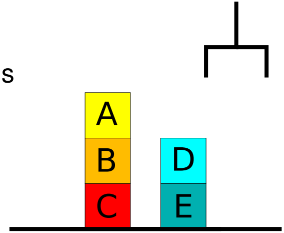
```
(:action unstack
   :parameters (?b1 ?b2 - block)
   :precondition (and (gripperFree)
                      (on ?b1 ?b2) (clear ?b1))
   :effect (and (not (gripperFree)) (holding ?b1)
                (not (on ?b1 ?b2)) (not (clear ?b1))
                (clear ?b2)))
```

Yes! We had that exact action on slide 1! :)

Blocksworld Revisited

We want automated support in creating actions (and their interactions) like the one below:

Warm-up: ***Is the action correctly modeled?***

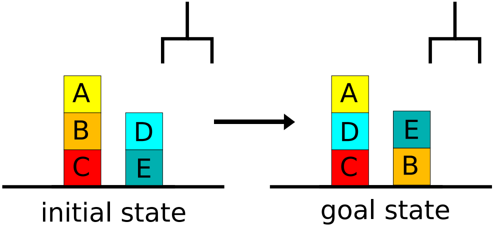


```
(:action unstack
   :parameters (?b1 ?b2 – block)
   :precondition (and (gripperFree)
                      (on ?b1 ?b2) (clear ?b1))
   :effect (and (not (gripperFree)) (holding ?b1)
                (not (on ?b1 ?b2)) (not (clear ?b1))
                (clear ?b2)))
```

Yes! We had that exact action on slide 1! :) Though one can argue: Did we forget to demand ?b1≠?b2? (Not required here.)

Patient Zero

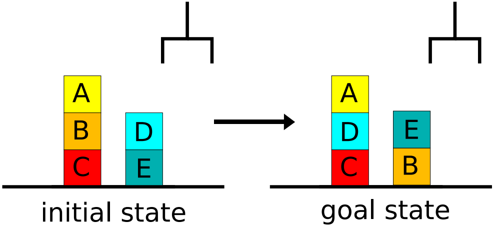
How about this situation? Is it modeled correctly?



```
(define (problem blocksworld-prob)
  (:domain blocksworld)
  (:init (clear A) (on A B) (on B C) (onTable C)
         (clear D) (on D E) (onTable E))
  (:goal (and (clear A) (on A D) (on D C) (onTable C)
              (clear E) (on E B) (onTable B))))
```

Patient Zero

How about this situation? Is it modeled correctly?



```
(define (problem blocksworld-prob)
  (:domain blocksworld)
  (:init (clear A) (on A B) (on B C) (onTable C)
         (clear D) (on D E) (onTable E))
  (:goal (and (clear A) (on A D) (on D C) (onTable C)
              (clear E) (on E B) (onTable B))))
```

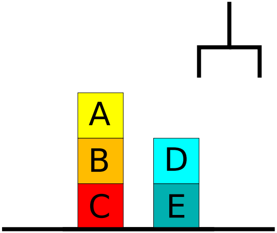
No! The gripper being initially empty is missing!

Okay, modeling is hard, so...

How do provide modeling support?

So far, our proposal was to provide a set of test plans:

- Some are supposed to be solutions (but might not), *whitelist plans*
- others should not be solutions (but might). *blacklist plans*



Example:

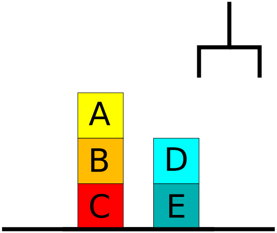
- "unstack(A,B) putdown(A) unstack(B,C) putdown(B) is executable"
- "unstack(A,B) putdown(A) unstack(B,C) unstack(D,E) is *not* executable"

Okay, modeling is hard, so...

How do provide modeling support?

So far, our proposal was to provide a set of test plans:

- Some are supposed to be solutions (but might not), *whitelist plans*
- others should not be solutions (but might). *blacklist plans*



Example:

- "unstack(A,B) putdown(A) unstack(B,C) putdown(B) is executable"
- "unstack(A,B) putdown(A) unstack(B,C) unstack(D,E) is *not* executable"

What's a (good) repair?

- A set of precondition or effect additions or removals.
- Minimal number of repairs, or getting assessed by LLMs.

Overview

Trivia / work so far:

- Complexity investigations:
  - for classical planning (i.e., change preconditions/effects): almost always NP-complete (IJCAI 2021)
  - for HTN planning (i.e., add/delete actions to/from methods) also almost always NP-complete (IJCAI 2021); complexity for black + whitelist plans: [NP-hard,in Σ_2^p] (AAAI'23)

Overview

Trivia / work so far:

- Complexity investigations:
  - for classical planning (i.e., change preconditions/effects): almost always NP-complete (IJCAI 2021)
  - for HTN planning (i.e., add/delete actions to/from methods) also almost always NP-complete (IJCAI 2021); complexity for black + whitelist plans: [NP-hard,in Σ_2^p] (AAAI'23)
- (My) Repair algorithms for classical planning:
  - all use a compilation to hitting sets (an NP-complete problem):
    - Just whitelist plans (AAAI'23b)
    - White + blacklist plans (AAAI'25)
    - Lifted input plans (ECAI'25)
    - Ideas how to integrate with LLMs (HAXP workshop 2025)

Overview

Trivia / work so far:

- Complexity investigations:
  - for classical planning (i.e., change preconditions/effects): almost always NP-complete (IJCAI 2021)
  - for HTN planning (i.e., add/delete actions to/from methods) also almost always NP-complete (IJCAI 2021); complexity for black + whitelist plans: [NP-hard,in Σ_2^p] (AAAI'23)
- (My) Repair algorithms for classical planning:
  - all use a compilation to hitting sets (an NP-complete problem):
    - Just whitelist plans (AAAI'23b)
    - White + blacklist plans (AAAI'25)
    - Lifted input plans (ECAI'25)
    - Ideas how to integrate with LLMs (HAXP workshop 2025)
- (My, and all(?)) Repair algorithms for HTN planning:
  - Compilation into another HTN problem (SoCS 2024)
  - Just ask an LLM to repair the problem (AAAI 2026)

Summary

Research Expertise and Interests

My Research Directions:

(1) Algorithm and heuristic design (solve problems quickly)

(2) Computational Complexity Analyses

times (cross product)

(a) Solving (mostly hierarchical) planning problems

(b) Repairing flawed (both classical and hierarchical) models

Today / Talk Outline:

- What's HTN planning? (Check!)
- Solving HTN problems via A* search.
- Extension of HTNs to uncertainty.
- Model Repair: some overview.

Thank you
for listening!