Artificial Intelligence Foundation – JC3001

Lecture 32: Hierarchical Planning - II

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

October 2025

Material adapted from:
Russell and Norvig (AIMA Book): Chapter 11 (11.4)
Dana Nau (University of Maryland)

UNIVERSITY OF ABERDEEN

Course Progression

- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. Search 2: Heuristic Search ✓
  3. Search 3: Local Search
  4. Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
  1. Reasoning 1: Constraint Satisfaction ✓
  2. Reasoning 2: Logic and Inference ✓
  3. Probabilistic Reasoning 1: BNs ✓
  4. Probabilistic Reasoning 2: HMMs ✓
- Part 4: Planning
  1. Planning 1: Intro and Formalism ✓
  2. Planning 2: Algorithms & Heuristics ✓
  3. **Planning 3: Hierarchical Planning**
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

Objectives

- Control Knowledge in Planning
- Hierarchical Planning

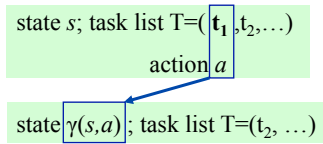
Outline

Solving Total-Order STN Planning Problems

1: **function** TFD(s, $\langle t_1, \ldots, t_k \rangle$, O, M)
2: **if** $k = 0$ **then return** $\langle\rangle$ ▷ *(i.e. the empty plan)*
3: **if** t_1 is primitive **then**
4: $active \leftarrow \{\langle a, \sigma \rangle | a \in O, a = t_1 \cdot \sigma,$ and $s \models pre(a)\}$
▷ *Match applicable ground instances of actions to the task*
5: **if** $active = \emptyset$ **then return** $failure$
6: nondeterministically choose any $\langle a, \sigma \rangle \in$ `active`
7: $\pi \leftarrow$ TFD($\gamma(s, a), \langle t_2, \ldots, t_k \rangle \sigma, O, M$)
8: **if** $\pi = failure$ **then return** $failure$
9: **else return** $a \cdot \pi$
10: **else if** t_1 is nonprimitive **then**
11: $active \leftarrow \{(m, \sigma) | m \in M, m = t_1 \sigma,$ and $s \models pre(m)\}$
▷ *Match applicable ground instances of methods to the task*
12: **if** $active = \emptyset$ **then return** $failure$
13: nondeterministically choose any $\langle m, \sigma \rangle \in active$
14: $w = tn(m) \cdot \langle t_2, \ldots, t_k \rangle \sigma$
15: **return** TFD(s, w, O, M)

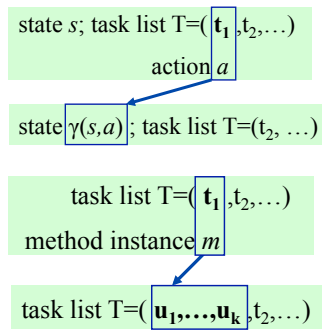
Solving Total-Order STN Planning Problems

1: **function** TFD(s, $\langle t_1, \ldots, t_k \rangle$, O, M)
2: **if** $k = 0$ **then return** $\langle\rangle$ ▷ *(i.e. the empty plan)*
3: **if** t_1 is primitive **then**
4: $active \leftarrow \{\langle a, \sigma \rangle | a \in O, a = t_1 \cdot \sigma,$ and $s \models pre(a)\}$
▷ *Match applicable ground instances of actions to the task*
5: **if** $active = \emptyset$ **then return** *failure*
6: nondeterministically choose any $\langle a, \sigma \rangle \in$ `active`
7: $\pi \leftarrow$ TFD($\gamma(s, a), \langle t_2, \ldots, t_k \rangle \sigma, O, M$)
8: **if** $\pi = failure$ **then return** *failure*
9: **else return** $a \cdot \pi$
10: **else if** t_1 is nonprimitive **then**
11: $active \leftarrow \{(m, \sigma) | m \in M, m = t_1 \sigma,$ and $s \models pre(m)\}$
▷ *Match applicable ground instances of methods to the task*
12: **if** $active = \emptyset$ **then return** *failure*
13: nondeterministically choose any $\langle m, \sigma \rangle \in active$
14: $w = tn(m) \cdot \langle t_2, \ldots, t_k \rangle \sigma$
15: **return** TFD(s, w, O, M)



Solving Total-Order STN Planning Problems

1: **function** TFD(s, $\langle t_1, \ldots, t_k \rangle$, O, M)
2: **if** $k = 0$ **then return** $\langle\rangle$ ▷ *(i.e. the empty plan)*
3: **if** t_1 is primitive **then**
4: $active \leftarrow \{\langle a, \sigma \rangle | a \in O, a = t_1 \cdot \sigma,$ and $s \models pre(a)\}$
▷ *Match applicable ground instances of actions to the task*
5: **if** $active = \emptyset$ **then return** *failure*
6: nondeterministically choose any $\langle a, \sigma \rangle \in$ `active`
7: $\pi \leftarrow$ TFD($\gamma(s, a), \langle t_2, \ldots, t_k \rangle \sigma, O, M$)
8: **if** $\pi = failure$ **then return** *failure*
9: **else return** $a \cdot \pi$
10: **else if** t_1 is nonprimitive **then**
11: $active \leftarrow \{(m, \sigma) | m \in M, m = t_1 \sigma,$ and $s \models pre(m)\}$
▷ *Match applicable ground instances of methods to the task*
12: **if** $active = \emptyset$ **then return** *failure*
13: nondeterministically choose any $\langle m, \sigma \rangle \in active$
14: $w = tn(m) \cdot \langle t_2, \ldots, t_k \rangle \sigma$
15: **return** TFD(s, w, O, M)



Domain (HDDL)

1 Simple Task Network (STN) Planning

```
(:action getTicket
   :parameters (?x ?y ?xc ?yc — object)
   :precondition (and (airport ?x ?xc)
      (airport ?y ?yc))
   :effect (and (ticket ?x ?y))
)

(:action fly
   :parameters (?x ?y ?x1 ?y1 — object)
   :precondition (and (airport ?x ?x1)
    (airport ?y ?y1) (ticket ?x ?y) (at ?x))
   :effect (and (not (at ?x)) (not (ticket ?x ?y))
           (at ?y))
)

(:action getTaxi
   :parameters (?x — object)
   :precondition (and (at ?x))
   :effect (and (hasTaxi ?x))
)

(:action rideTaxi
   :parameters (?x ?y — object)
   :precondition (and (at ?x)
      (not (longDistance ?x ?y)) (hasTaxi ?x))
   :effect (and (not (at ?x)) (not (hasTaxi ?x))
      (at ?y))
)
```

```
(:method travel—by—taxi
   :parameters (?x ?y — object)
   :task (travel ?x ?y)
   :precondition (and (at ?x)
       (not (longDistance ?x ?y)))
   :ordered—subtasks (and (getTaxi ?x)
      (rideTaxi ?x ?y)))

(:method travel—by—plane
   :parameters (?x ?y ?ax ?ay — object)
   :task (travel ?x ?y)
   :precondition (and (at ?x) (longDistance ?x ?y)
                 (airport ?ax ?x) (airport ?ay ?y))
   :subtasks (and (task0 (getTicket ?ax ?ay ?x ?y))
                 (task1 (travel ?x ?ax))
                 (task2 (fly ?ax ?ay ?x ?y))
                 (task3 (travel ?ay ?y)))
   :ordering (and (task0 < task2) (task0 < task2)
                 (task2 < task3) )
)
```

Problem 1

```
(define (problem pb1) (:domain travel)
   (:objects
        portoalegre saopaulo london — city
        poa gru lhr — airport)
   (:htn
     :tasks (and (travel portoalegre saopaulo))
     :ordering ()
     :constraints ()
   )

   (:init
       (at portoalegre)
       (airport poa portoalegre)
       (airport gru saopaulo)
       (airport lhr london)
       (longDistance portoalegre saopaulo)
       (longDistance portoalegre london)
       (longDistance saopaulo london)
   )
)
```

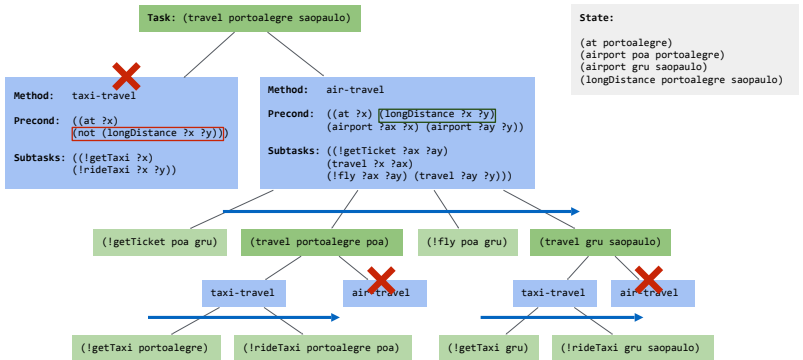
Problem 2

```
(define (problem pb2) (:domain travel)
   (:objects
        portoalegre saopaulo london viamao — city
        poa gru lhr — airport)
   (:htn
     :tasks (and (travel portoalegre viamao))
     :ordering ()
     :constraints ()
   )

   (:init
       (at portoalegre)
       (airport poa portoalegre)
       (airport gru saopaulo)
       (airport lhr london)
       (longDistance portoalegre saopaulo)
       (longDistance portoalegre london)
       (longDistance saopaulo london)
   )
)
```

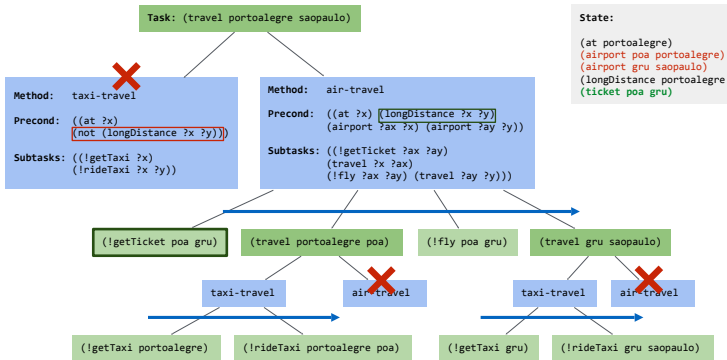
Solution to the travel problem

1 Simple Task Network (STN) Planning



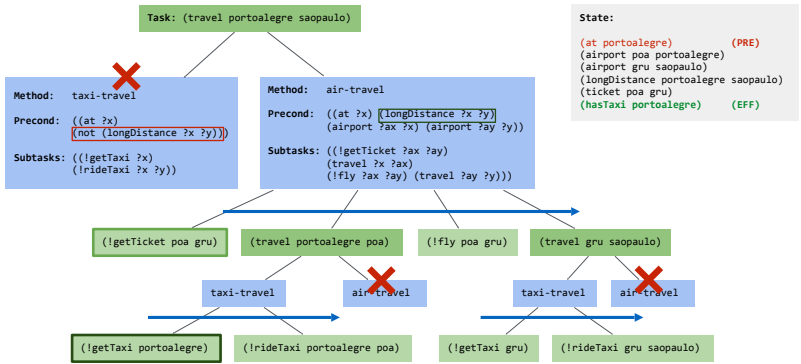
Solution to the travel problem

1 Simple Task Network (STN) Planning



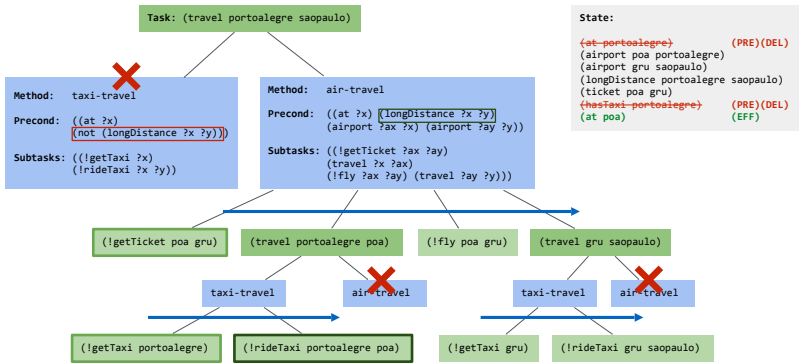
Solution to the travel problem

1 Simple Task Network (STN) Planning



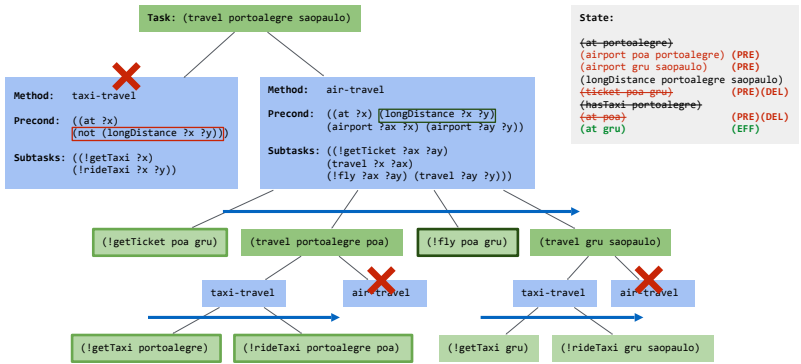
Solution to the travel problem

1 Simple Task Network (STN) Planning



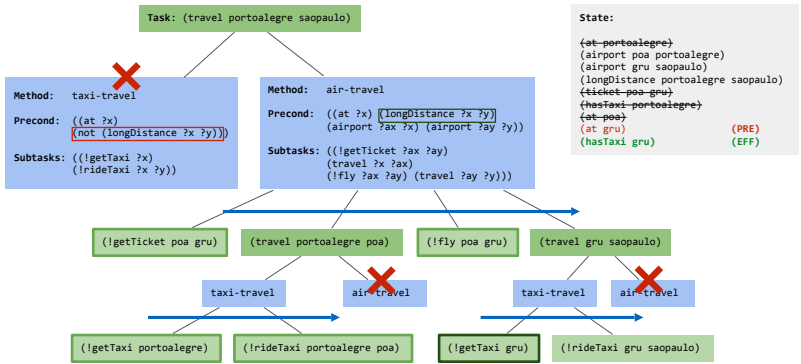
Solution to the travel problem

1 Simple Task Network (STN) Planning



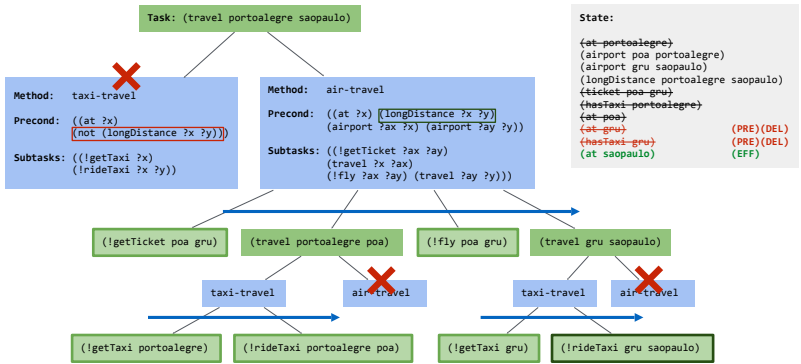
Solution to the travel problem

1 Simple Task Network (STN) Planning



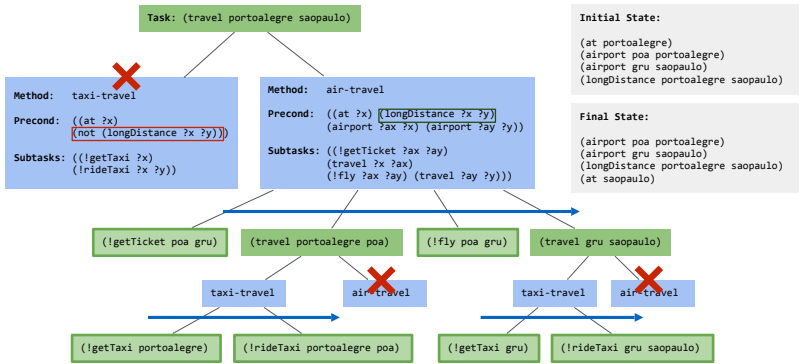
Solution to the travel problem

1 Simple Task Network (STN) Planning



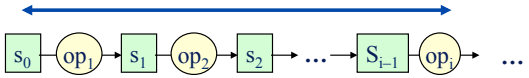
Solution to the travel problem

1 Simple Task Network (STN) Planning

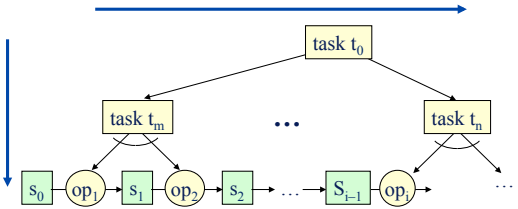


Comparison to Forward and Backward Search

- In state-space planning, must choose whether to search forward or backward

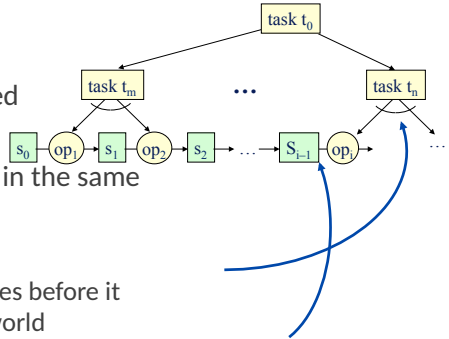


- In HTN planning, there are two choices to make about direction:
  - forward or backward
  - up or down
- TFD goes down and forward



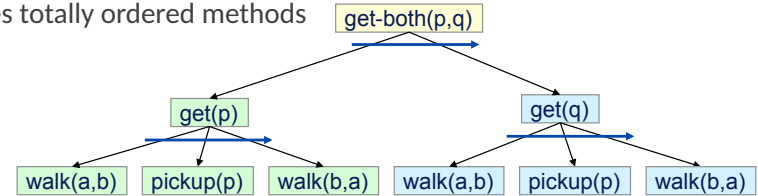
Comparison to Forward and Backward Search

- Like a backward search, TFD is goal-directed
  - Goals correspond to tasks
- Like a forward search, it generates actions in the same order in which they will be executed
- Whenever we want to plan the next task
  - We already planned everything that comes before it
  - Thus, we know the current state of the world

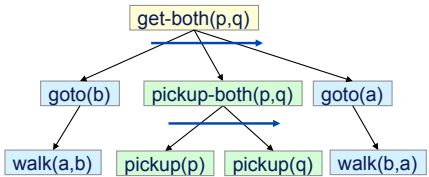


Limitation of Ordered-Task Planning

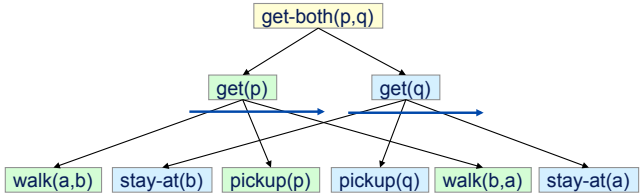
- TFD requires totally ordered methods



- Cannot interleave subtasks of different tasks
  - Sometimes this makes things awkward
  - Need to write methods that reason globally instead of locally



- With partially ordered methods, the subtasks can be interleaved



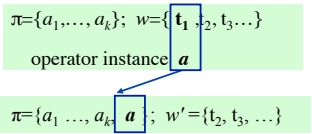
- Fits many planning domains better
- Requires a more complicated planning algorithm

Algorithm for Partial-Order STNs

function PFD(s, w, O, M)
  if $w = \emptyset$ **then return** $\langle\rangle$ ▷ *(i.e. the empty plan)*
  nondeterministically choose any $t_u \in w$ that has no predecessors in w
  if t_u is primitive **then**
    $active \leftarrow \{\langle a, \sigma\rangle | a \in O, a = t_u \cdot \sigma,$ and $s \models pre(a)\}$
      ▷ *Match applicable ground instances of actions to the task*
    if $active = \emptyset$ **then return** $failure$
    nondeterministically choose any $\langle a, \sigma\rangle \in$ `active`
    $\pi \leftarrow$ PFD($\gamma(s, a), (w - \{t_u\})\sigma, O, M$)
    if $\pi = failure$ **then return** $failure$
    else return $a \cdot \pi$
  else ▷ t_u *is nonprimitive*
    $active \leftarrow \{(m, \sigma) | m \in M, m = t_u\sigma,$ and $s \models pre(m)\}$
      ▷ *Match applicable ground instances of methods to the task*
    if $active = \emptyset$ **then return** $failure$
    nondeterministically choose any $\langle m, \sigma\rangle \in active$
    nondeterministically choose any task network $w' \in \delta(w, t_u, m, \sigma)$
    return PFD(s, w', O, M)

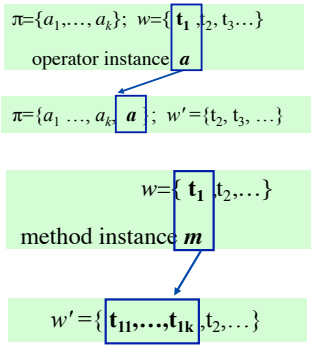
Algorithm for Partial-Order STNs

1: **function** PFD(s, w, O, M)
2: **if** $w = \emptyset$ **then return** $\langle\rangle$ ▷ *(i.e. the empty plan)*
3: nondeterministically choose any $t_u \in w$ that has no predecessors in w
4: **if** t_u is primitive **then**
5: $active \leftarrow \{\langle a, \sigma\rangle | a \in O, a = t_u \cdot \sigma,$ and $s \models pre(a)\}$
▷ *Match applicable ground instances of actions to the task*
6: **if** $active = \emptyset$ **then return** $failure$
7: nondeterministically choose any $\langle a, \sigma\rangle \in$ `active`
8: $\pi \leftarrow$ PFD($\gamma(s, a), (w - \{t_u\})\sigma, O, M$)
9: **if** $\pi = failure$ **then return** $failure$
10: **else return** $a \cdot \pi$
11: **else** ▷ t_u *is nonprimitive*
12: $active \leftarrow \{(m, \sigma) | m \in M, m = t_u\sigma,$ and $s \models pre(m)\}$
▷ *Match applicable ground instances of methods to the task*
13: **if** $active = \emptyset$ **then return** $failure$
14: nondeterministically choose any $\langle m, \sigma\rangle \in active$
15: nondeterministically choose any task network $w' \in \delta(w, t_u, m, \sigma)$
16: **return** PFD(s, w', O, M)



Algorithm for Partial-Order STNs

1: **function** PFD(s, w, O, M)
2: **if** $w = \emptyset$ **then return** $\langle\rangle$ ▷ *(i.e. the empty plan)*
3: nondeterministically choose any $t_u \in w$ that has no predecessors in w
4: **if** t_u is primitive **then**
5: $active \leftarrow \{\langle a, \sigma\rangle | a \in O, a = t_u \cdot \sigma,$ and $s \models pre(a)\}$
▷ *Match applicable ground instances of actions to the task*
6: **if** $active = \emptyset$ **then return** $failure$
7: nondeterministically choose any $\langle a, \sigma\rangle \in$ `active`
8: $\pi \leftarrow$ PFD$(\gamma(s, a), (w - \{t_u\})\sigma, O, M)$
9: **if** $\pi = failure$ **then return** $failure$
10: **else return** $a \cdot \pi$
11: **else** ▷ *t_u is nonprimitive*
12: $active \leftarrow \{(m, \sigma) | m \in M, m = t_u\sigma,$ and $s \models pre(m)\}$
▷ *Match applicable ground instances of methods to the task*
13: **if** $active = \emptyset$ **then return** $failure$
14: nondeterministically choose any $\langle m, \sigma\rangle \in active$
15: nondeterministically choose any task network $w' \in \delta(w, t_u, m, \sigma)$
16: **return** PFD(s, w', O, M)



Solving Partial-Order STNs

Data structures are slightly more complex

- Intuitively, w is a partially ordered set of tasks $\{t_1, t_2, \ldots\}$
- But w may contain a task more than once
  e.g., travel from PUCRS to CMU twice
- The mathematical definition of a set does not allow this
- Define w as a partially ordered set of task nodes $\{u_1, u_2, \ldots\}$
  - Each task node u corresponds to a task t_u
- In the explanation, I talked about t and ignore u

And there is more book keeping:

- $\delta(w, t_u, m, \sigma)$ has a complicated definition. Here is what it means:
  - We nondeterministically selected t_u as the task to do first
  - Must do t_u's first subtask before the first subtask of every $t_i \neq t_u$
  - Insert ordering constraints to ensure that this happens

Comparison to Classical Planning

- STN planning is strictly more expressive than classical planning
- Any classical planning problem can be translated into an STN problem in polynomial time
- Several ways to do this.

STRIPS to HTN

1 Simple Task Network (STN) Planning

It is possible to convert STRIPS problems directly to a solvable HTN instance without domain knowledge. A simplified conversion consists of:[1]

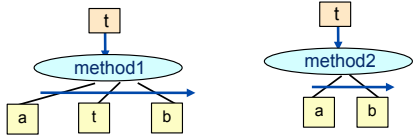
- For each STRIPS operator o, declare a primitive task f with the same effects and preconditions as o
- Add a dummy primitive task f_d with no effects or preconditions:
- Declare a single non-primitive task symbol t.
- For each primitive task f, construct a method of the form: $t \Rightarrow \langle f, t \rangle$
- Declare one last method $t \Rightarrow \langle f_d \rangle$, and note that
  - t can be expanded to any sequence of actions ending with f_d
  - provided that $pre(f)$ are satisfied for all primitive tasks f
- For each goal predicate G_i, create a single non-primitive task t_{gi}, with a single method $t_{gi} \Rightarrow \langle \rangle$ and precondition G_i
- The input task network has the form $\langle t, t_{g1}, \ldots t_{gn} \rangle$, where $G_1, \ldots, G_n$ are the STRIPS-style goals we want to achieve.

[1] For more details: Erol, K.; Hendler, J.; and Nau, D. Complexity results for hierarchical task-network planning. In Annals of Mathematics and Artificial Intelligence, 1994.

Comparison to Classical Planning

Some STN planning problems are not expressible in classical planning.
Example:

- Two STN methods: No arguments and no preconditions
- Two operators, a and b:
  Again, no arguments and no preconditions
- Initial state is empty, initial task is t
- Set of solutions is $\{a^n b^n \mid n > 0\}$
- No classical planning problem has this set of solutions
  - The state-transition system is a finite-state automaton
  - No finite-state automaton can recognize $\{a^n b^n \mid n > 0\}$
- Can even express undecidable problems using STNs



To continue in the next session.