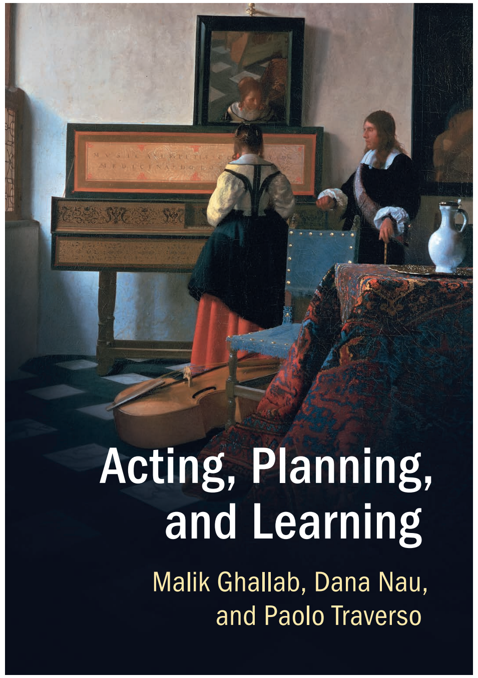
Last update: 11:09 PM, March 7, 2025



Chapter 6
Acting with HTNs

Dana S. Nau

University of Maryland

with contributions from

Mark “mak” Roberts

Acting, Planning, and Learning

Malik Ghallab, Dana Nau, and Paolo Traverso

Using HTN Domain Models for Acting

- Unlike an HTN domain model, the actor’s environment is not necessarily deterministic or static
  - Exogenous events, unanticipated action outcomes ⇒ current state may be different from what an HTN model would predict
- Actor can’t backtrack to a previous state; prior actions are in the past

- HTN domain models still are very useful for providing *operational* models to the actor
  - How to carry out “standard operating procedures”
  - How to perform complex tasks without searching through a large state space
  - How to avoid situations where unanticipated events are likely to cause bad outcomes
  - How to recover when unanticipated events occur

Reactive HTN Actor

- Like TO-HTN-Forward but executes each action
  - Can similarly modify other Chapter 5 algorithms

Line

0 Return success or failure, not a plan

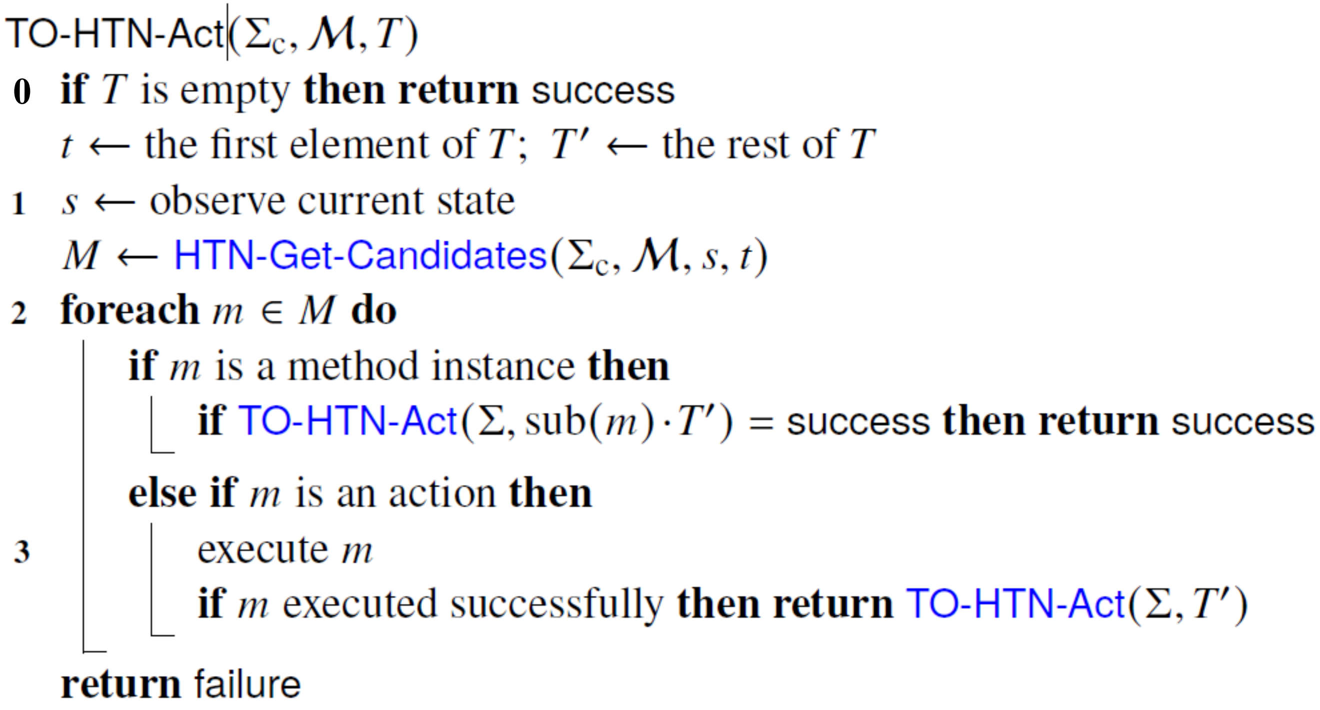
1 *s* isn't an argument, observe it instead

3 Instead of computing γ, execute action

2 Failure recovery: if *m* fails, try next one

- if they all fail, return failure to next higher level in the recursion stack, to try other methods there

- At Line 2, a bad method instance can lead to non-optimal solution or failure
  - Can use a heuristic function
  - Can call an HTN planner – but other ways have less computational overhead



```
TO-HTN-Act($\Sigma_c, \mathcal{M}, T$)
0  if $T$ is empty then return success
   $t \leftarrow$ the first element of $T$;  $T' \leftarrow$ the rest of $T$
1  $s \leftarrow$ observe current state
   $M \leftarrow$ HTN-Get-Candidates($\Sigma_c, \mathcal{M}, s, t$)
2  foreach $m \in M$ do
       if $m$ is a method instance then
           if TO-HTN-Act($\Sigma$, sub($m$)$\cdot T'$) = success then return success
       else if $m$ is an action then
3          execute $m$
           if $m$ executed successfully then return TO-HTN-Act($\Sigma, T'$)
   return failure
```

Poll 1. Is line doing backtracking?

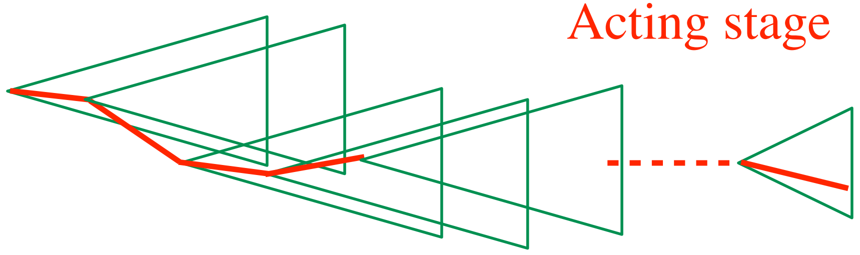
A. Yes B. No C. Unsure

Run-HLookahead

```
HTN-Run-Lookahead($\Sigma$, $T$)
    while True do:
        $s \leftarrow$ observed current state
        $\pi$ = Lookahead($\Sigma$, $s$, $T$)
        if $\pi$ = failure then return failure
        if $\pi = \langle\,\rangle$ then return success
        $a \leftarrow$ pop($\pi$)
        trigger execution of $a$
```

- Here, *Lookahead* is an HTN planner
- Goal formula may not exist
  - Cannot rely on $s \vDash g$
  - Need *Lookahead* to return $\langle\,\rangle$ iff no actions are needed to accomplish T

- Call *Lookahead*, get π, perform 1st action, call *HLookahead* again …
- Useful when unexpected things are likely to happen
  - Replans immediately
- *Lookahead* needs to return quickly
  - Otherwise, HTN-Run-Lookahead may pause repeatedly waiting for *Lookahead* to return
  - May want *Lookahead* to look a limited distance or horizon ahead



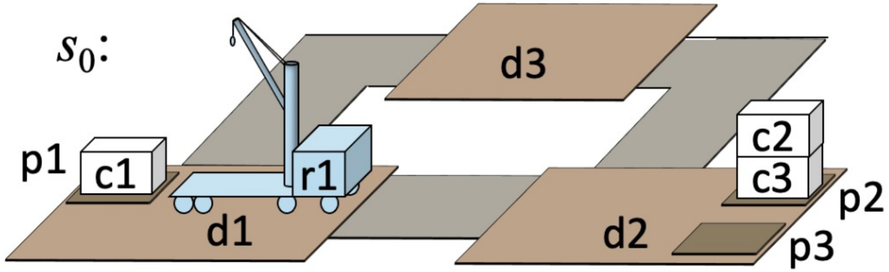
Planning stage

Acting stage

Run-HLookahead (Example 1)

```
HTN-Run-Lookahead(Σ, $\mathcal{T}$)
    while True do:
        $s \leftarrow$ observed current state
        $\pi$ = Lookahead(Σ, $s$, $\mathcal{T}$)
        if $\pi$ = failure then return failure
        if $\pi = \langle\ \rangle$ then return success
        $a \leftarrow \text{pop}(\pi)$
        trigger execution of $a$
```

s_0:



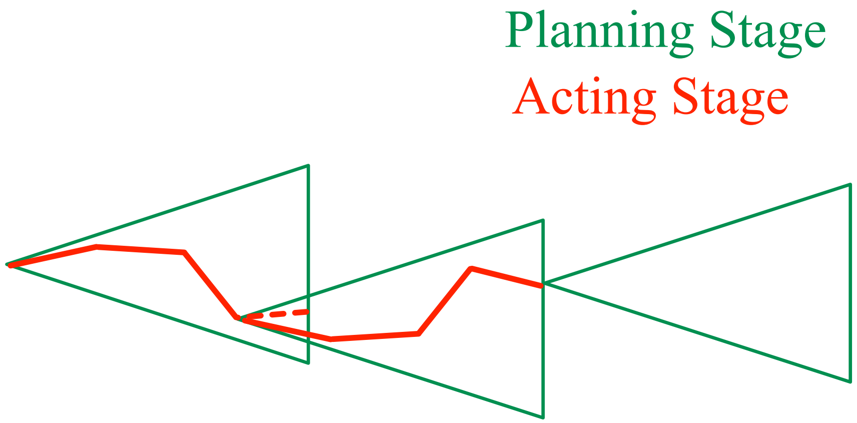
- Call HTN-Run-Lookahead with *Lookahead* = TO-HTN-Forward (THF)
  - Σ = the TOHTN domain in Example 5.8
  - P = (Σ, s_0, T = ⟨{pile(c1)=p2}⟩)
- If nothing unexpected happens:
  - Call TO-HTN-Forward(Σ, s_0, $\mathcal{T}$)
    - π = ⟨take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)⟩
  - Execute take(r1,c1,c2,p1,d1)
  - Call THF(..), get π = ⟨move(r1,d1,d2), put(r1,c1,c3,p2,d2)⟩
  - execute move(r1,d1,d2),
  - call THF(..), get π = ⟨put(r1,c1,c3,p2,d2)⟩
  - execute put(r1,c1,c3,p2,d2),
  - Call THF(..), get π = ⟨ ⟩, return success
- If something unexpected happens but the problem is still solvable:
  - Call THF(..) with latest observed state, it returns a new plan
  - This could fail if there is no applicable method for the new state!

Run-Lazy-HLookahead

```
HTN-Run-Lazy-Lookahead($\Sigma$, $\mathcal{T}$)
    $\pi \leftarrow \langle\,\rangle$; $a \leftarrow$ nil
    while True do:
        if $\pi = \langle\,\rangle$ or execution of $a$ failed then
            $s \leftarrow$ observed state
            $\pi$ = Lookahead($\Sigma$, $s$, $\mathcal{T}$)
            if $\pi$ = failure then return failure
            if $\pi = \langle\,\rangle$ then return success
        $a \leftarrow$ pop($\pi$)
        trigger execution of $a$
```

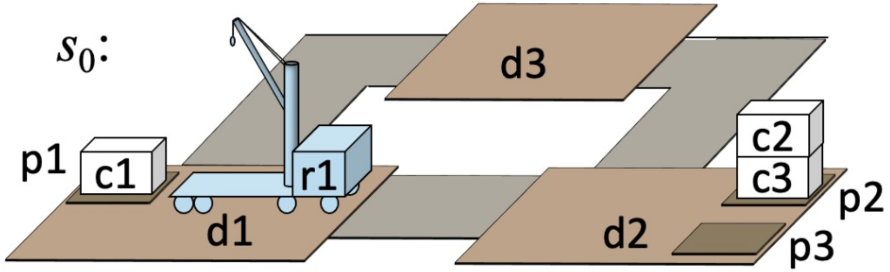
- Two different tests for ⟨ ⟩
  - If we've exhausted the current plan, call *Lookahead*
  - If *Lookahead* returns ⟨ ⟩, return success
- Requires *Lookahead* to return ⟨ ⟩ iff no actions are needed to accomplish $\mathcal{T}$

- Could also add a *Simulate* program as in Run-Lazy-Lookahead



Run-Lazy-HLookahead (Example 1)

```
HTN-Run-Lazy-Lookahead($\Sigma$, $\mathcal{T}$)
    $\pi \leftarrow \langle\,\rangle$; $a \leftarrow$ nil
    while True do:
        if $\pi = \langle\,\rangle$ or execution of $a$ failed then
            $s \leftarrow$ observed state
            $\pi$ = Lookahead($\Sigma$, $s$, $\mathcal{T}$)
            if $\pi$ = failure then return failure
            if $\pi = \langle\,\rangle$ then return success
        $a \leftarrow$ pop($\pi$)
        trigger execution of $a$
```



- Call HTN-Run-Lazy-Lookahead with *Lookahead* = TO-HTN-Forward (THF)
  - Σ = TOHTN domain in Example 5.8
  - initial state s_0, $\mathcal{T} = \langle$\{pile(c1)=p2\}$\rangle$
- If nothing unexpected happens:
  - Call THF(Σ, s_0, $\mathcal{T}$)
    - $\pi = \langle$take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$
  - Pop actions from π and execute them, until $\pi = \langle\,\rangle$
  - Call THF again, get $\pi = \langle\,\rangle$, return success
- If something unexpected happens but the problem is still solvable:
  - Eventually, either $\pi = \langle\,\rangle$ or a has failed
  - Call THF with observed state, it returns a new plan
- HTN-Run-Lookahead is similar but it calls *Lookahead* before each action is executed

Example 2

- POHTN planning domain
  - Cranes at loading docks, not on the robots
- Actions:
  - The usual move action, and these:

$\mathsf{unstack}(k, c, c', p, d)$ *// take container c from pile p*
  pre: $\mathsf{at}(k, d), \mathsf{at}(p, d), \mathsf{holding}(k) = \mathsf{nil}, \mathsf{pos}(c) = c', \mathsf{top}(p) = c$
  eff: $\mathsf{holding}(k) \leftarrow c, \mathsf{pos}(c) \leftarrow k, \mathsf{pile}(c) \leftarrow \mathsf{nil}, \mathsf{top}(p) \leftarrow c'$

$\mathsf{stack}(k, c, c', p, d)$ *// put container c onto pile p*
  pre: $\mathsf{at}(k, d), \mathsf{at}(p, d), \mathsf{holding}(k) = c, \mathsf{top}(p) \leftarrow c'$
  eff: $\mathsf{holding}(k) \leftarrow \mathsf{nil}, \mathsf{pos}(c) = c', \mathsf{pile}(c) \leftarrow p, \mathsf{top}(p) = c$

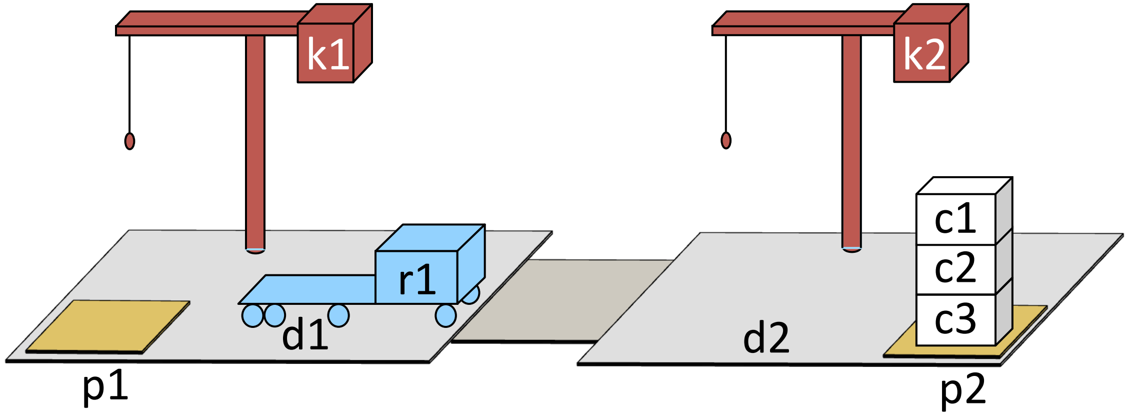
$\mathsf{unload}(k, c, r, d)$ *// take container c from robot r*
  pre: $\mathsf{at}(k, d), \mathsf{holding}(k) = c, \mathsf{loc}(r) = d$
  eff: $\mathsf{cargo}(r) \leftarrow c, \mathsf{pos}(c) \leftarrow r, \mathsf{holding}(k) \leftarrow \mathsf{nil}$

$\mathsf{load}(k, c, r, d)$ *// put container c onto robot r*
  pre: $\mathsf{at}(k, d), \mathsf{holding}(k) = \mathsf{nil}, \mathsf{loc}(r) = d, \mathsf{cargo}(r) = c$
  eff: $\mathsf{pos}(c) \leftarrow k, \mathsf{holding}(k) \leftarrow c, \mathsf{cargo}(r) \leftarrow \mathsf{nil}$

- Methods

$\mathsf{m1\text{-}put\text{-}on\text{-}robot}(k, c, c', r, d, p)$
  task: $\mathsf{put\text{-}on\text{-}robot}(c, r)$
  pre: $\mathsf{cargo}(r) = \mathsf{nil}, \mathsf{top}(p) = c, \mathsf{at}(p, d),$
    $\mathsf{attached}(k, d), \mathsf{holding}(k) = \mathsf{nil}$
  sub: $(\mathsf{t1}, \mathsf{navigate}(r, d)),$ *// compound task*
    $(\mathsf{t2}, \mathsf{unstack}(k, c, c', p, d)),$ *// action*
    $(\mathsf{t3}, \mathsf{load}(k, c, r, d))$ *// action*
  $\prec$: $\mathsf{t1} \prec \mathsf{t3},\ \mathsf{t2} \prec \mathsf{t3}$

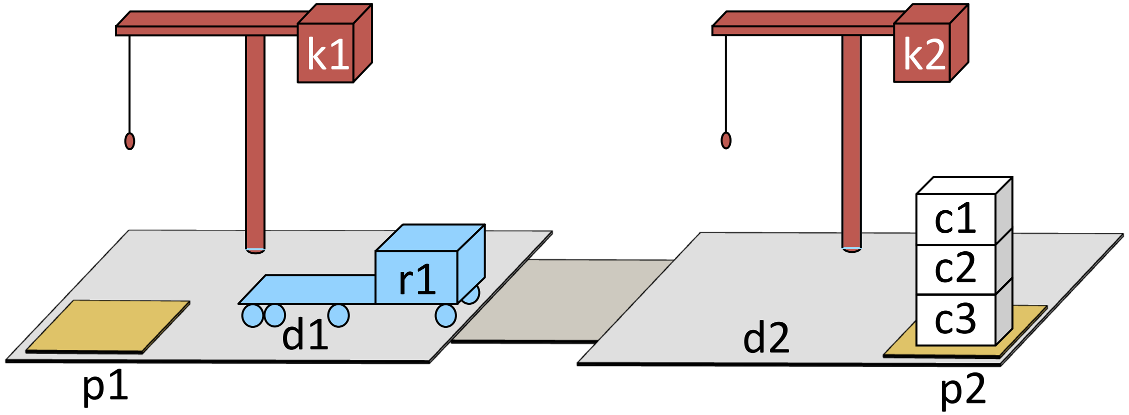
- The usual navigate methods



Example 2

- Call HTN-Run-Lazy-Lookahead with *Lookahead* = POHTN-Forward
  - Σ = POHTN domain on previous page
  - initial state s_0, the only task in $\mathcal{T}$ is put-on-robot(c1,r1)
- If nothing unexpected happens:
  - Call POTHN-Forward(Σ, s_0, $\mathcal{T}$)
    - Two solution plans, suppose it returns this one:
    - π_2 = ⟨unstack(k2,c1,c2,p2,d2), move(r1,d1,d2), load(k2,c1,r1,d2)⟩
  - Pop actions from π and execute them, until π = ⟨ ⟩
  - Call POHTN-Forward again, get π = ⟨ ⟩, return success
- Suppose move fails without changing the current state:
  - Call POHTN-Forward(Σ, s_0, $\mathcal{T}$)
    - failure: no applicable methods when k2 is holding c1
- Run-Lookahead
  - Call POHTN-Forward, get plan, execute unstack, call PPlan, PPlan fails

```
Run-Lazy-HLookahead($\Sigma$, $\mathcal{T}$)
    $\pi \leftarrow \langle\,\rangle$; $a \leftarrow$ nil
    while True do:
        if $\pi = \langle\,\rangle$ or execution of $a$ failed then
            $s \leftarrow$ observed state
            $\pi$ = HTN-Lookahead($\Sigma$, $s$, $\mathcal{T}$)
            if $\pi$ = failure then return failure
            if $\pi = \langle\,\rangle$ then return success
        $a \leftarrow$ pop($\pi$)
        trigger execution of $a$
```



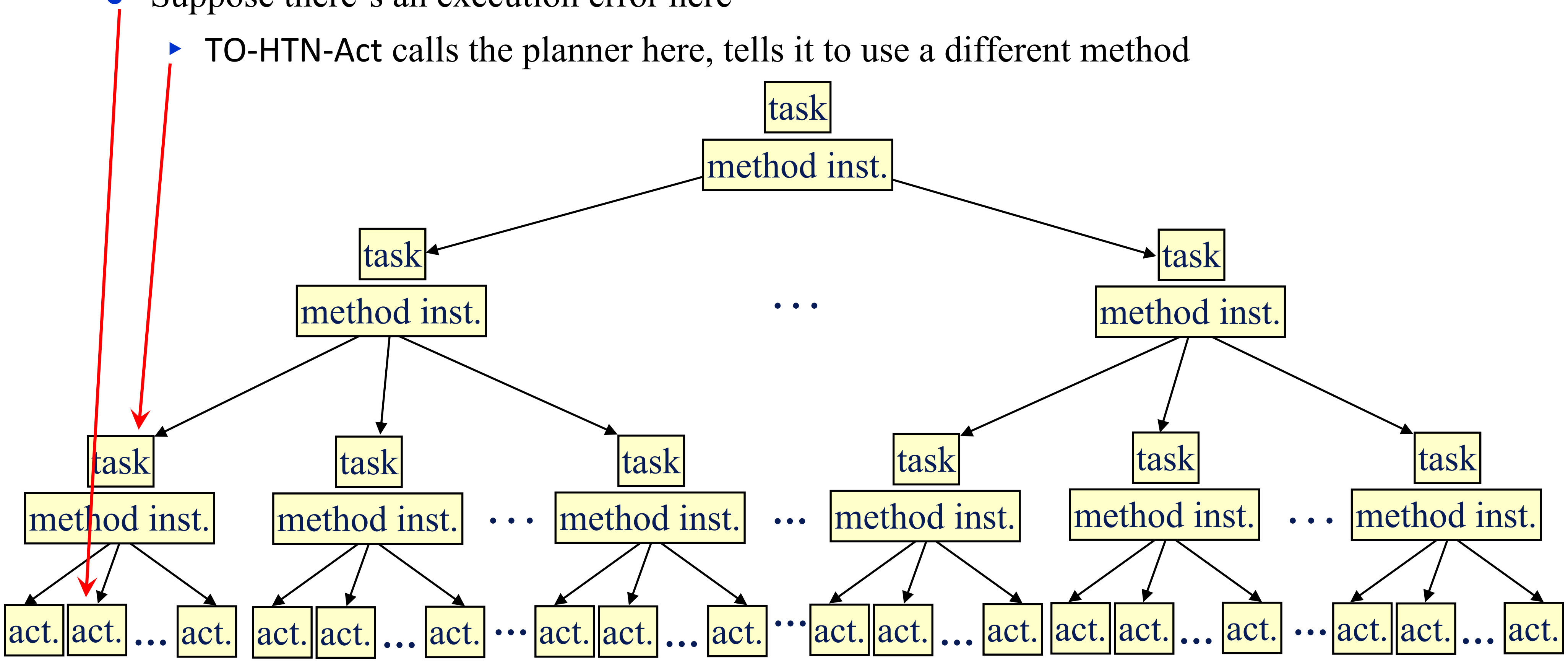
Error Recovery in HTN Domains

- HTN methods require the solution plan to follow a particular trajectory
- Encode requirements that aren't explicit in the classical planning domain
  - Safety requirements:
    - Secure a container onto the robot before starting to move the robot
  - Commitments to other agents
    - Don't use a particular resource, because others may need it
  - A company's standard operating procedures

- HTN-Run-Lookahead and HTN-Run-Lazy-Lookahead don't know anything about the trajectory requirements
- That's OK if nothing goes wrong
- If unexpected events occur, need to recover in a way that still satisfies the trajectory requirements
- Three approaches
  1. Modify TO-HTN-Act to call an HTN planner
     - HTN planner returns a method selection
  2. Modify HTN planner to return a solution tree
     - Actor traverses the tree
  3. Actor calls HTN planner to do replanning in a modified domain

TO-HTN-Act (modified) with an HTN Planner

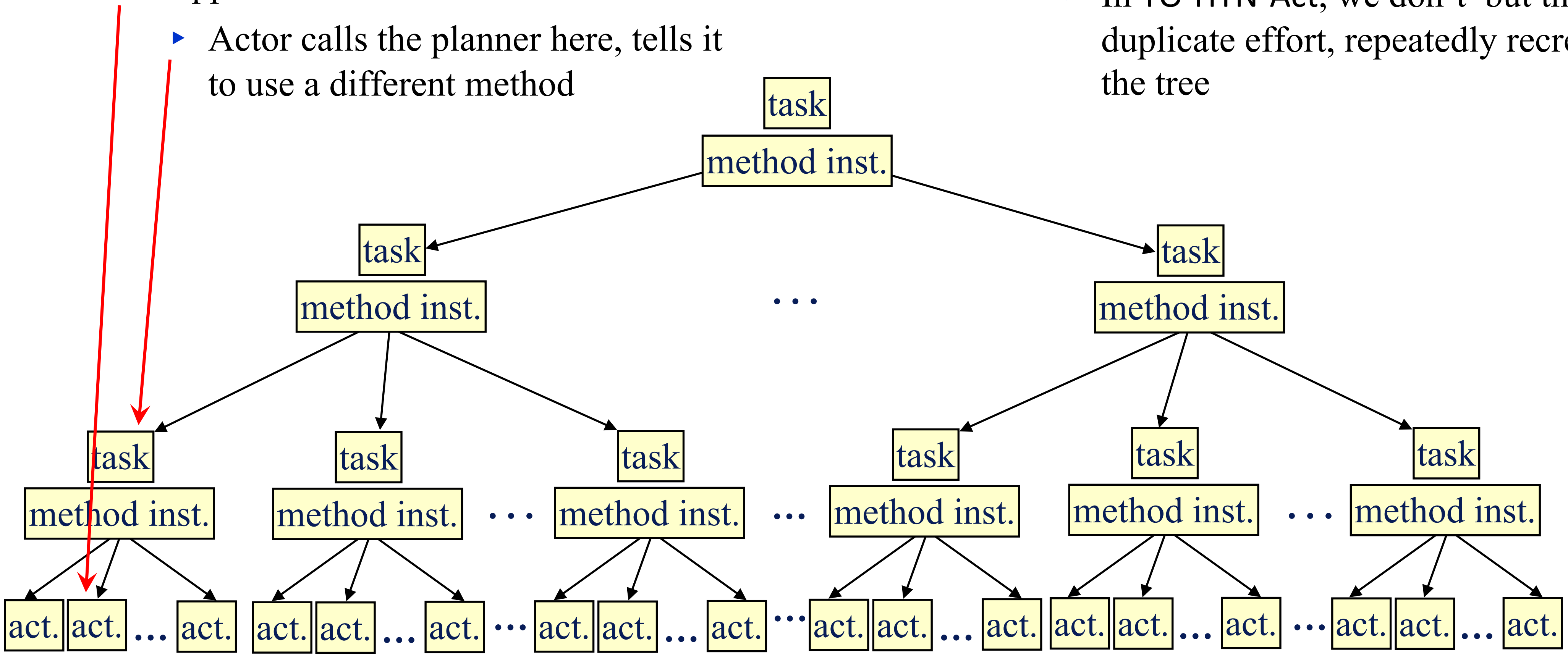
- HTN planner similar to TO-HTN-Forward, but returns the top-level method in its solution tree
- Suppose there's an execution error here
  - TO-HTN-Act calls the planner here, tells it to use a different method



Traversing a Solution Tree

- HTN planner returns a solution tree
- Actor traverses the tree
- Suppose there’s an execution error here
    - Actor calls the planner here, tells it to use a different method

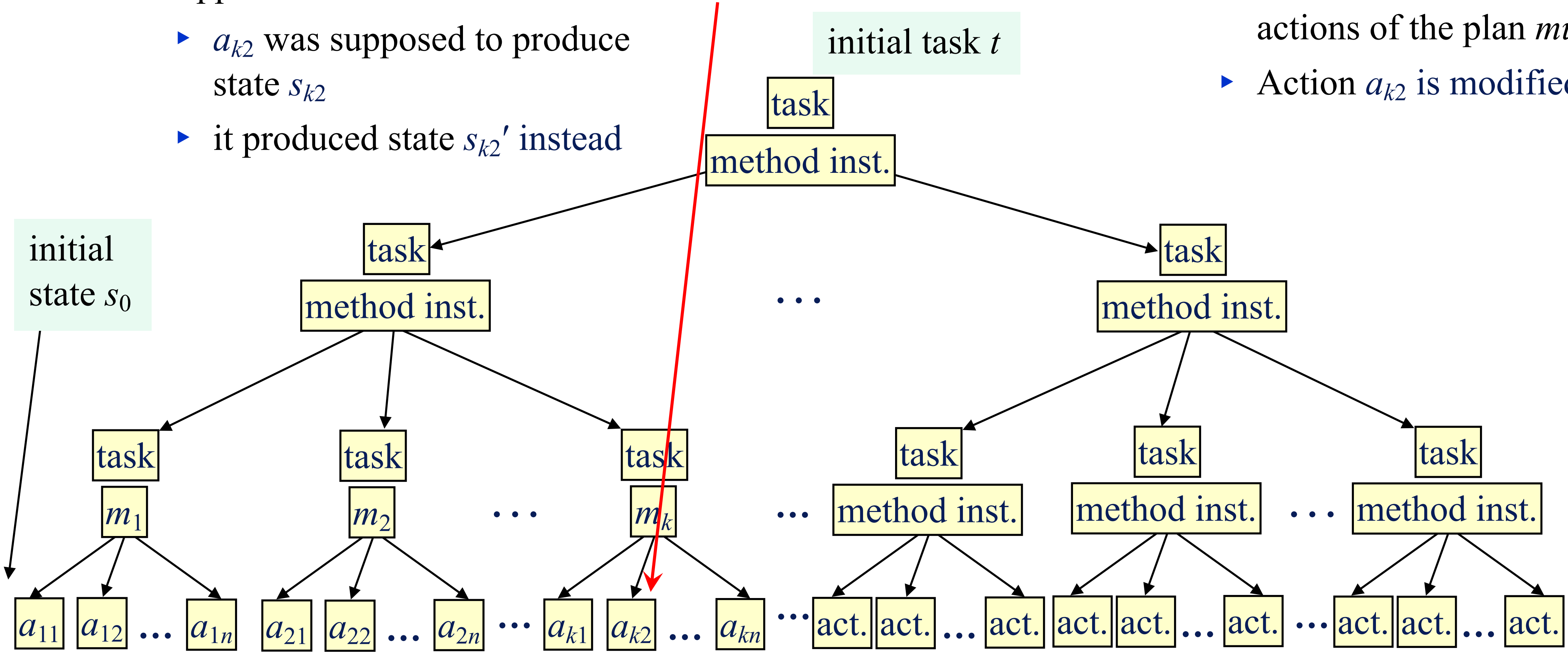
- HTN planner returns a solution tree, actor traverses the tree
- Time vs. space tradeoff
    - Here, we need the entire tree
    - In TO-HTN-Act, we don’t but the actor and planner duplicate effort, repeatedly recreating the current part of the tree



Modifying the Planning Domain

- Modified version of HTN-Run-Lazy-Lookahead
  - Calls TPlan to get a plan
- Suppose there’s an execution error here
  - a_{k2} was supposed to produce state s_{k2}
  - it produced state s_{k2}' instead

- Actor calls TO-HTN-Forward again, with the same initial state s_0 and task t as before
- Modified planning domain
  - Methods are modified so that the initial actions of the plan *must* be $a_{11}, \ldots, a_{kn}$
  - Action a_{k2} is modified so that $\gamma(s_{k1}, a_{k2}) = s_{k2}'$



Summary

- Issues
  - Actor’s environment may not be deterministic or static
  - Actor can’t backtrack to a previous state
- TO-HTN-Act: reactive actor similar to TO-HTN-Forward
- HTN-Run-Lookahead, HTN-Run-Lazy-Lookahead
  - Examples where they work well, where they don’t
- Error recovery in HTN domains
- Three approaches
  - TO-HTN-Act modified to call an HTN planner
  - Actor that traverses a solution tree
  - Actor that re-invokes TO-HTN-Forward on the original problem in a modified planning domain

- Tradeoff: time versus space