Lecture 17

CS 6260 Automated Planning and Learning

Acting with HTNs

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Some material based on lecture slides for Automated Planning and Acting by Nau/Roberts.

Announcement

Talk Announcement

- **Title:** Integration of Structured Reasoning and Data-driven Learning for Acting and Planning
- **Date:** March 24, 2026
- **Time:** 09:30 AM – 10:30 AM
- Online link will be posted on Moodle

Dr. Sunandita Patra
J. P. Morgan AI Research, USA
https://sunanditapatra.wixsite.com/camp

Using HTN Domains for Acting in the Environment

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$(\Sigma_c, \mathcal{M}, T)$

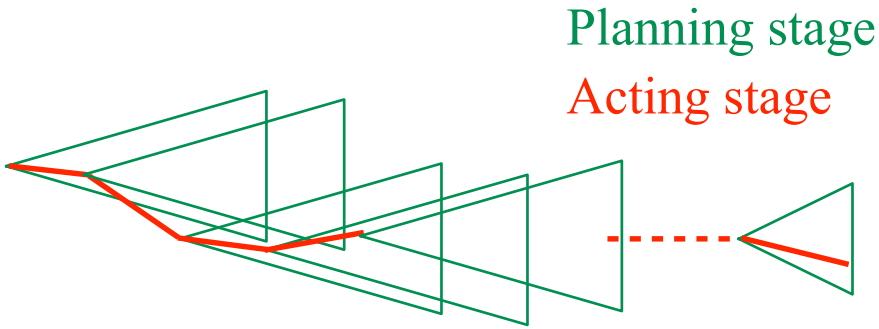
0 **if** T is empty **then return** success
  $t \leftarrow$ the first element of T; $T' \leftarrow$ the rest of T
1 $s \leftarrow$ observe current state
  $M \leftarrow$ HTN-Get-Candidates$(\Sigma_c, \mathcal{M}, s, t)$
2 **foreach** $m \in M$ **do**
    if m is a method instance **then**
      if TO-HTN-Act$(\Sigma,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

HTN-Run-Lookahead

HTN-Run-Lookahead(Σ, *T*)
  while True **do:**
    s ← observed current state
    π = *Lookahead*(Σ, *s*, *T*)
    if π = failure **then return** failure
    if π = ⟨ ⟩ **then return** success
    a ← pop(π)
    trigger execution of *a*

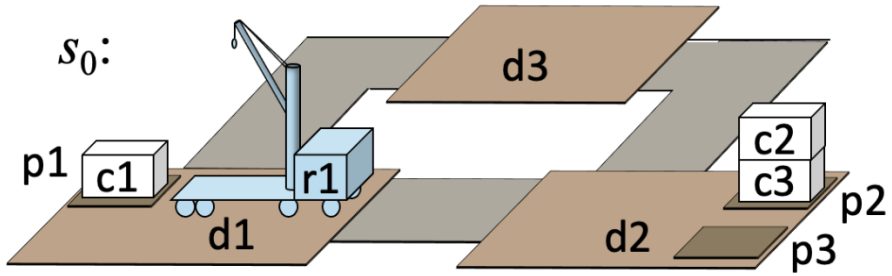
- Here, *Lookahead* is an HTN planner
- Goal formula may not exist
  - Cannot rely on $s \vDash g$
  - Need *Lookahead* to return ⟨ ⟩ iff no actions are needed to accomplish *T*

- Call *Lookahead*, get π, perform 1st action, call *Lookahead* 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



HTN-Run-Lookahead (Example)

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



- Call HTN-Run-Lookahead with *Lookahead* = TO-HTN-Forward (THF)
  - Σ = the TOHTN domain we saw earlier
  - P = (Σ, s_0, $T = \langle\{\text{pile(c1)=p2}\}\rangle$)
- If nothing unexpected happens:
  - Call TO-HTN-Forward(Σ, s_0, T)
    - $\pi = \langle$take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$
  - Execute take(r1,c1,c2,p1,d1)
  - Call THF(..), get $\pi = \langle$move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$
  - execute move(r1,d1,d2),
  - Call THF(..), get $\pi = \langle$put(r1,c1,c3,p2,d2)$\rangle$
  - execute put(r1,c1,c3,p2,d2),
  - Call THF(..), get $\pi = \langle\ \rangle$, 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!

HTN-Run-Lazy-Lookahead

HTN-Run-Lazy-Lookahead(Σ, $\mathcal{T}$)

$\pi \leftarrow \langle\,\rangle$; $a \leftarrow$ nil

while True **do**:

if $\pi = \langle\,\rangle$ or execution of a failed **then**

$s \leftarrow$ observed state

$\pi =$ *Lookahead*(Σ, s, $\mathcal{T}$)

if π = failure **then return** failure

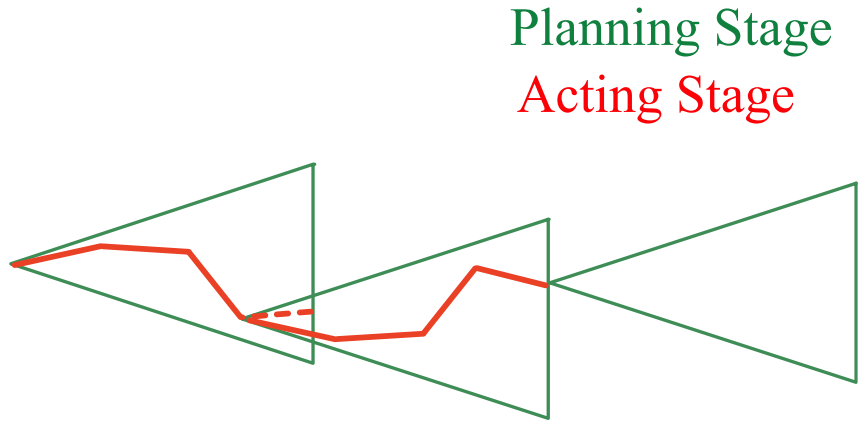
if $\pi = \langle\,\rangle$ **then return** success

$a \leftarrow$ pop(π)

trigger execution of a

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

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



HTN-Run-Lazy-Lookahead (Example)

HTN-Run-Lazy-Lookahead(Σ, $\mathcal{T}$)

$\pi \leftarrow \langle\,\rangle$; $a \leftarrow$ nil

while True **do**:

if $\pi = \langle\,\rangle$ or execution of a failed **then**

$s \leftarrow$ observed state

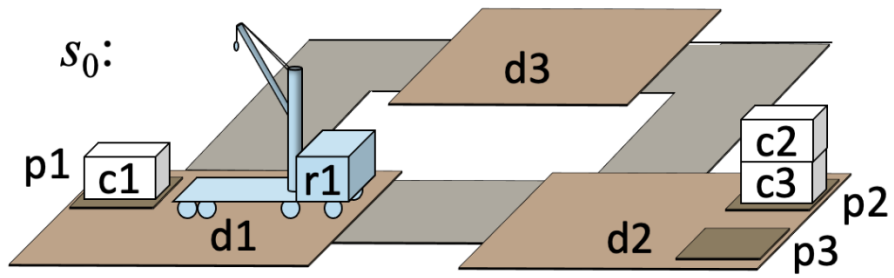
π = *Lookahead*(Σ, s, $\mathcal{T}$)

if π = failure **then return** failure

if $\pi = \langle\,\rangle$ **then return** success

$a \leftarrow$ pop(π)

trigger execution of a



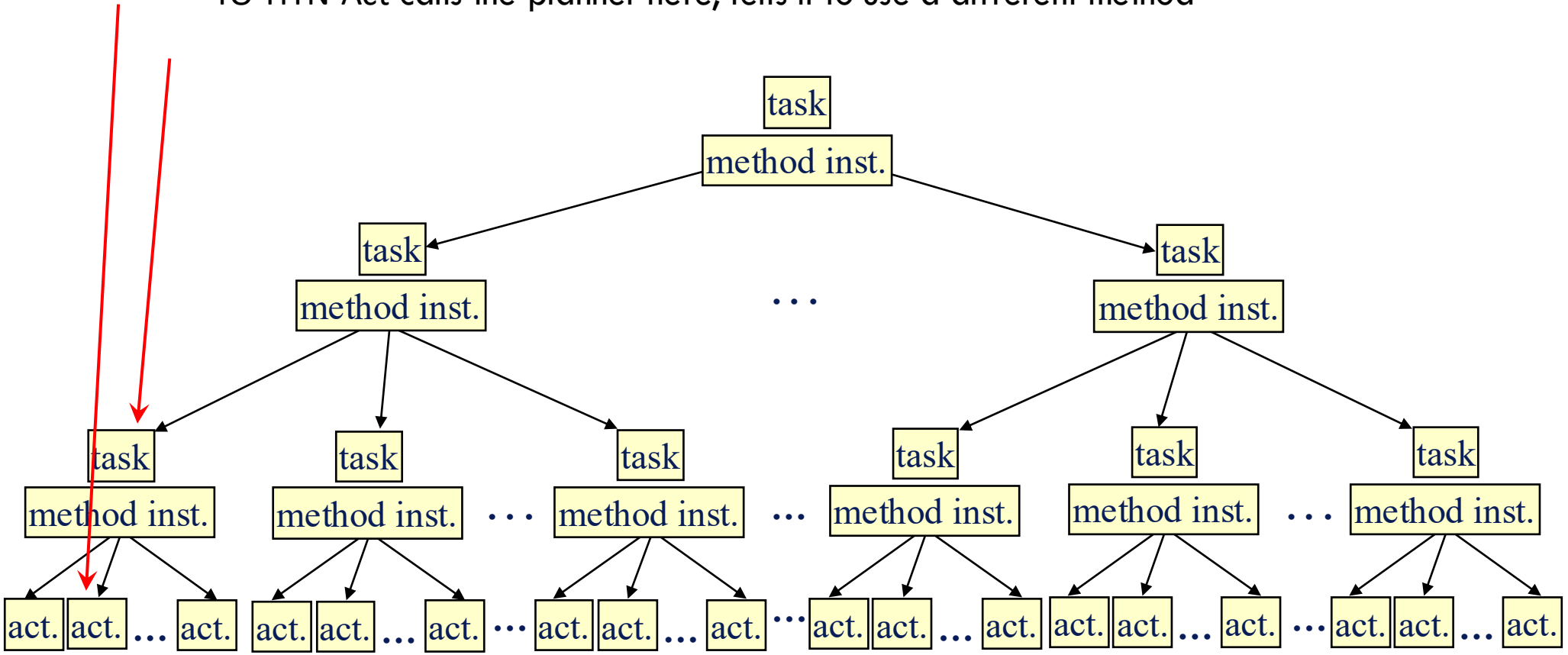
- Call HTN-Run-Lazy-Lookahead with *Lookahead* = TO-HTN-Forward (THF)
  - Σ = TOHTN domain in Example 5.8
  - initial state s_0, $\mathcal{T}$ = ⟨{pile(c1)=p2}⟩
- If nothing unexpected happens:
  - Call THF(Σ, s_0, $\mathcal{T}$)
    - π = ⟨take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)⟩
  - 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

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

Using an HTN Planner while Acting

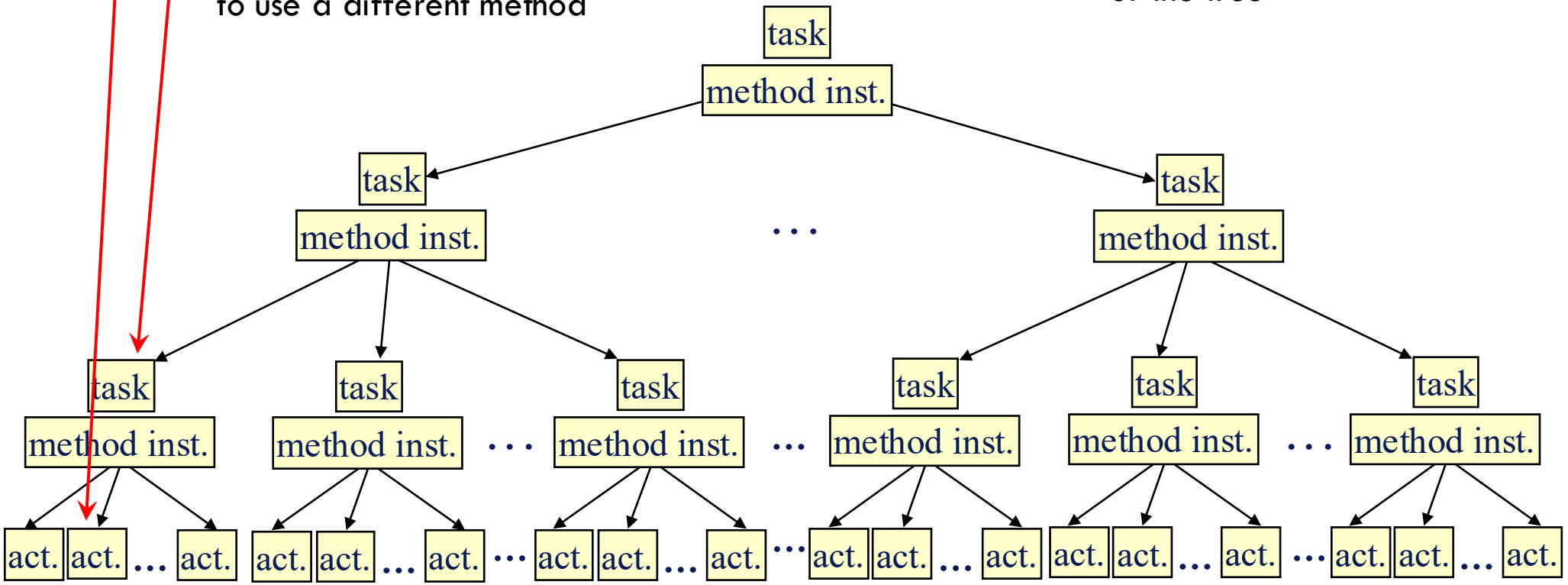
- HTN Planner returns the top-level method in its solution tree
- Suppose there's an execution error here
  - TO-HTN-Act calls the planner here, tells it to use a different method



Traversing a Solution Tree

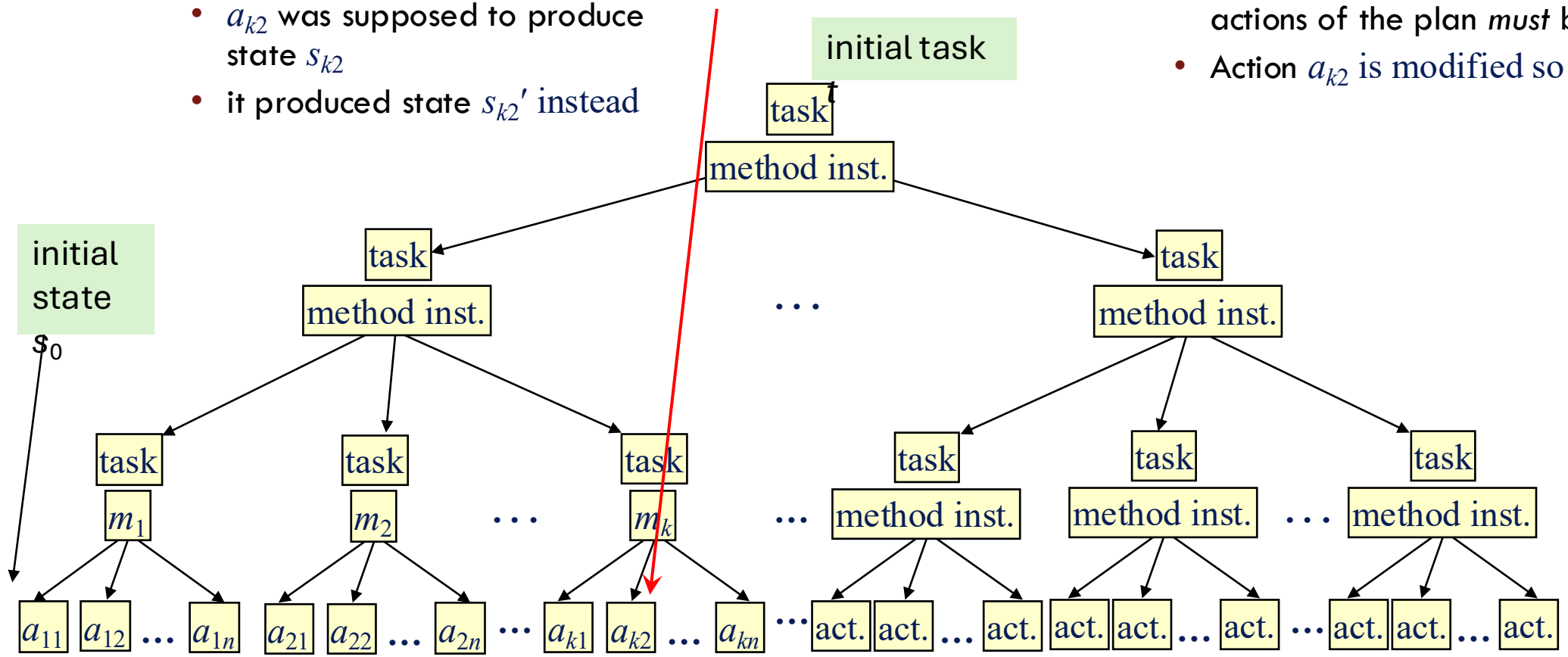
- HTN Planner returns a solution tree
- Actor traverses the tree
- Suppose there's an execution error here
  - Actor calls the planner here, tells it to use a different method

- HTN Planner returns a solution tree, actor traverses the tree
- Time vs. space tradeoff
  - Here, we need to 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
- 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