CAP6671 Intelligent Systems

Lecture 5:
Learning to Plan

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

Schedule: T & Th 9:00-10:15am

Location: HEC 302

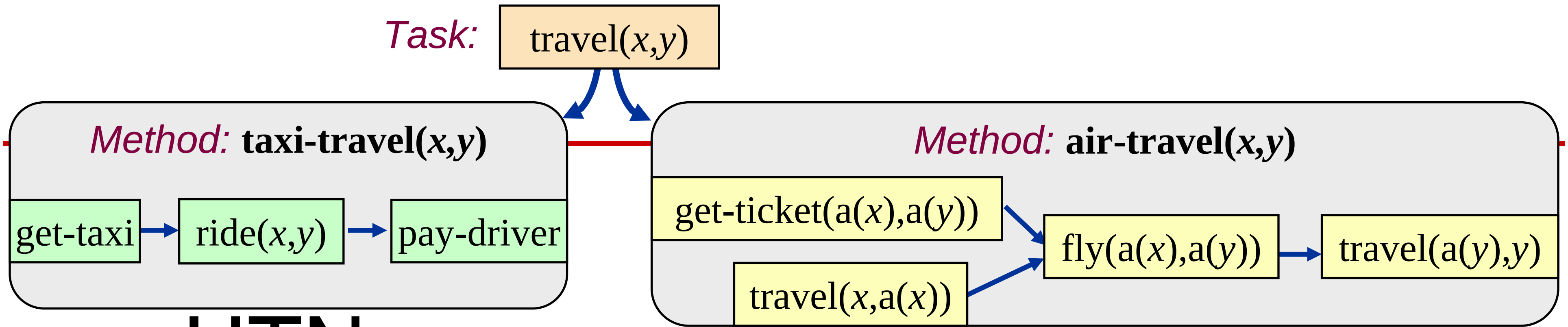
Office Hours (in HEC 232):
T & Th 10:30am-12

Reading

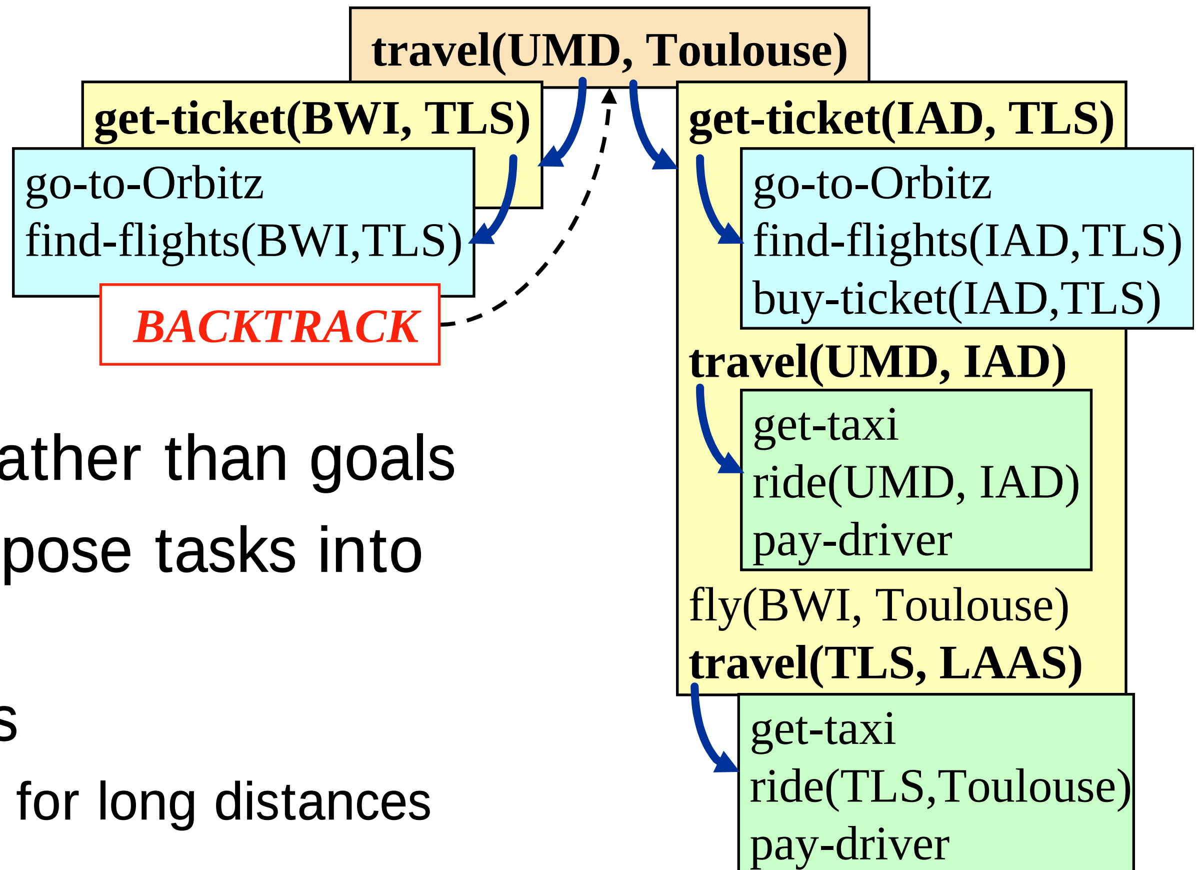
- Reading**:** Amy Greenwald and Peter Stone, Autonomous Bidding Agents in the Trading Agent Competition IEEE Internet Computing, 5(2):52.60, March/April 2001.

Two Approaches to Planning

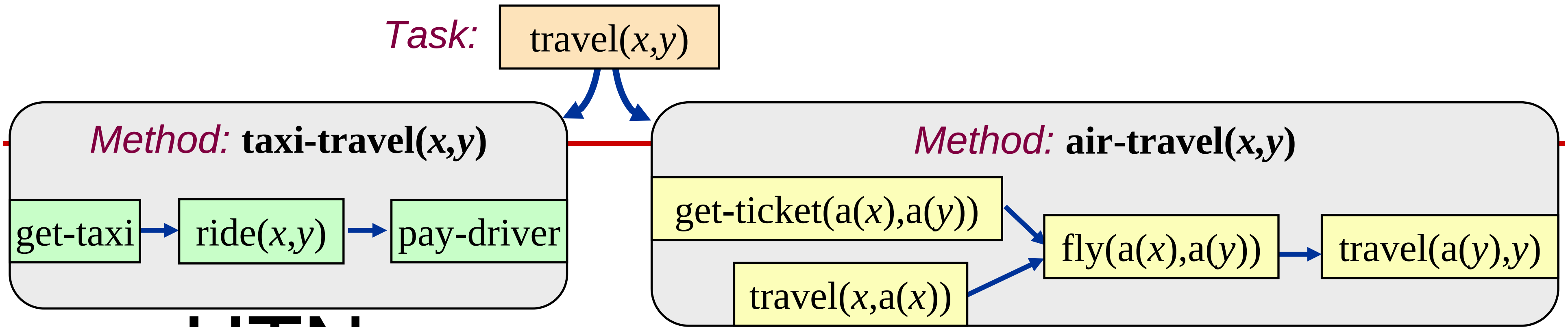
- Control rules:
  - Write rules to prune every action that *doesn't* fit the recipe
- Hierarchical Task Network (HTN) planning:
  - Describe the actions and subtasks that *do* fit the recipe



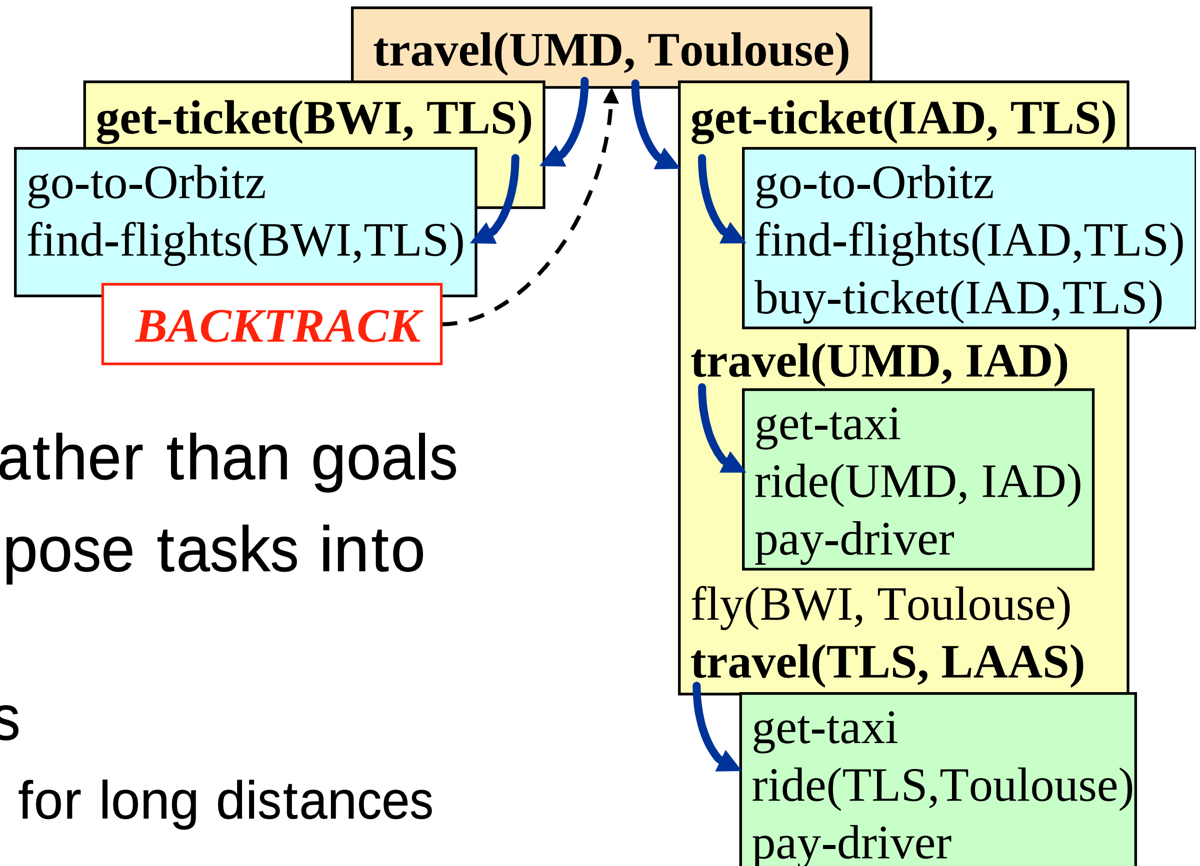
HTN Planning



- Problem reduction
  - *Tasks* (activities) rather than goals
  - *Methods* to decompose tasks into subtasks
  - Enforce constraints
    - E.g., taxi not good for long distances
  - Backtrack if necessary



HTN Planning



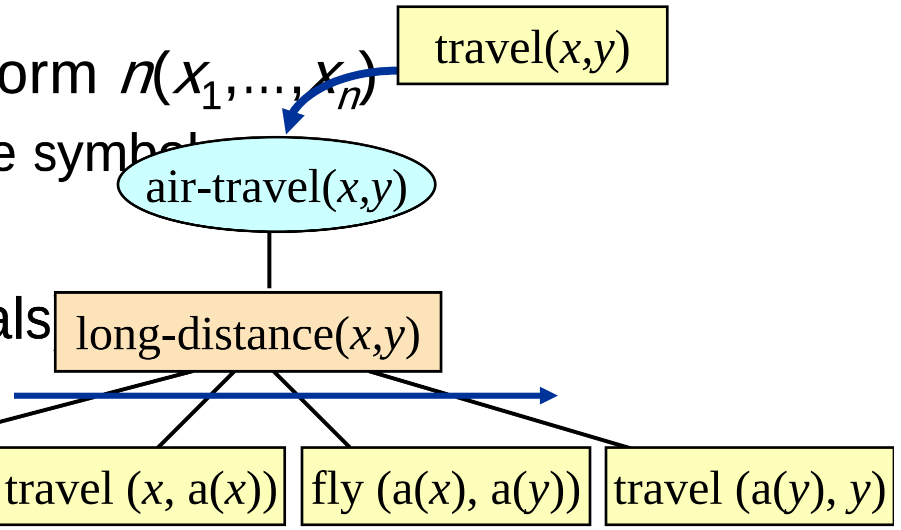
- Problem reduction
  - *Tasks* (activities) rather than goals
  - *Methods* to decompose tasks into subtasks
  - Enforce constraints
    - E.g., taxi not good for long distances
  - Backtrack if necessary

Simple Task Network (STN)

- A special case of HTN planning
- States and operators
  - The same as in classical planning
- *Task*: an expression of the form $t(u_1,\ldots,u_n)$
  - t is a *task symbol*, and each u_i is a term
  - Two kinds of task symbols (and tasks):
    - *primitive*: tasks that we know how to execute directly
      - task symbol is an operator name
    - *nonprimitive*: tasks that must be decomposed into subtasks
      - use *methods* (next slide)

Totally Ordered Method

- Totally ordered method: a 4-tuple
  m = (name(m), task(m), precond(m), subtasks(m))
  - name(m): an expression of the form $n(x_1,\ldots,x_n)$
    - $x_1,\ldots,x_n$ are parameters - variable symb
  - task(m): a nonprimitive task
  - precond(m): preconditions (literals
  - subtasks(m): a sequence of tasks $\langle t_1, \ldots,$



air-travel(x,y)

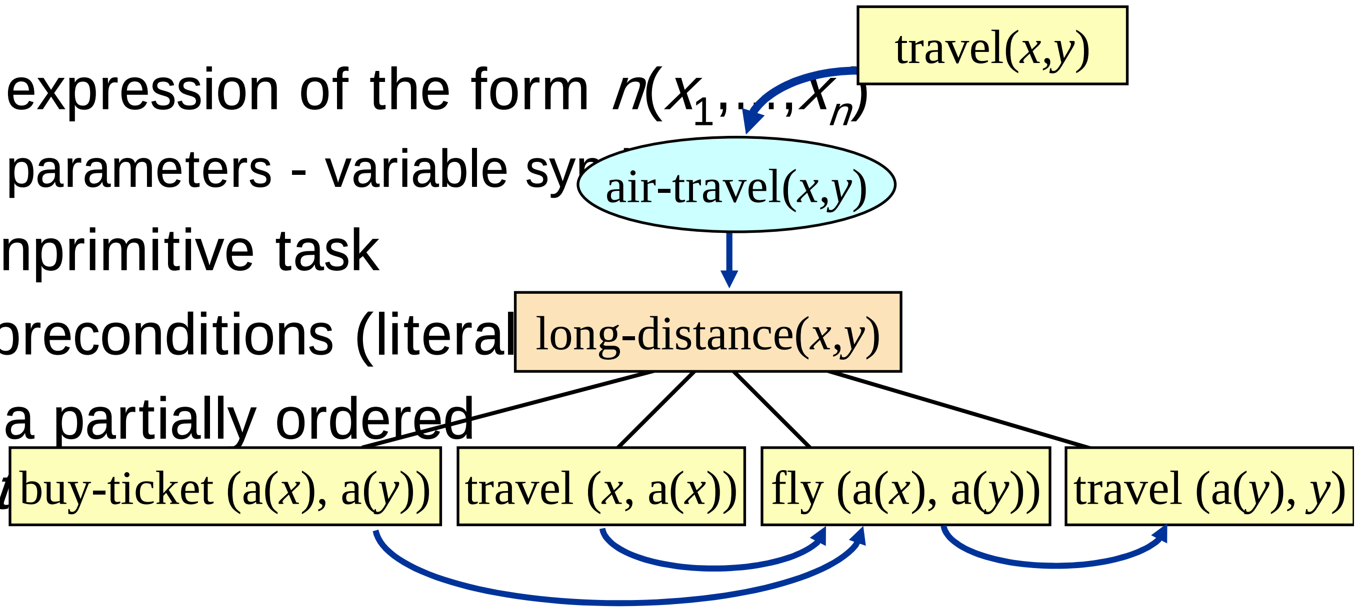
task: travel(x,y)

precond: long-distance(x,y)

subtasks: $\langle$buy-ticket($a(x), a(y)$), travel($x,a(x)$), fly($a(x), a(y)$), travel($a(y),y$)$\rangle$

Partially Ordered Methods

- Partially ordered method: a 4-tuple
  m = (name(m), task(m), precond(m), subtasks(m))
  - name(m): an expression of the form $n(x_1,\ldots,x_n)$
    - $x_1,\ldots,x_n$ are parameters - variable sy
  - task(m): a nonprimitive task
  - precond(m): preconditions (literal
  - subtasks(m): a partially ordered set of tasks {t



air-travel(x,y)

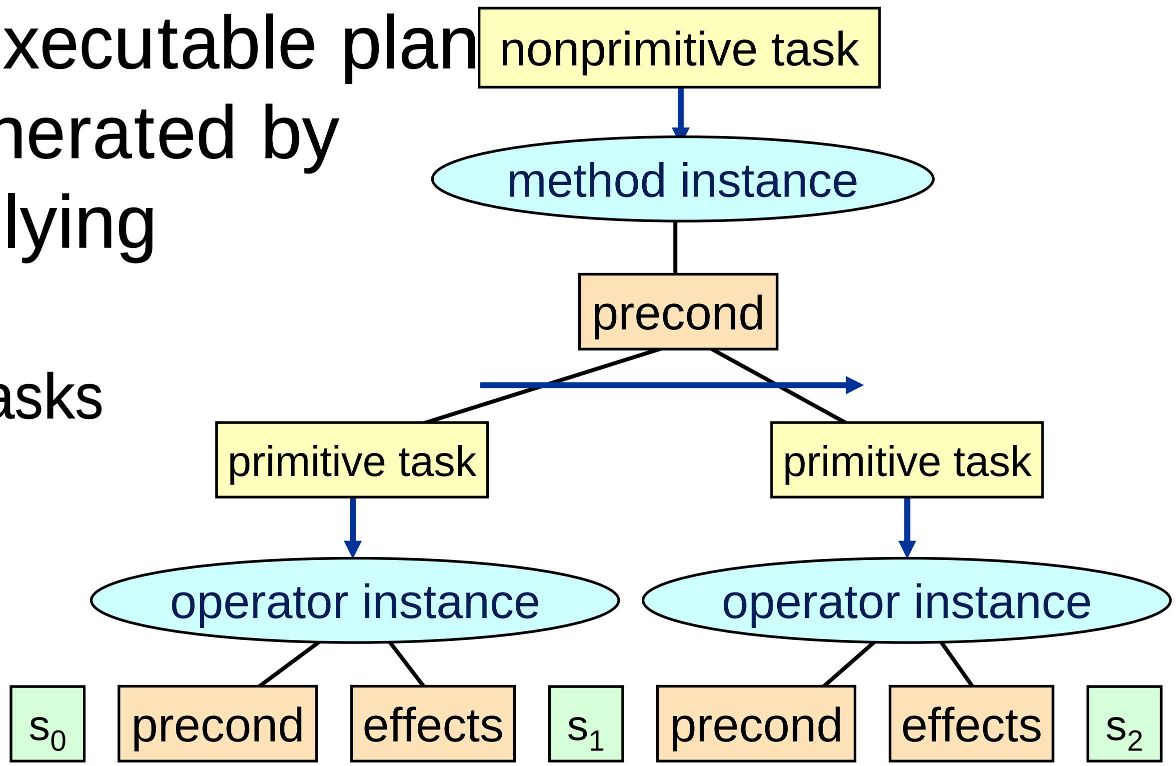
task: travel(x,y)

precond: long-distance(x,y)

network: u_1=buy-ticket($a(x),a(y)$), u_2= travel($x,a(x)$), u_3= fly($a(x)$, $a(y)$) u_4= travel($a(y),y$), {(u_1,u_3), (u_2,u_3), (u_3 ,u_4)}

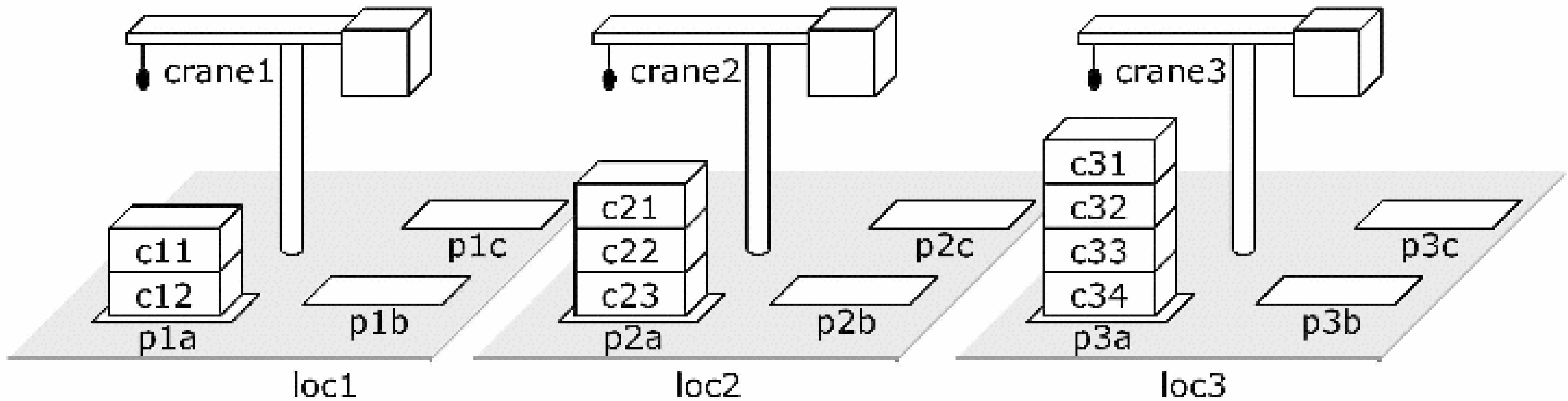
Domains, Problems, Solutions

- STN planning domain: methods, operators
- STN planning problem: methods, operators, initial state, task list
- Solution: any executable plan that can be generated by recursively applying
  - methods to nonprimitive tasks
  - operators to primitive tasks

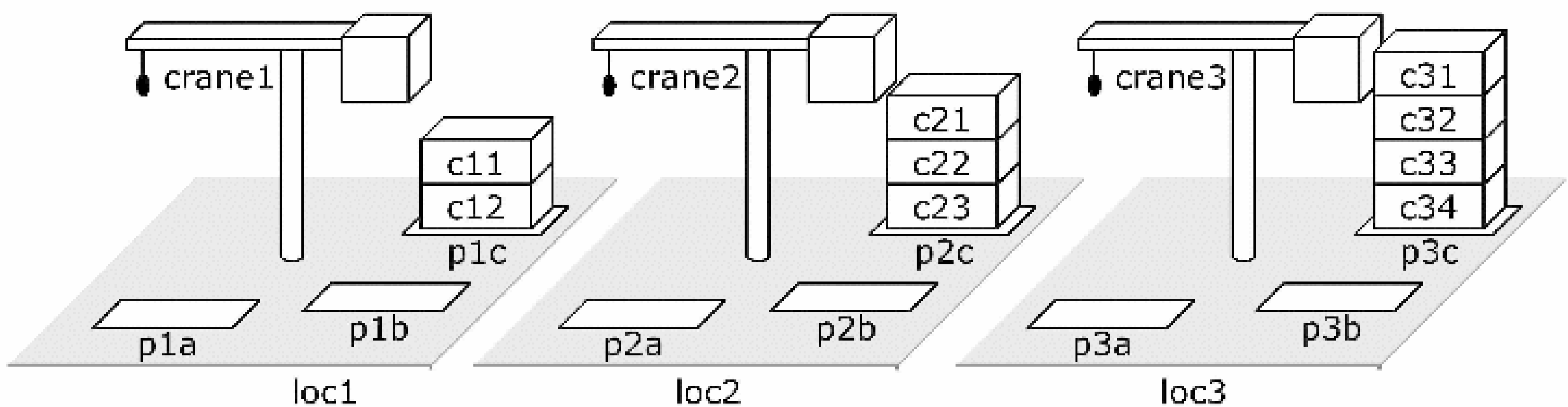


Example

- Suppose we want to move three stacks of containers while preserving container order



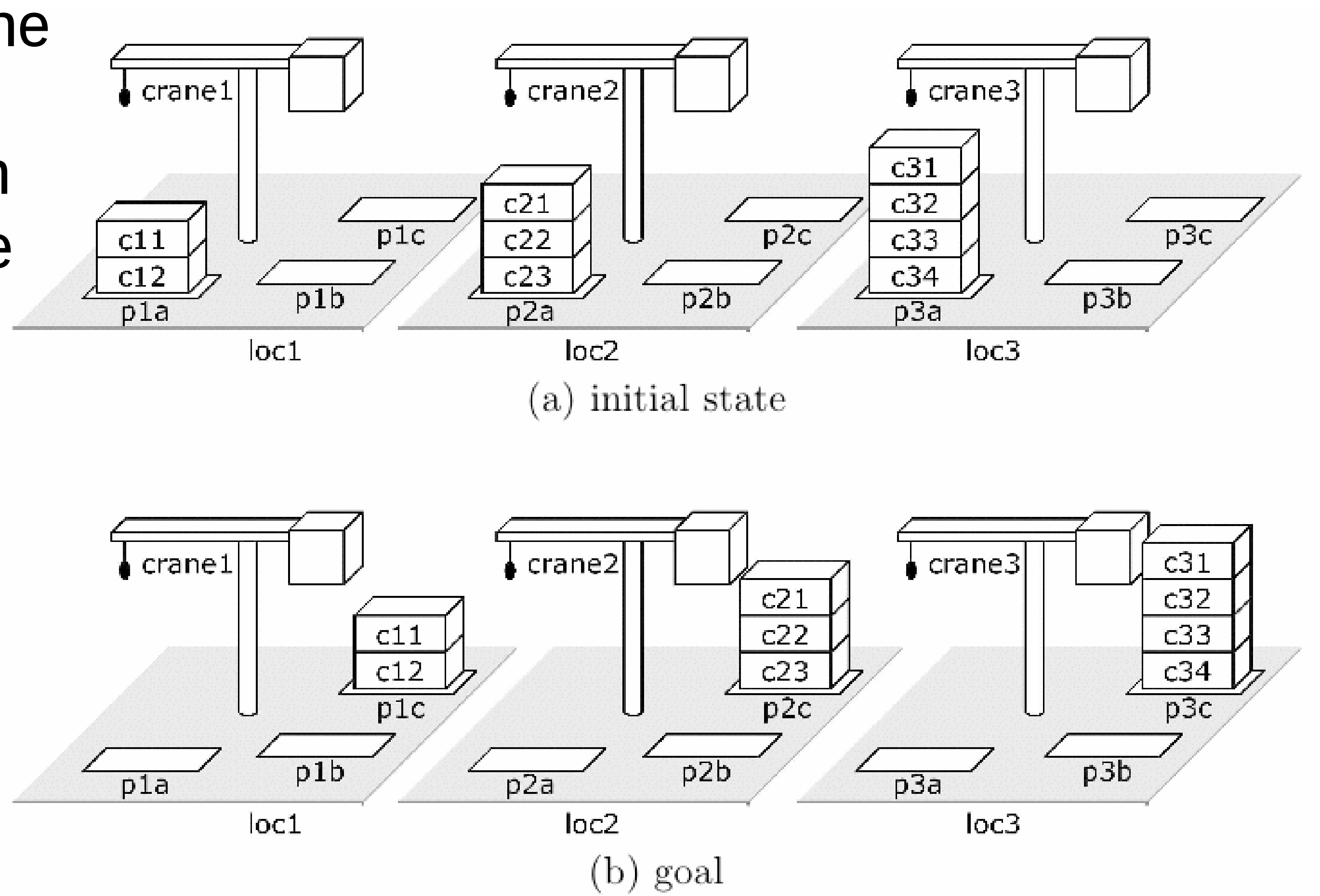
(a) initial state



(b) goal

Example (continued)

- A way to move each stack:
  - first move the containers from *p* to an intermediate pile *r*
  - then move them from *r* to *q*



(a) initial state

(b) goal

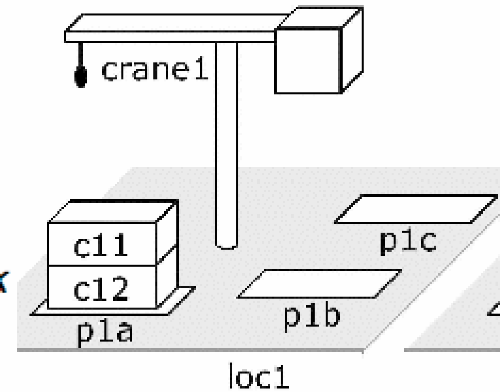
Total-Order Formulation

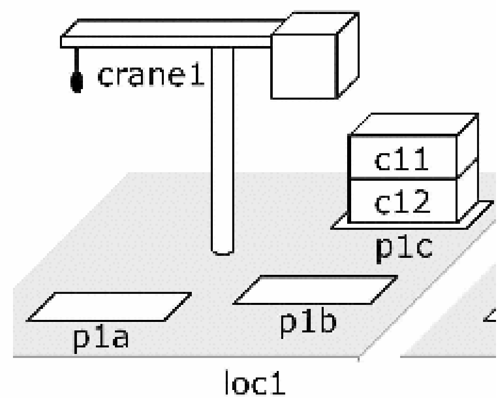
take-and-put$(c, k, l_1, l_2, p_1, p_2, x_1, x_2)$:
- task: move-topmost-container(p_1, p_2)
- precond: top(c, p_1), on(c, x_1), *; true if p_1 is not empty*
  attached(p_1, l_1), belong(k, l_1), *; bind l_1 and k*
  attached(p_2, l_2), top(x_2, p_2) *; bind l_2 and x_2*
- subtasks: ⟨take(k, l_1, c, x_1, p_1), put(k, l_2, c, x_2, p_2)⟩

recursive-move(p, q, c, x):
- task: move-stack(p, q)
- precond: top(c, p), on(c, x) *; true if p is not empty*
- subtasks: ⟨move-topmost-container(p, q), move-stack(p, q)⟩
  ;; the second subtask recursively moves the rest of the stack

do-nothing(p, q)
- task: move-stack(p, q)
- precond: top$(pallet, p)$ *; true if p is empty*
- subtasks: ⟨⟩ *; no subtasks, because we are done*

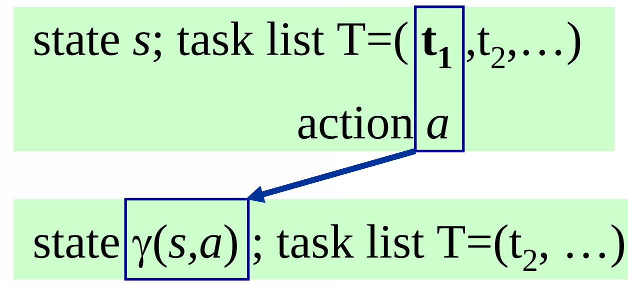
move-each-twice()
- task: move-all-stacks()
- precond: *; no preconditions*
- subtasks: *; move each stack twice:*
  ⟨move-stack(p1a,p1b), move-stack(p1b,p1c),
  move-stack(p2a,p2b), move-stack(p2b,p2c),
  move-stack(p3a,p3b), move-stack(p3b,p3c)⟩

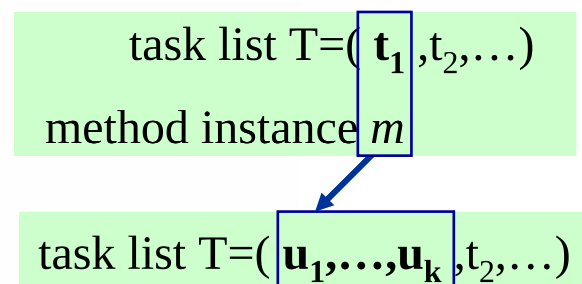




Solving Total-Order STNs

- TFD($s, \langle t_1, \ldots, t_k \rangle, O, M$)
    - if $k = 0$ then return $\langle\rangle$ (i.e., the empty plan)
    - if t_1 is primitive then
        - *active* $\leftarrow \{(a, \sigma) \mid a$ is a ground instance of an operator in O, σ is a substitution such that a is relevant for $\sigma(t_1)$, and a is applicable to $s\}$
        - if *active* $= \emptyset$ then return failure
        - nondeterministically choose any $(a, \sigma) \in$ *active*
        - $\pi \leftarrow$ TFD($\gamma(s, a), \sigma(\langle t_2, \ldots, t_k \rangle), O, M$)
        - if π = failure then return failure
        - else return $a.\pi$
    - else if t_1 is nonprimitive then
        - *active* $\leftarrow \{m \mid m$ is a ground instance of a method in M, σ is a substitution such that m is relevant for $\sigma(t_1)$, and m is applicable to $s\}$
        - if *active* $= \emptyset$ then return failure
        - nondeterministically choose any $(m, \sigma) \in$ *active*
        - $w \leftarrow$ subtasks$(m).\sigma(\langle t_2, \ldots, t_k \rangle)$
        - return TFD(s, w, O, M)





SHOP2 includes the following

- SHOP is very similar to the STN planner
- SHOP2 includes the following extensions
  - Partially-ordered tasks
  - Quantifiers
  - Axioms to specify preconditions
  - Conditional effects
  - Search criterion to use when satisfying a method’s preconditions
  - Extensions for temporal planning

Learning and Planning

- What kind of things would we like our planner to be able to learn?

What would we like to learn?

- Learn macro-operators
- Learning search control knowledge
- Learn task hierarchies
- Learn plan abstraction
- Transfer learning (this paper)

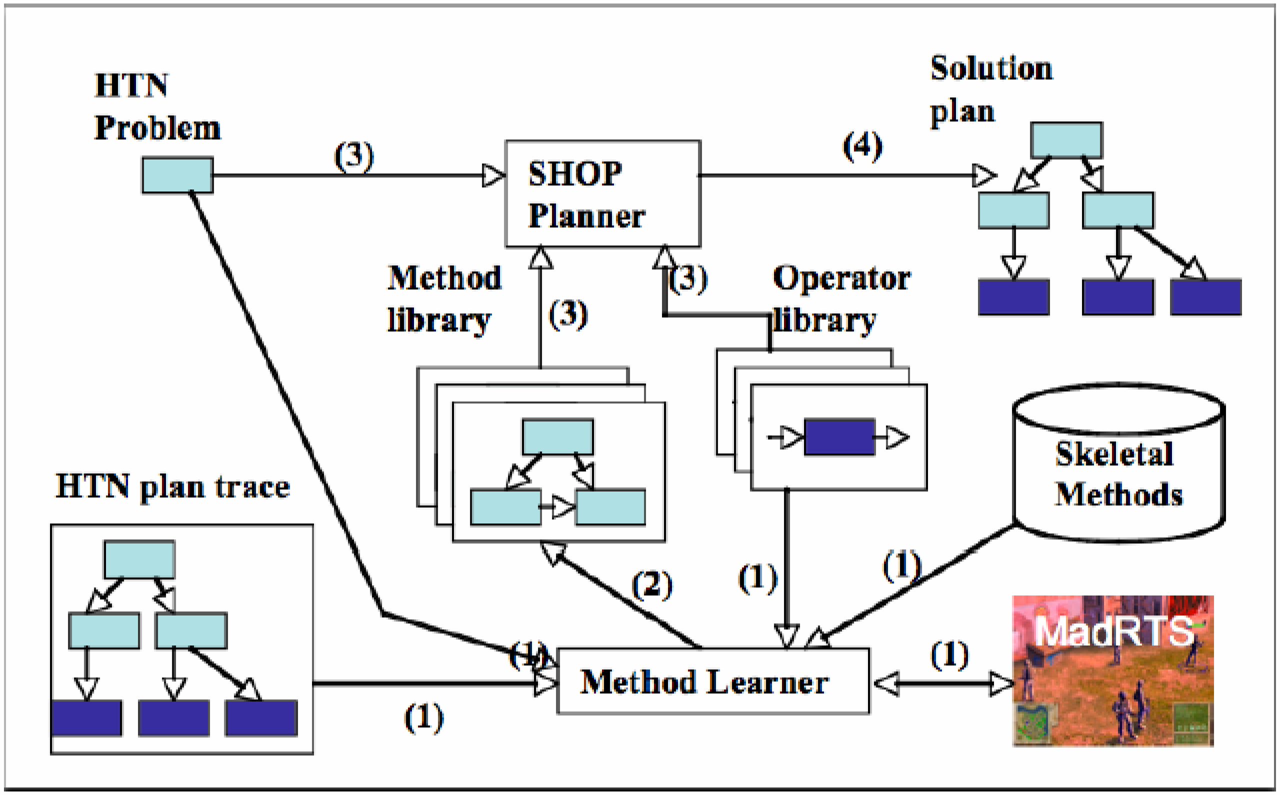
Learn2SHOP

- Learn how to play different scenarios in a real-time strategy game
- Uses HTN traces from games that were successful at a scenario
- Learns what the necessary preconditions are to apply a method

Concept Learning

- Candidate learning:
  - Use training examples to determine which hypotheses are applicable
- Version space: set of all hypotheses which correctly label examples
  - G: most general hypothesis set consistent with examples
  - S: set of most specific hypotheses consistent with examples
  - Hypotheses are in the form of fact sets that represent method preconditions
  - Search through hypotheses by reintroducing variables into the state

System Architecture



Results

- Evaluated system on performance measures
  - Success rate
  - Jump start (advantage of transfer knowledge on 1st trial)
  - Transfer ratio (overall advantage of transferred knowledge on whole training set)
  - Asymptotic advantage (advantage of transferred knowledge in last trial)