Applications of planning, Hierarchical Task Network

Jiří Vokřínek

A4M36PAH - 30.3.2015

Motivation

- Example: travel to a destination that’s far away:
  - Domain-independent planner:
    - many combinations of vehicles and routes

Motivation

- Example: travel to a destination that’s far away:
  - Domain-independent planner:
    - many combinations of vehicles and routes

  - Experienced human: small number of “recipes”

    e.g., flying:

    1. buy ticket from local airport to remote airport
    2. travel to local airport
    3. fly to remote airport
    4. travel to final destination

Motivation

- Example: travel to a destination that's far away:
  - Domain-independent planner:
    - many combinations of vehicles and routes
  - Experienced human: small number of "recipes"

    e.g., flying:

    1. buy ticket from local airport to remote airport
    2. **travel to local airport**
    3. fly to remote airport
    4. **travel to final destination**

Motivation

- Example: travel to a destination that's far away:
  - Domain-independent planner:
    - many combinations of vehicles and routes
  - Experienced human: small number of "recipes"

    e.g., flying:

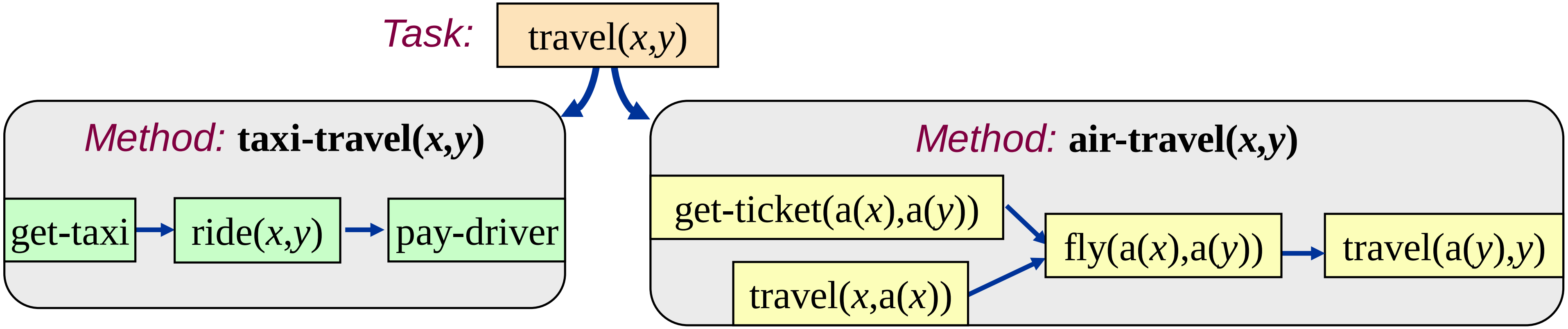
    1. buy ticket from local airport to remote airport
    2. **travel to local airport**
    3. fly to remote airport
    4. **travel to final destination**

Motivation

- Hierarchical Task Network (HTN)
  - Classical planning representation – states (set of atoms) and actions (deterministic state transition)
  - HTN differs in approach – set of ***tasks*** instead of set of ***goals***
  - ***Non-primitive*** (compound) vs. ***primitive*** tasks
  - ***Methods*** – prescriptions to decompose a ***task*** into ***sub-tasks***
  - Widely used for practical applications (intuitive representation)

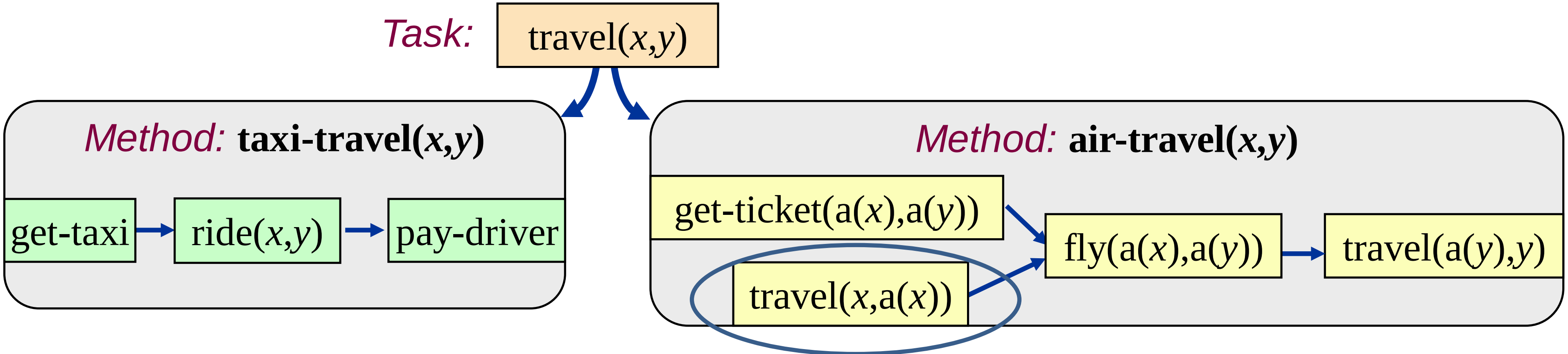
HTN Planning

- 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

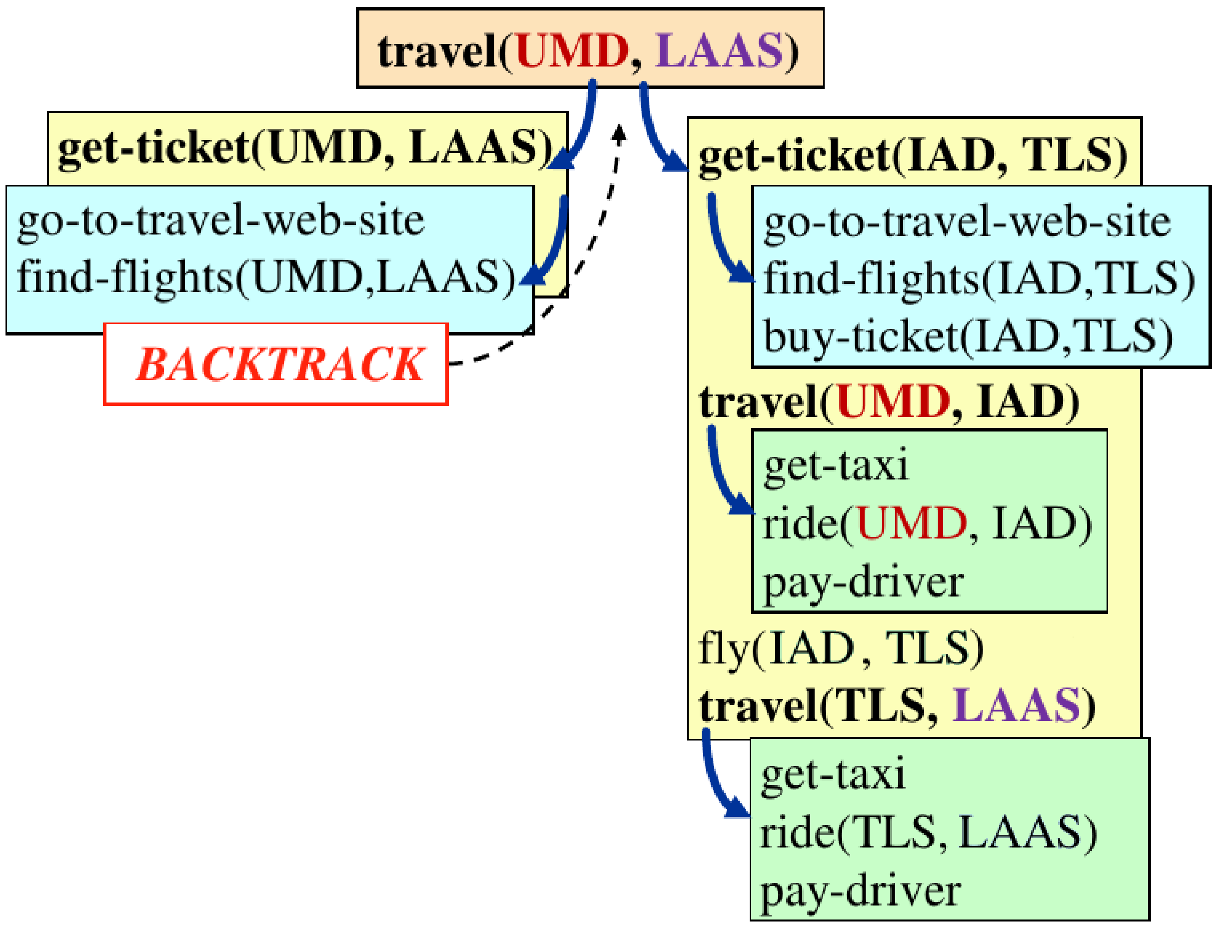


HTN Planning

- 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



HTN Planning



HTN Planning

- Objective: perform a given set of tasks
- Input includes:
  - Set of operators
  - Set of methods: recipes for decomposing a complex task into more primitive subtasks
- Planning process:
  - Decompose non-primitive tasks recursively until primitive tasks are reached

Simple Task Network (STN)

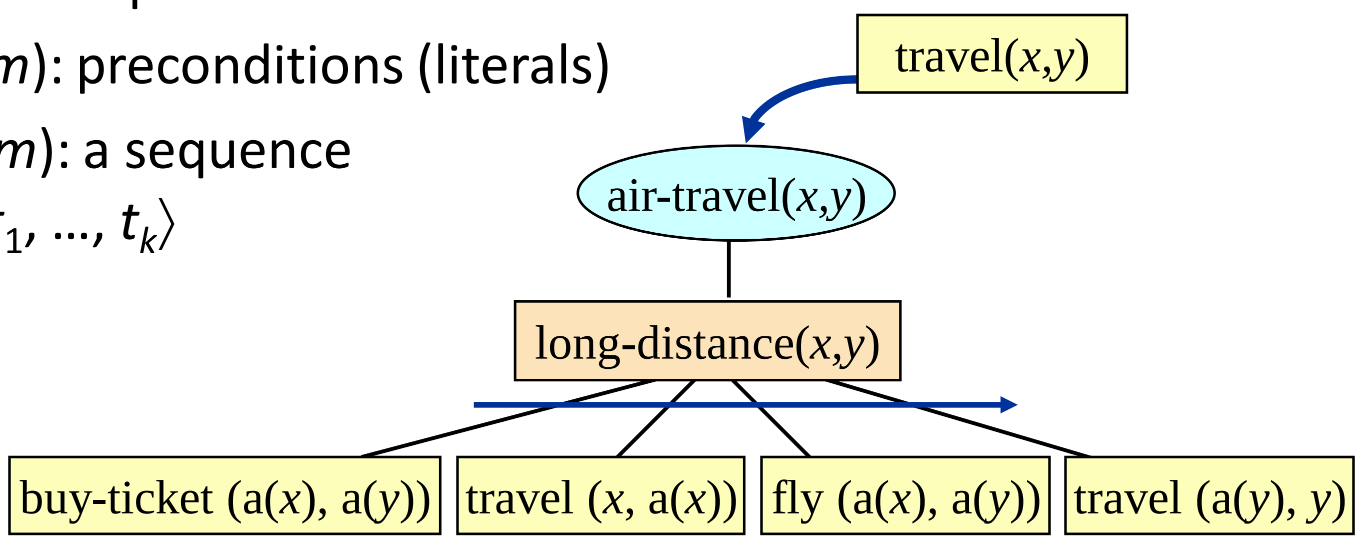
- A special case of HTN planning
- States and operators
  - The same as in classical planning
- *Task*: an expression of the form $t(u_1,\ldots,u_n)$
  - t is a ***task symbol***, and each u_i is a term
  - Two kinds of task symbols (and tasks):
    - ***primitive***: tasks that we know how to execute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

Methods

- 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 symbols
  - task(m): a non-primitive task
  - precond(m): preconditions (literals)
  - subtasks(m): a sequence of tasks $\langle t_1, \ldots, t_k \rangle$



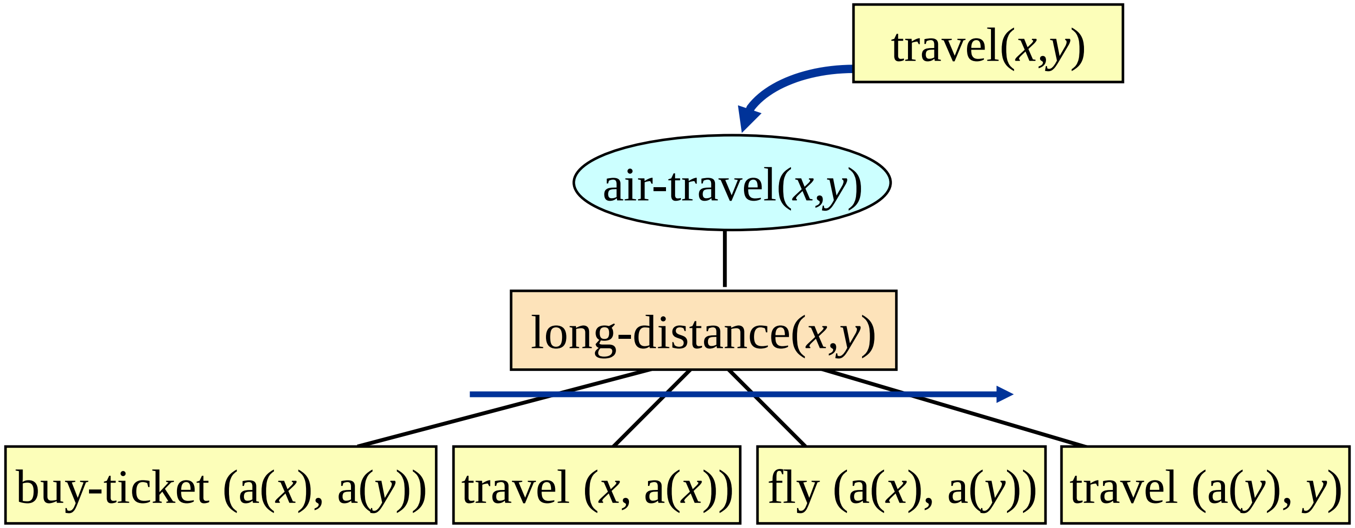
Methods

air-travel(x,y)

task: travel(x,y)

precond: long-distance(x,y)

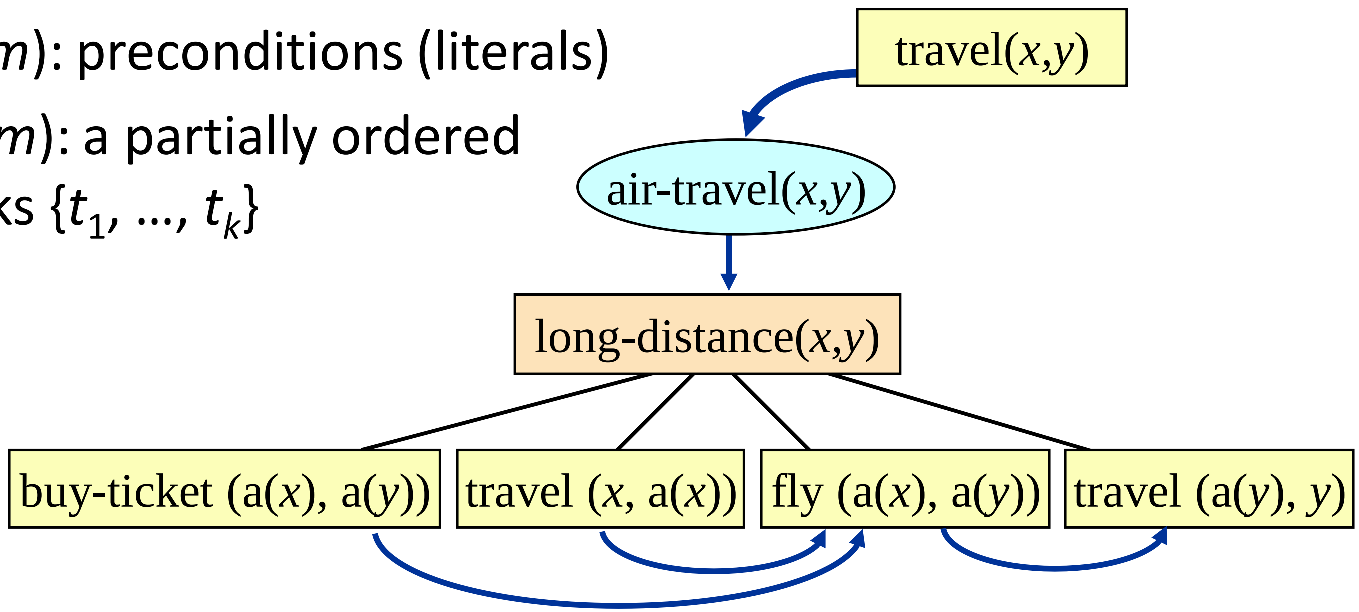
subtasks: $\langle$buy-ticket($a(x), a(y)$), travel($x,a(x)$), fly($a(x), a(y)$), travel($a(y),y$)$\rangle$



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 symbols
  – task(m): a nonprimitive task
  – precond(m): preconditions (literals)
  – subtasks(m): a partially ordered set of tasks $\{t_1, \ldots, t_k\}$



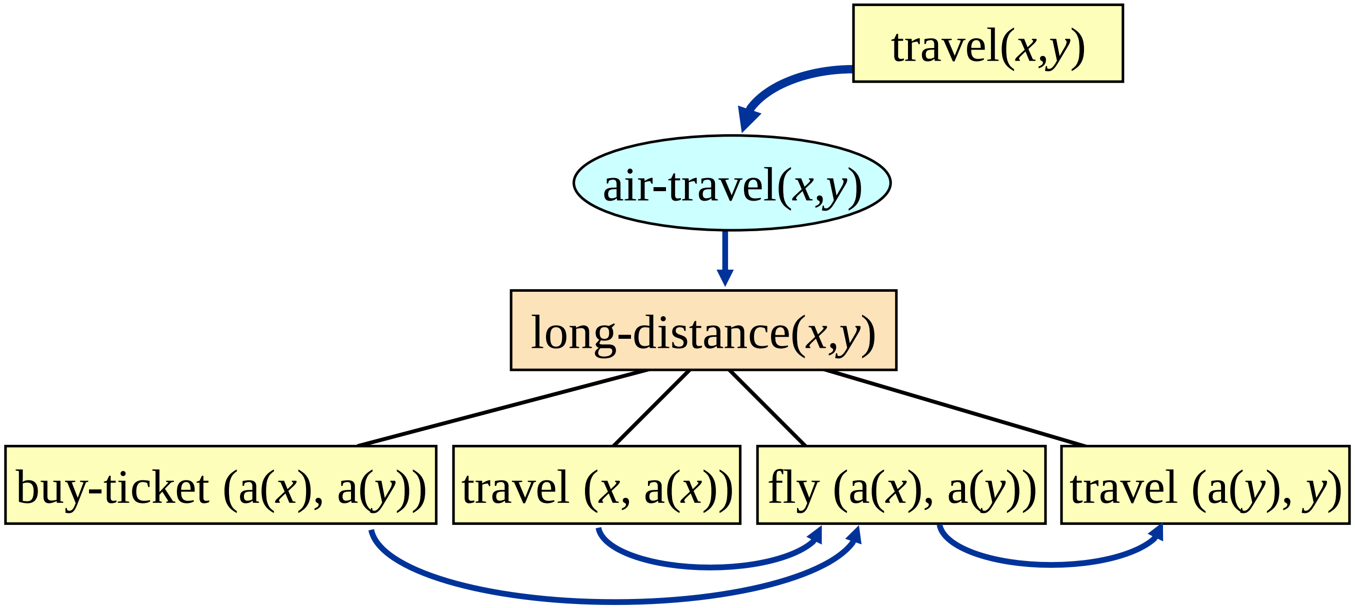
Methods

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($a(y),y$), $\{(u_1,u_3), (u_2,u_3), (u_3, u_4)\}$



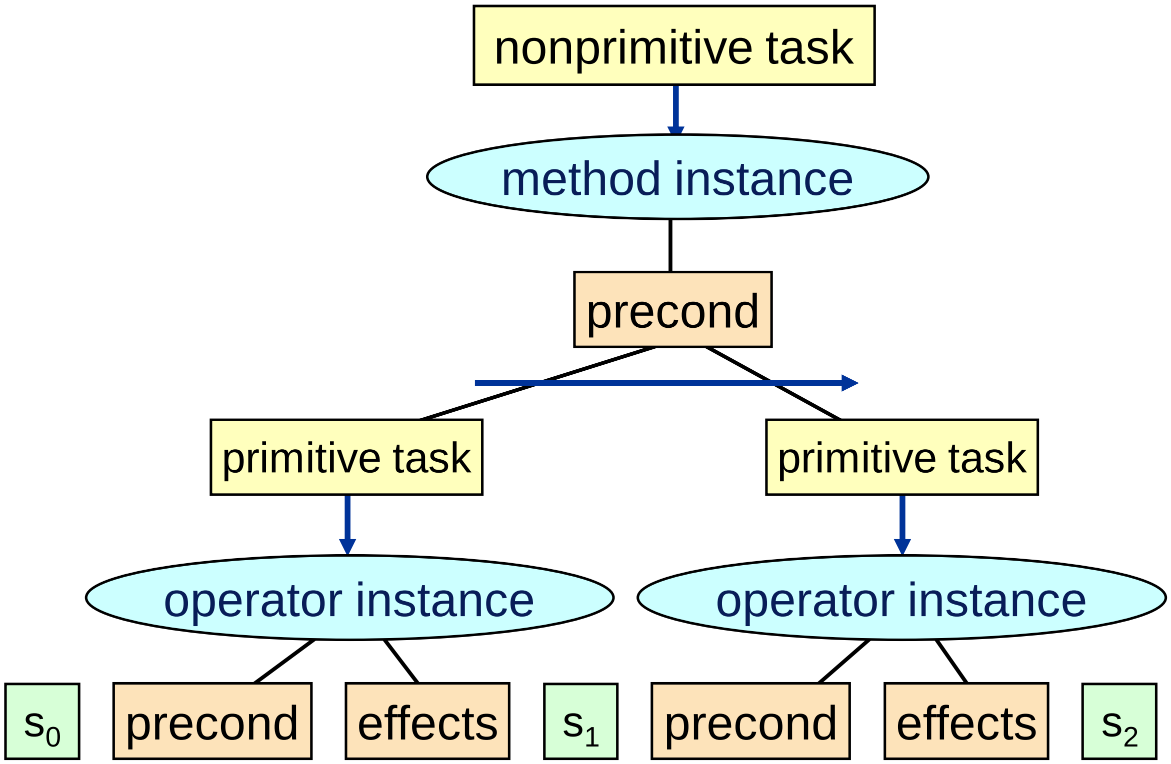
Domains, Problems, Solutions

- STN planning domain: methods, operators
- STN planning problem: methods, operators, initial state, task list
- Total-order STN planning domain and planning problem:
  - Same as above except that all methods are totally ordered

Domains, Problems, Solutions

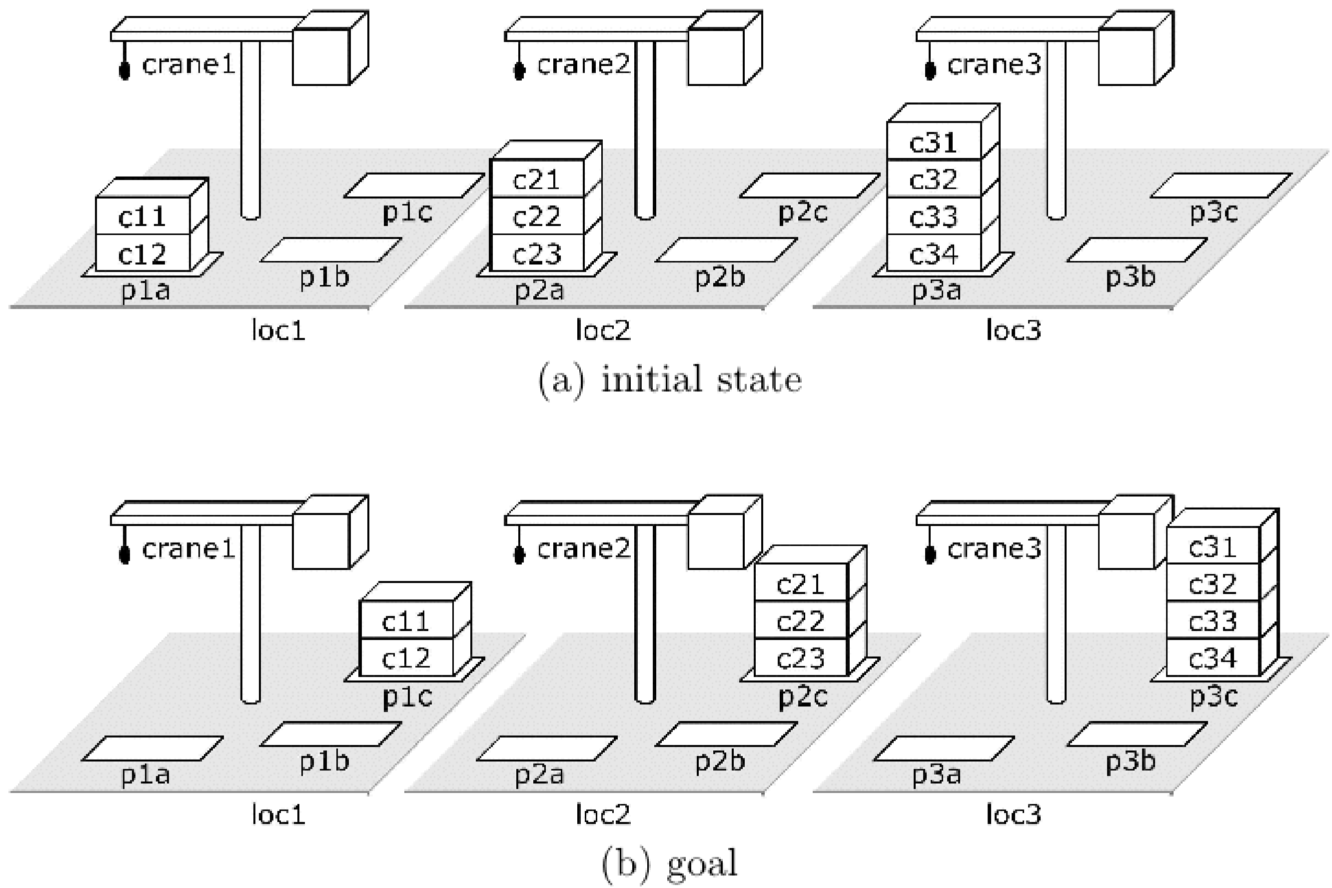
- STN planning domain: methods, operators
- STN planning problem: methods, operators, initial state, task list
- Total-order STN planning domain and planning problem:
  - Same as above except that all methods are totally ordered
- Solution: any executable plan that can be generated by recursively applying
  - Methods to non-primitive tasks
  - Operators to primitive tasks

Domains, Problems, Solutions



DWR Stack Moving Example

- Suppose we want to move three stacks of containers in a way that preserves the order of the containers



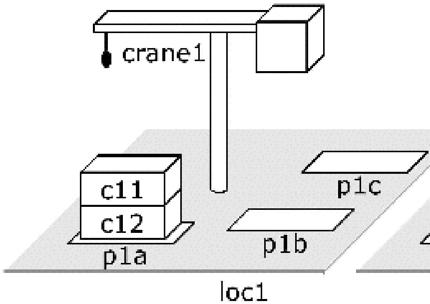
(a) initial state

(b) goal

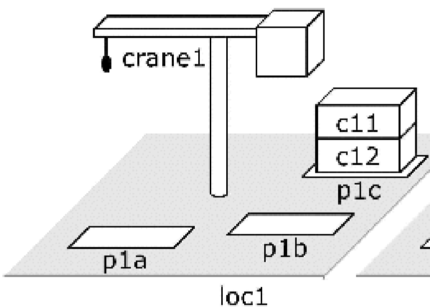
Total-Order Formulation

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: $\langle$take(k, l_1, c, x_1, p_1), put$(k, l_2, c, x_2, p_2)\rangle$

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: $\langle$move-topmost-container(p, q), move-stack$(p, q)\rangle$
    ;; 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: $\langle\rangle$ *; 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)⟩

Partial-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: $\langle$take(k, l_1, c, x_1, p_1), put$(k, l_2, c, x_2, p_2)\rangle$

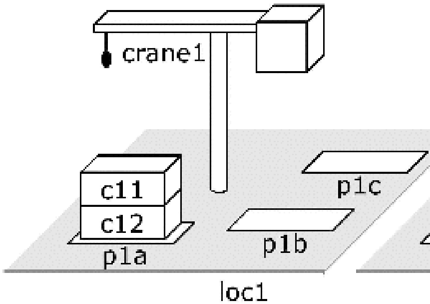
recursive-move(p, q, c, x):

task: move-stack(p, q)

precond: top(c, p), on(c, x) *; true if p is not empty*

subtasks: $\langle$move-topmost-container(p, q), move-stack$(p, q)\rangle$

;; the second subtask recursively moves the rest of the stack

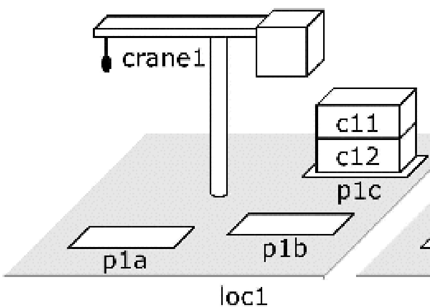


do-nothing(p, q)

task: move-stack(p, q)

precond: top$(pallet, p)$ *; true if p is empty*

subtasks: $\langle\rangle$ *; no subtasks, because we are done*



move-each-twice()

task: move-all-stacks()

precond: *; no preconditions*

network: *; move each stack twice:*

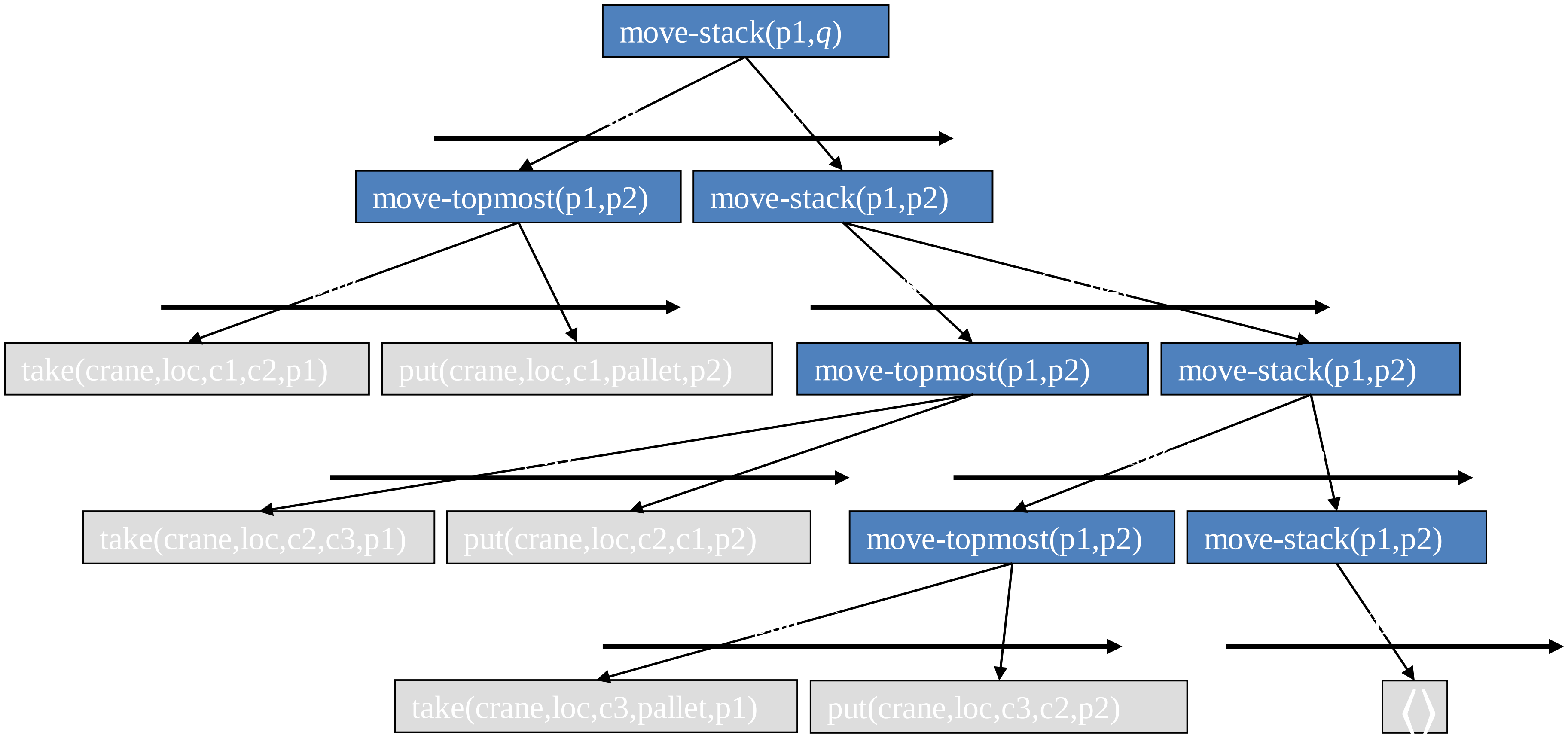
u_1 =move-stack(p1a,p1b), u_2 =move-stack(p1b,p1c),

u_3 =move-stack(p2a,p2b), u_4 =move-stack(p2b,p2c),

u_5 =move-stack(p3a,p3b), u_6 =move-stack(p3b,p3c),

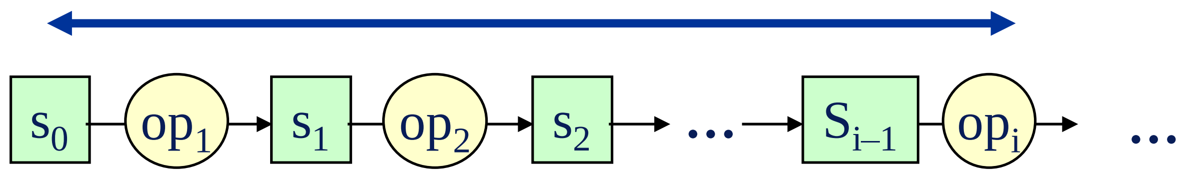
$\{(u_1, u_2), (u_3, u_4), (u_5, u_6)\}$

Decomposition Tree: DWR Example

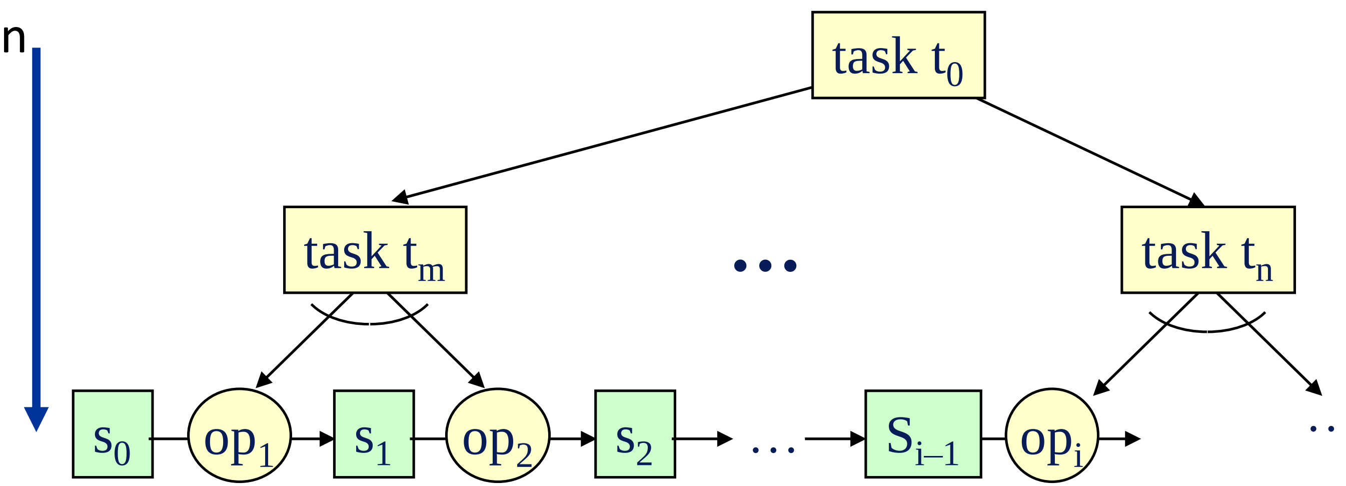


Comparison to F/B Search

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

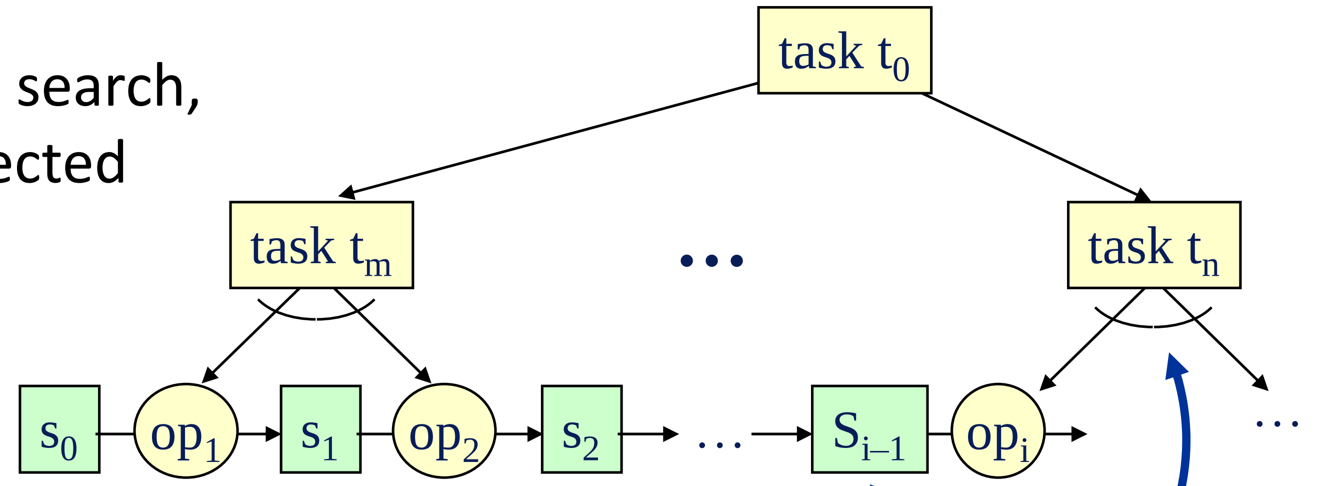


- In HTN planning, there are *two* choices to make about direction:
  - forward or backward
  - up or down



- TFD goes *down* and *forward*

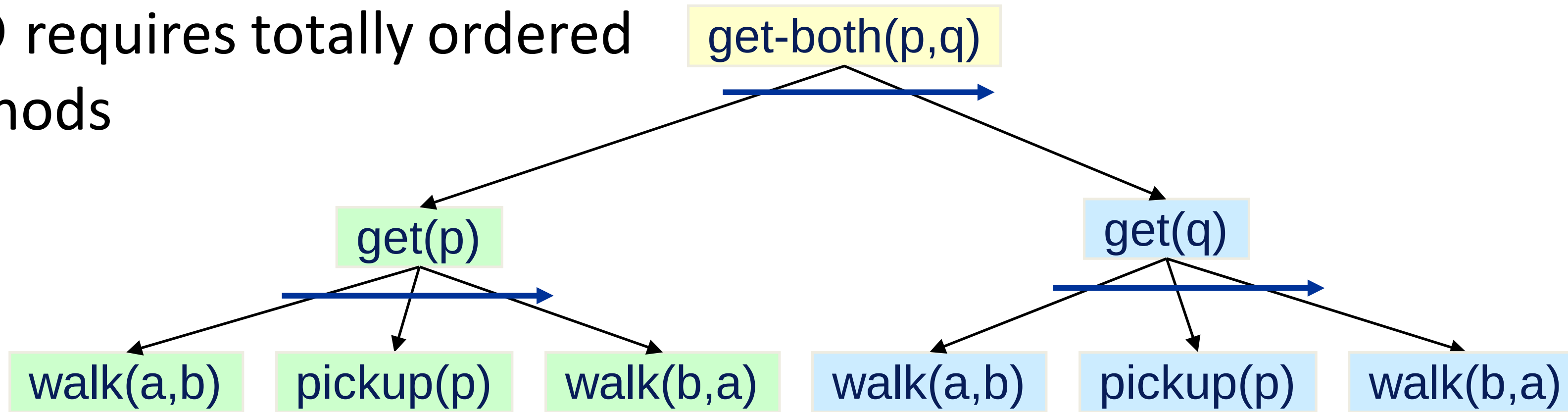
Comparison to F/B Search



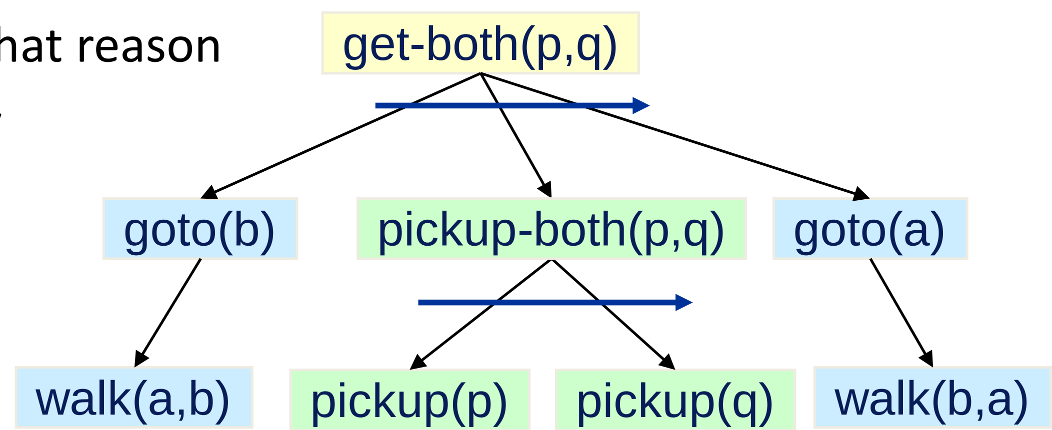
- Like a backward search, TFD is goal-directed
  - Goals correspond to tasks
- Like a forward search, it generates actions in the same order in which they'll be executed
- Whenever we want to plan the next task
  - We've already planned everything that comes before it
  - Thus, we know the current state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

Limitation of Ordered-Task Planning

- TFD requires totally ordered methods

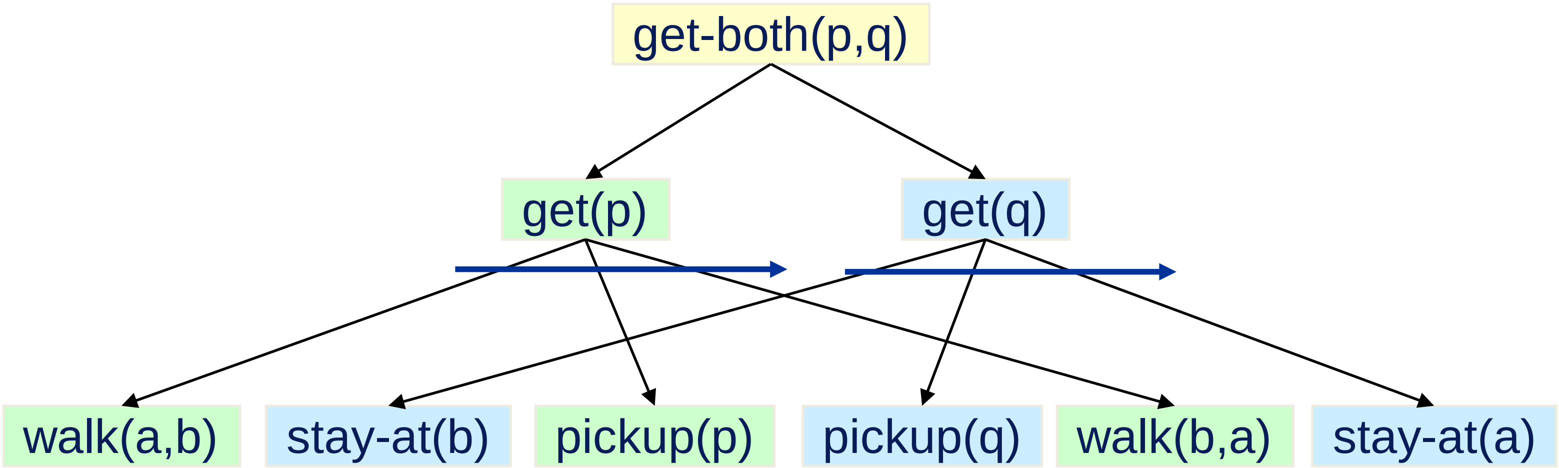


- Can’t interleave subtasks of different tasks
- Sometimes this makes things awkward
  - Need to write methods that reason globally instead of locally



Partially Ordered Methods

- With partially ordered methods, the subtasks can be interleaved



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

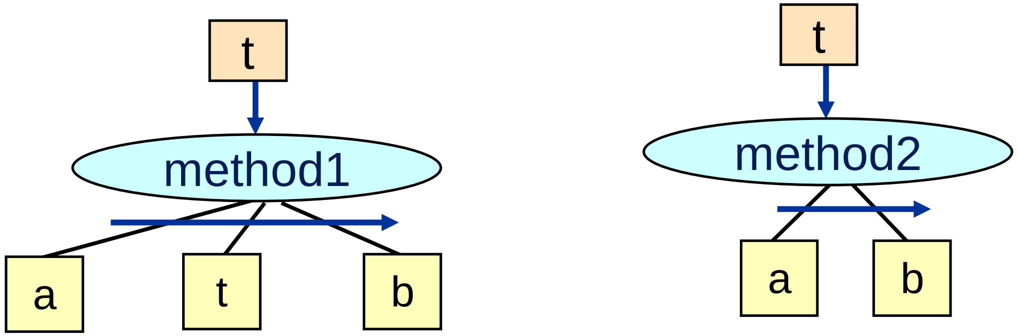
Comparison to Classical Planning

STN planning is strictly more expressive than classical planning

- Any classical planning problem can be translated into an ordered-task-planning problem in polynomial time
- Several ways to do this. One is roughly as follows:
  - For each goal or precondition *e*, create a task t_e
  - For each operator o and effect *e*, create a method $m_{o,e}$
    - Task: t_e
    - Subtasks: $t_{c1}, t_{c2}, \ldots, t_{cn}, o,$ where $c_1, c_2, \ldots, c_n$ are the preconditions of *o*
    - Partial-ordering constraints: each t_{ci} precedes *o*

Comparison to Classical Planning

- Some STN planning problems aren't expressible in classical planning
- Example:
  - Two STN methods:
    - No arguments
    - 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

Example

method travel-by-foot
  precond: $distance(x, y) \leq 2$
  task: travel(a, x, y)
  subtasks: walk(a, x, y)

method travel-by-taxi
  task: travel(a, x, y)
  precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
  subtasks: $\langle$call-taxi(a, x), ride(a, x, y), pay-driver$(a, x, y)\rangle$

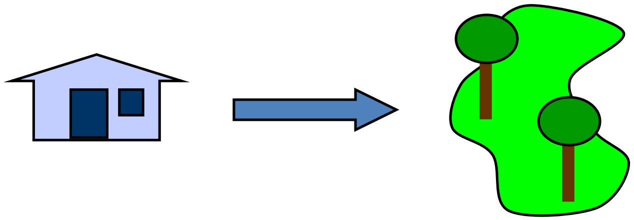
operator walk
  precond: $location(a) = x$
  effects: $location(a) \leftarrow y$

operator call-taxi(a, x)
  effects: $location(taxi) \leftarrow x$

operator ride-taxi (a, x)
  precond: $location(taxi) = x$, $location(a) = x$
  effects: $location(taxi) \leftarrow y$, $location(a) \leftarrow y$

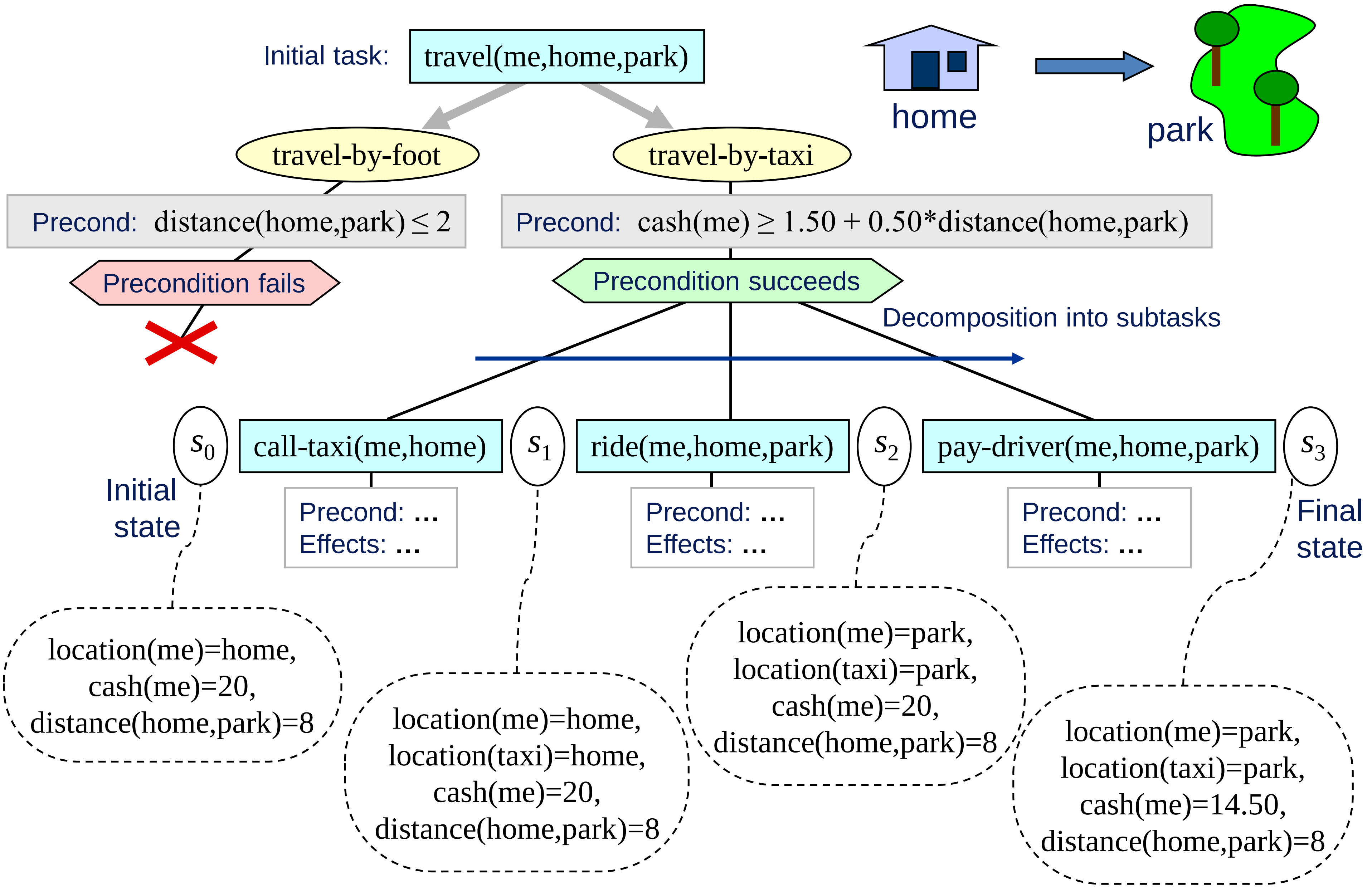
operator pay-driver(a, x, y)
  precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
  effects: $cash(a) \leftarrow cash(a) - 1.5 - 0.5 \times distance(x, y)$

- Simple travel-planning domain
  - State-variable formulation
- Planning problem:
  - I'm at home, I have \$20
  - Want to go to a park 8 miles away



  - s_0 = {location(me) = home, cash(me) = 20, distance(home,park) = 8}
  - t_0 = travel(me,home,park)

Example, Continued



HTN Planning

- STN planning constraints:
  - ordering constraints: maintained in network
  - preconditions:
    - enforced by planning procedure
    - must know state to test for applicability
    - must perform forward search
- HTN planning can be even more general
  - Can have constraints associated with tasks and methods
    - Things that must be true before, during, or afterwards
  - Some algorithms use causal links and threats like those in PSP

Methods in STN

- Let M_S be a set of method symbols. An ***STN method*** is a 4-tuple m=(name(m),task(m),precond(m),network(m)) where:
  - name(m):
    - the name of the method
    - syntactic expression of the form $n(x_1,\ldots,x_k)$
      - $n \in M_S$: unique method symbol
      - $x_1,\ldots,x_k$: all the variable symbols that occur in m;
  - task(m): a non-primitive task;
  - precond(m): set of literals called the method's preconditions;
  - network(m): task network (U,E) containing the set of ***subtasks*** U of m

Methods in HTN

- Let M_S be a set of method symbols. An ***HTN method*** is a 4-tuple m=(name(m),task(m),subtasks(m),constr(m)) where:
  - name(m):
    - the name of the method
    - syntactic expression of the form $n(x_1,\ldots,x_k)$
      - $n \in M_S$: unique method symbol
      - $x_1,\ldots,x_k$: all the variable symbols that occur in m;
  - task(m): a non-primitive task;
  - (subtasks(m),constr(m)): a task network.

STN Methods: DWR Example (1)

- move topmost: take followed by put action
- take-and-put(c,k,l,p_o,p_d,x_o,x_d)
  - task: move-topmost(p_o,p_d)
  - precond: top(c,p_o), on(c,x_o), attached(p_o,l), belong(k,l), attached(p_d,l), top(x_d,p_d)
  - subtasks: ⟨take(k,l,c,x_o,p_o),put(k,l,c,x_d,p_d)⟩

HTN Methods: DWR Example (1)

- move topmost: take followed by put action
- take-and-put(c,k,l,p_o,p_d,x_o,x_d)
  - task: move-topmost(p_o,p_d)
  - network:
    - subtasks: {t_1=take(k,l,c,x_o,p_o), t_2=put(k,l,c,x_d,p_d)}
    - constraints: {$t_1 \prec t_2$, before({t_1}, top(c,p_o)), before({t_1}, on(c,x_o)), before({t_1}, attached(p_o,l)), before({t_1}, belong(k,l)), before({t_2}, attached(p_d,l)), before({t_2}, top(x_d,p_d))}

STN Methods: DWR Example (2)

- move stack: repeatedly move the topmost container until the stack is empty
- recursive-move(p_o,p_d,c,x_o)
  - task: move-stack(p_o,p_d)
  - precond: top(c,p_o), on(c,x_o)
  - subtasks: ⟨move-topmost(p_o,p_d), move-stack(p_o,p_d)⟩
- no-move(p_o,p_d)
  - task: move-stack(p_o,p_d)
  - precond: top(pallet,p_o)
  - subtasks: ⟨⟩

HTN Methods: DWR Example (2)

- move stack: repeatedly move the topmost container until the stack is empty
- recursive-move(p_o,p_d,c,x_o)
  - task: move-stack(p_o,p_d)
  - network:
    - subtasks: {t_1=move-topmost(p_o,p_d), t_2=move-stack(p_o,p_d)}
    - constraints: {$t_1 \prec t_2$, before({t_1}, top(c,p_o)), before({t_1}, on(c,x_o))}
- move-one(p_o,p_d,c)
  - task: move-stack(p_o,p_d)
  - network:
    - subtasks: {t_1=move-topmost(p_o,p_d)}
    - constraints: {before({t_1}, top(c,p_o)), before({t_1}, on(c,pallet))}

Some Planning Features

- Expansion of a high level abstract plan into greater detail where necessary.

- High level ‘chunks’ of procedural knowledge at a human scale - typically 5-8 actions - can be manipulated within the system.

- Ability to establish that a feasible plan exists, perhaps for a range of assumptions about the situation, while retaining a high level overview.

- Analysis of potential interactions as plans are expanded or developed.