Lecture 23 of 42

Planning:
More Hierarchical Task Networks (HTN), Sensorless & Conditional Planning

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:

Sections 12.4 – 12.8, p. 441– 454, Russell & Norvig 2nd edition
HTN Planning: http://en.wikipedia.org/wiki/Hierarchical_task_network
SAT Solvers: http://en.wikipedia.org/wiki/SAT_solver

Lecture Outline

- **Reading for Next Class: Sections 12.5 – 12.8 (p. 441 – 454), R&N 2e**
- **Last Class: Sections 11.4 – 11.7 (p. 395 – 408), R&N 2e**
  - **Graph planning (11.4)**
  - **Planning with propositional logic (11.5)**
  - **Analysis of planning approaches (11.6)**
  - **Summary (11.7)**
- **Today: Real-World Planning Systems, 12.1 – 12.4 (p. 417 – 440), R&N 2e**
  - **Time (12.1)**
  - **HTN Planning (12.2)**
  - **Nondeterminism (12.3)**
  - **Conditional planning (12.4)**
- **Next Class: Robust Planning Concluded**
  - **Practical planning: monitoring and replanning, continual planning (Ch. 12)**
  - **Need for representation language for uncertainty**
- **Next Week: Uncertain Reasoning and KR**

Acknowledgements

Héctor Muñoz-Avila
Associate Professor
Department of Computer Science & Engineering (CSE)
Lehigh University

Hai Hoang (not pictured)
Graduate Teaching Assistant, CSE
Lehigh University

http://bit.ly/2nUghl

Gerhard Wickler
Research Fellow
Artificial Intelligence Applications Institute
Centre for Intelligent Systems and their Applications
University of Edinburgh

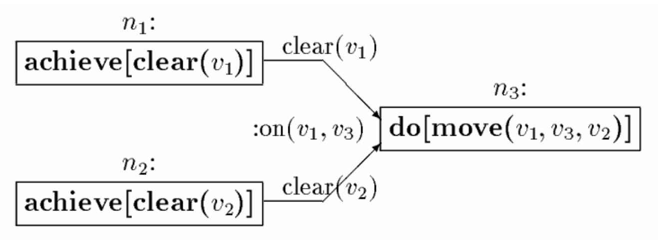
http://bit.ly/Diir0

Austin Tate
Director,
Artificial Intelligence Applications Institute
University of Edinburgh

Frame, Qualification, and Ramification Problems – Review

- **Frame Problem: Need to Describe and Propagate Non-Action**
  - **Representational** – proliferation of frame axioms: *e.g.*, in Wumpus World
    - ⇨ SHOOT doesn't clobber HOLDINGGOLD
    - ⇨ MOVENORTH doesn't clobber HAVEARROW (precondition for SHOOT)
  - **Inferential** – copying state: HOLDINGGOLD (S_3) $\rightarrow_{\text{SHOOT}}$ HOLDINGGOLD (S_4)
- **Qualification Problem: Specifying All Preconditions ("Exceptions")**
  - "Action A is possible *unless*..."
  - Improbable operator failures
- **Ramification Problem: Specifying All Effects ("Side Effects")**
  - "Action A *also* causes…"
  - Small incremental changes (*e.g.*, "wear and tear"), *aka* "butterflies in China"
- **Solution Approaches**
  - Representational FP: successor state axioms, graph/propositional planning
  - Inferential FP: defeasible reasoning (*e.g.*, defaults)
  - Qualification problem: abstraction; reaction; replanning
  - Ramification problem: defaults, abstraction

Hierarchical Task Network Planning: Review



$$\begin{array}{l}[(n_1 : achieve[clear(v_1)])(n_2 : achieve[clear(v_2)])(n_3 : do[move(v_1, v_3, v_2)]) \\ (n_1 \prec n_3) \land (n_2 \prec n_3) \land (n_1, clear(v_1), n_3) \land (n_2, clear(v_2), n_3) \land (on(v_1, v_3), n_3) \\ \land \neg(v_1 = v_2) \land \neg(v_1 = v_3) \land \neg(v_2 = v_3)]\end{array}$$

© 2003 José Luis Ambite, ISI   http://bit.ly/3ldmiM

HTN *vs.* Classical Planning [1]: Similarities

- **Each state of the world is represented by a set of atoms**
- **Each action corresponds to a deterministic state transition**
- **(block b1) (block b2) (block b3) (block b4) (on-table b1) (on b2 b1) (clear b2) (on-table b3) (on b4 b3) (clear b4)**

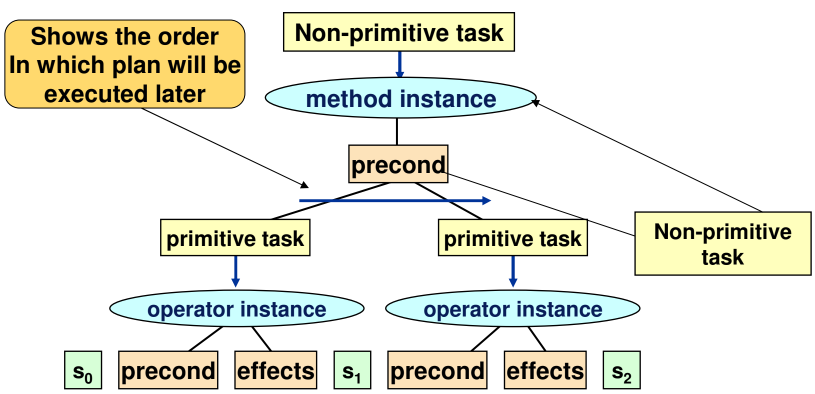
Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN *vs.* Classical Planning [2]: Differences

- Objective: to perform a set of tasks not a set of goals
- Terms, literals, operators, actions, plans have same meaning as classical planning.
- Added tasks, methods, task networks
- Tasks decompose into subtasks
  - Constraints
  - Backtrack if necessary

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghI

Decomposition



Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghI

Control Rules

- Classical planning efficiency suffers from combinatorial complexity (intractable)
- Prune function detects and cuts unpromising nodes
  - Can improve solving
  - Exponential to polynomial
- $\Phi_1(c,d,p) = [GOAL(in(c,p)) \vee \neg\exists q\ GOAL(in(c,q))] \wedge [GOAL(on(c,d) \vee \neg\exists e\ GOAL(on(c,e))]$
  - No goal requiring c in another pile or on top of something else (prune if exist?)
  - Holds if acceptable when container c is on item d in pile p

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN Methods

- Instead of detecting and cutting unpromising nodes
- HTN methods are only applied only when the preconditions are satisfied.

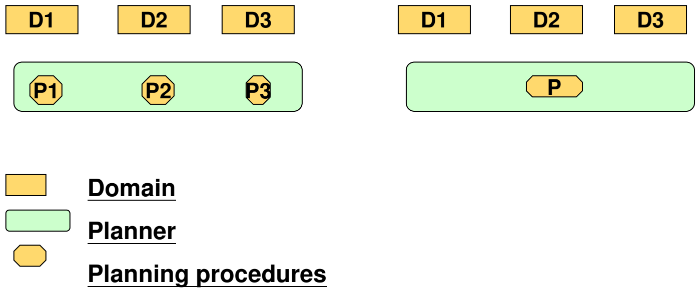
```
(:method (pick-up ?x)
   ((clear ?x))
   ((!pick-up ?x))
)
```

Only pick up if x is on top

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

Planning Domains & Domain Independence

- Domain dependent – Bridge Baron game
- Domain independent – SHOP



Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN Specification

- Domain consists of
  - Methods
  - Operators (SHOP: axioms)
- Problem consists of
  - Domain
  - Initial state
  - Initial task network
    - tasks to accomplish
    - some ordering of the tasks defined
- Solution
  - Plan: Totally ordered collection of primitive tasks (SHOP)
  - General HTN planner - partially ordered collection of primitive tasks

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN Tasks

- *Task*: an expression of the form $t(u_1,\ldots,u_n)$
  - *t* is a *task symbol*, and each u_i is a term (variable, constant, function expression (f t_1 t_2 t_n)
  - (move-block ?nomove)
  - (move-block (list ?x . ?nomove))
- Two types of task
  1. Non-primitive (compound) – decomposed into subtasks.
  2. Primitive – cannot be decomposed, know how to perform directly (task name is the operator name).
     - (!drive-truck ?truck ?loc-from ?loc-to)

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN Methods & Operators [1]: Definition

- Defined slightly differently in textbook, but we're more concerned with coding it in SHOP so forget book for now (book notations in other slides)
- Explain both with an example instead of notations
  - Spent a good amount of time arranging the next slide
  - Help to visualize how they map to a real shop method or operator.
- method as defined by SHOP (see manual)
  (:method h [n_1] C_1 T_1 [n_2] C_2 T_2 … [n_k] C_k T_k)
  - h method head – task atom with no call terms
  - [n_1] OPTIONAL name for succeeding C_i T_i pair
  - C_1 conjunct or tagged conjunct? Precondition list??
  - T1 task list
- Operator
  (:operator h P D A)
  - h head – primitive task atom with no call terms
  - P precondition list (logical atoms)
  - D delete list (logical atoms)
  - A add list (logical atoms)

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN Methods & Operators [2]: Example

Non prim

- *Method (decomposes into subtasks)*

  (:method (drive-truck ?truck ?loc-from ?loc-to)

  ((same ?loc-from ?loc-to))

  ((!do-nothing))

  ()

  ((!drive-truck ?truck ?loc-from ?loc-to)))

  - ✷ *Notice the if else structure*
  - ✷ *Invoke non-primitive task: (drive-truck ?t ?x ?y)*

Preconditions

Primitive task

Subtask list

- *Operator (achieves PRIMITIVE TASKS)*

  (:operator (!drive-truck ?truck ?locfrom ?locto)

  ()

  ((truck-at ?truck ?locfrom))

  ((truck-at ?truck ?locto)))

  Invoke primitive task: (!drive-truck ?t ?l1 ?l2)

Preconditions

Prim task

Delete list

Add list

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN Planning Algorithm (TFD): Pseudocode

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)

Applying an operator
Changing the state

Randomly pick an applicable method

Decompose method into tasks

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

HTN: Tasks

- task symbols: $T_S = \{t_1,\ldots,t_n\}$
  - operator names $\subsetneq T_S$: primitive tasks
  - non-primitive task symbols: T_S - operator names
- task: $t_i(r_1,\ldots,r_k)$
  - t_i: task symbol (primitive or non-primitive)
  - $r_1,\ldots,r_k$: terms, objects manipulated by the task
  - ground task: are ground
- action *a* accomplishes ground primitive task $t_i(r_1,\ldots,r_k)$ in state *s* iff
  - name(a) = t_i and
  - *a* is applicable in *s*

Adapted from slide © 2005 G. Wickler & A. Tate
University of Edinburgh
http://bit.ly/Diir0

HTN: Simple Task Networks

- A simple task network *w* is an acyclic directed graph (*U*,*E*) in which
  - the node set $U = \{t_1,\ldots,t_n\}$ is a set of tasks and
  - the edges in *E* define a partial ordering of the tasks in *U*.
- A task network w is ground/primitive if all tasks $t_u \in U$ are ground/primitive, otherwise it is unground/non-primitive.

Adapted from slide © 2005 G. Wickler & A. Tate
University of Edinburgh
http://bit.ly/Diir0

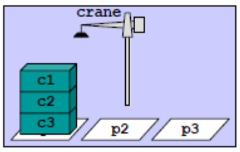
Simple Task Networks: DWR Example

- tasks:
  - t_1 = take(crane,loc,c1,c2,p1): primitive, ground
  - t_2 = take(crane,loc,c2,c3,p1): primitive, ground
  - t_3 = move-stack(p1,q): non-primitive, unground
- task networks:
  - $w_1 = (\{t_1,t_2,t_3\}, \{(t_1,t_2), (t_1,t_3)\})$
    - partially ordered, non-primitive, unground
  - $w_2 = (\{t_1,t_2\}, \{(t_1,t_2)\})$
    - totally ordered: $w_2 = \langle t_1,t_2 \rangle$, ground, primitive
    - $\pi(w_2)$ = ⟨take(crane,loc,c1,c2,p1),take(crane,loc,c2,c3,p1)⟩

Adapted from slide © 2005 G. Wickler & A. Tate
University of Edinburgh
http://bit.ly/Diir0

DWR – Crane Domain Example [1]: Stack Moving Problem Description

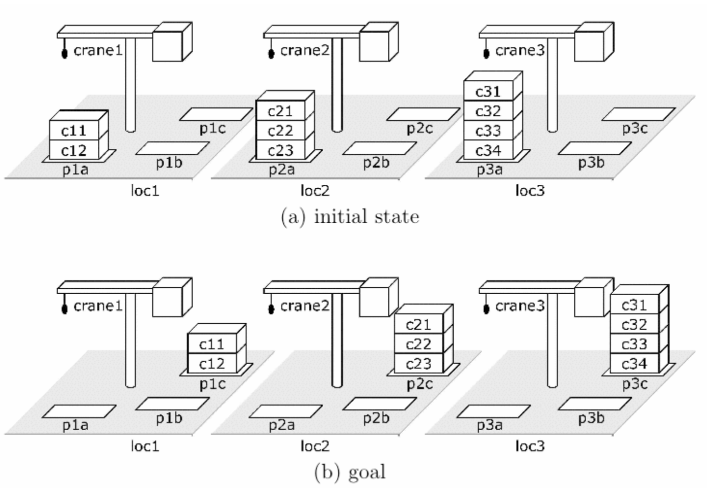
- task: move stack of containers from pallet p1 to pallet p3 in a way the preserves the order



- (informal) methods:
  - move via intermediate: move stack to intermediate pile (reversing order) and then to final destination (reversing order again)
  - move stack: repeatedly move the topmost container until the stack is empty
  - move topmost: take followed by put action

Adapted from slides © 2005 G. Wickler & A. Tate
University of Edinburgh
http://bit.ly/Diir0

DWR – Crane Domain Example [2]: Initial & Goal States



(a) initial state

(b) goal

Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

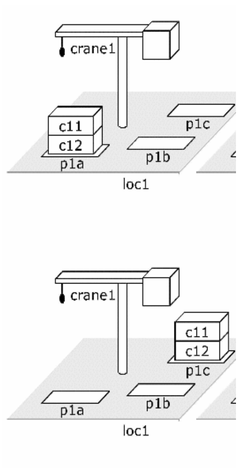
DWR Crane Domain Example [3]: Total Order Tasks (SHOP-STD)

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

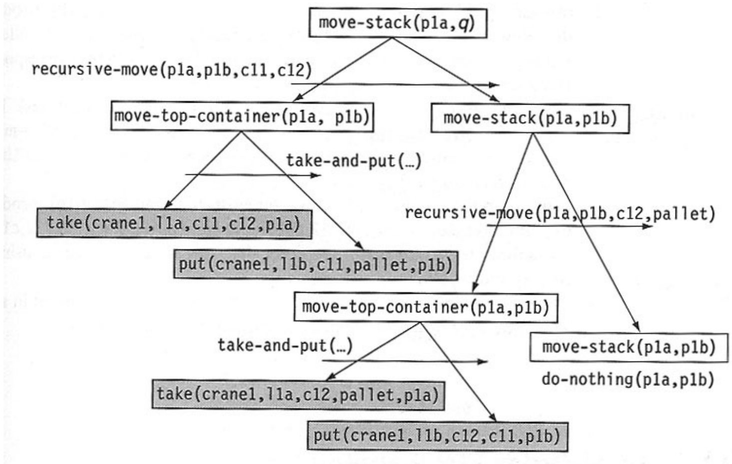
do-nothing(p, q)
  task: move-stack(p, q)
  precond: top($pallet, p$) *; true if p is empty*
  subtasks: ⟨⟩ *; no subtasks, because we are done*

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)⟩



Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

DWR – Crane Domain Example [4]: Example TFD



Adapted from slide © 2007 H. Hoang & H. Muñoz-Avila
Lehigh University
http://bit.ly/2nUghl

References

- Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*, chapter 11. Elsevier/Morgan Kaufmann, 2004.
- E. Sacerdoti. The nonlinear nature of plans. In: *Proc. IJCAI*, pages 206-214, 1975.
- A. Tate. Generating project networks. In: *Proc. IJCAI*, pages 888-893, 1977.

Adapted from slide © 2005 G. Wickler & A. Tate
University of Edinburgh
http://bit.ly/Diir0

Practical Planning Solutions [1]: Sensorless Planning

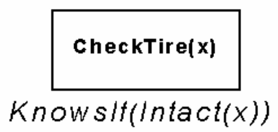
- **Problem: Bounded Indeterminacy**
  - **Uncertainty in answering intelligent agent's questions (see: Lectures 0 & 1)**
    - **"What world is like now"**
    - **"What it will be like if I do action A"**
  - **Scenario for boundedly rational decision-making**
- **Idea: Coerce State of World**
  - **Complete plan in all possible situations**
  - **Example: move forward to walk through door OR push it open**
- **Not Always Possible!**

Practical Planning Solutions [2]: Conditional Planning

$[\ldots, \mathbf{If}(p, [then\,plan], [else\,plan]), \ldots]$

Execution: check p against current KB, execute "then" or "else"

Conditional planning: just like POP except
  if an open condition can be established by <u>observation</u> action
    add the action to the plan
    complete plan for each possible observation outcome
    insert conditional step with these subplans



Terminology

- Propositional Planning Domains
  - Boolean variables (see cargo plane example)
  - Goal: to find truth assignment that satisfies goal, given initial conditions
  - Admit solution using Boolean satisfiability (SAT) solvers
- Hierarchical Abstraction Planning
  - Subplan: plan that is treated as operator of larger plan
    - Initial conditions: preconditions of operator
    - Goals: effects of operator
  - Decomposable plan: steps consist of subplans
  - Plan refinement: decomposition of plans (down to lowest level of operators)
- Hierarchical Task Network (HTN)
- Bounded Indeterminacy: Kind of Uncertainty about Domain
  - "How world is like"
  - "How it will be if I do A"
- Robust Planning: Plan Generation under Uncertainty

Summary Points

- Last Class: Graph Planning and HTN Preview
  - Graph planning (11.4)
  - Planning with propositional logic (11.5)
  - Analysis of planning approaches (11.6)
  - Summary (11.7)
- Today: Real-World Planning
  - Time (12.1)
  - HTN Planning (12.2)
  - Nondeterminism (12.3)
  - Conditional planning (12.4)
- Coming Week: Robust Planning Concluded; Uncertain Reasoning