Planning, Execution & Learning: Hierarchical Task Net Planning

Reid Simmons

Hierarchical Task Net (HTN) Planning

- **Basic Ideas**:
  - Complex plans often have identifiable structure
  - That structure can often be captured in the form of hierarchies of abstract subplans
  - Subplans are often (nearly) independent of one another
  - ***Problem Reduction*** search, rather than state space search

"To get to conference in ?x, get to the airport, take a plane to ?x, then get to the conference hotel"

"To get to the airport, either drive or take a cab"

"If you have enough money for the fare:
To take a cab to ?y, either call ahead, or flag a cab down, then enter the cab, say "I want to go to ?y", wait until at ?y, pay the fare, then exit the cab"

Abstraction

- Two Notions of Plan Abstraction
  1. Abstract operator ***decomposes*** into partial subplan
  2. Abstract operator is ***refined*** by adding details (less critical preconditions and effects)

- We will focus mainly on #1
  – #2 can be considered a special case of #1

Abstract Plans

- ***Primitive*** Operators
  - Standard STRIPS-style operators
  - “Executable”
- ***Compound*** Operators
  - Preconditions and Effects
  - ***Methods*** for decomposing operator into more detailed subplans
    - Details internal structure
    - Similar to *macros* or *subroutines*

- An ***Abstract Plan*** contains compound operators
- A ***Fully Instantiated Plan*** has only primitive operators

Methods

- ***Methods*** (also called ***schema***) are used for decomposing operators
  - Name and task variables
  - Preconditions
  - Effects

Part that describes what operator does (same as with primitive operators)

  - **Applicability conditions**
  - **Expansion**
  - **Temporal orderings**
  - **Causal links**

Part that details how the operator is to be decomposed

  - ***Time windows***
  - ***Resource utilization***

Part that describes quantitative aspects of subplan

Simple Hierarchical Order Planner (SHOP) *(Nau, et.al. 1999)*

- Basic HTN Algorithm:
  - Start with initial high-level ***task*** (*not* goals)
  - Create ***task net*** by repeatedly expanding subplans until plan is fully instantiated
  - Select ***methods*** whose applicability conditions hold

- SHOP Algorithm:
  - Forward search, linear planner
    - Plans in same order as execution
    - Essentially depth-first search
  - Primitive operators have no preconditions
  - No concurrent actions
  - Highly expressive operator representation (numeric calculations)
  - Efficient (but rather inflexible) planning algorithm

SHOP Domain Example

```
(:operator  (!putdown ?block)
   ((holding ?block))                 Delete list
   ((ontable ?block) (handempty)))        Add list

(:method  (make-clear ?y)
   ((clear ?y))          Applicability condition
   nil)                  Task list

(:method  (make-clear ?y)
   ((on ?x ?y))          Applicability condition
   ((make-clear ?x)
     (!unstack ?x ?y) (!putdown ?x))      Task list
```

Extensions to Simple HTN Planning

1. Threat detection
   - Deal with interacting goals
   - Find ways of reusing operators
2. Methods can include preconditions and effects
   - Find threats earlier in the planning process
3. Methods can indicate resource usage
   - Do some types of scheduling
4. Operators/Methods can include open conditions
   - Need to use action-based (refinement) planning
   - Enables planner to find “novel” solutions

Example: Home Construction (O-Plan)

Method Build (?house)

Precondition:	(and (own land) (have money))
Effects:	(built ?house)
Applicability:	(single-family-home ?house)
Expansion:	S1: Build-Foundation(?house)
	S2: Build-Frame(?house)
	S3: Build-Roof(?house)
	S4: Build-Walls(?house)
	S5: Build-Interior(?house)
	S6: Decorate(?house)
Orderings:	S1<S2, S2<S3, S2<S4, S3<S5, S5<S6
Links:	S1 causes (foundations laid) for S2
	S2 causes (frame erected) for S3 and S4
	S3 causes (roof built) for S5
	S4 causes (walls built) for S5
	S5 causes (interior done) for S6
TimeWindow:	start between 11:30 and 14:30 at S3
Resources:	bricklayers = between 1 and 2 men at S4

Where is the Power?

- Methods encode domain knowledge
- Methods encode problem solving knowledge
- Abstractions encapsulate patterns of interaction

Caveats

- – HTN planning, in worst case, is still NP-complete
- – May not terminate (recursive method expansions – may be hard to detect infinite loops)
- – May have to wait until completely expanded before finding plan is illegal

Expressivity of HTN Planning

- Theoretically, HTN Planning is More Expressive than Action-Based Planning!!
  - HTN planning can generate a larger class of plans
  - Proof (by Erol) involves reduction to classes of grammars
  - Action-based planning is analogous to ***right-linear*** (***regular***) grammars
  - HTN planning is analogous to ***context-free*** grammars

Example:

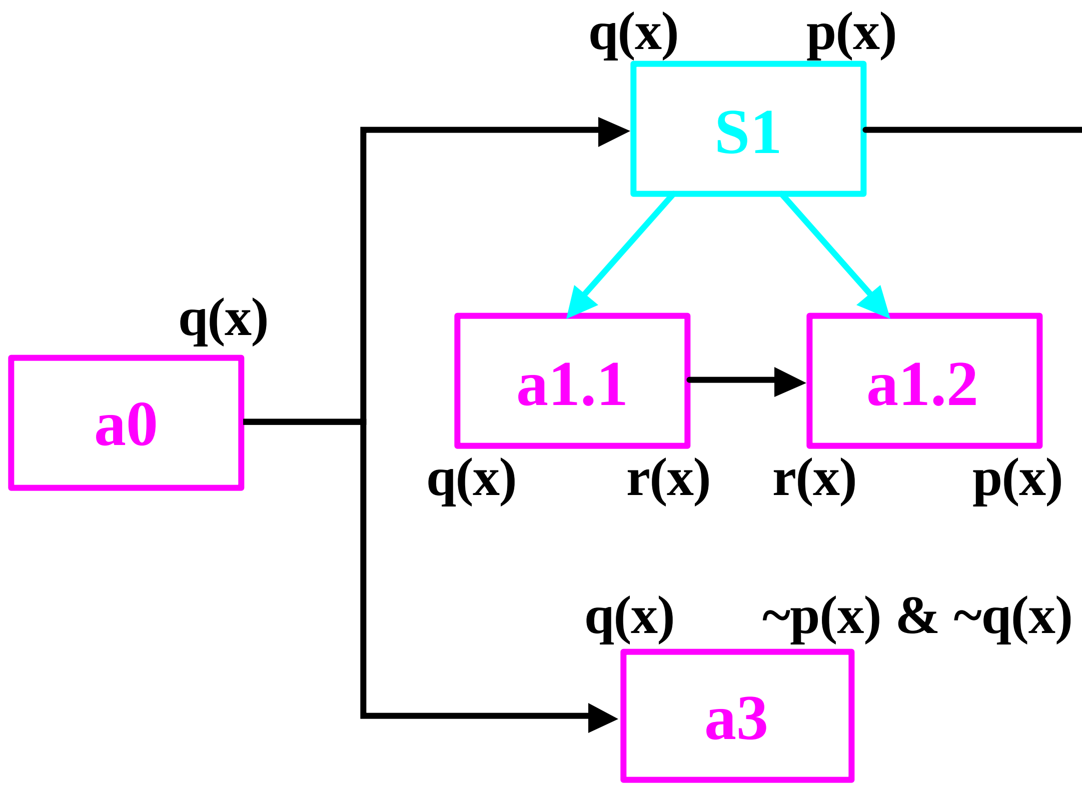
Want to create round-trip transportation plans such that the leg of the trip from X to Y always uses the same carrier as the leg of the trip from Y to X.

Dealing with Subplan Interactions

- Types of Interactions
  - Deleted condition (threat)
  - Resource (e.g., “existing object” bindings)
  - Redundant steps

- HTN Planners Often Use ***Critics*** to Detect Certain Types of Illegal Plans and/or Synergies
  - Time window bounds
  - Resource bounds
  - Interaction of effects between compound tasks
  - Operators that can be ***merged***
    - In contrast, POP planners try to ***share*** operators
    - Merging is more efficient, but may not be complete

Danger of Over-Commitment

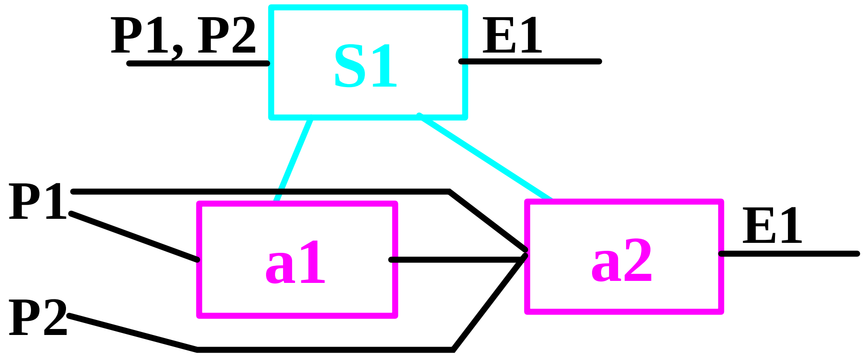


Properties of Abstract Plans

- Abstract Plans can be ***Consistent*** and ***Complete***
  - **Consistent**: No inconsistent orderings or bindings
  - **Complete**: Every precondition is achieved
- ***Downward*** Solution Property
  - If an abstract plan is consistent and complete, then there is a full instantiation of it that is also consistent and complete
  - Implies that one can focus exclusively on that plan
- ***Upward*** Solution Property
  - If an abstract plan is inconsistent, then no full instantiation of that plan is consistent
  - Implies that one can prune away inconsistent abstract plans

Properties of Abstract Plans

- If Downward and/or Upward Solution Properties Hold, HTN Planning is Very Efficient
  - ***$O(bs^d)$*** *vs.* ***$O(b^n)$***
    - ***b***: branching factor; ***s***: average steps in method; ***d***: depth of expansion; ***n***: length of plan
- Upward solution property holds if method has ***unique main subaction***
  - There is one step of the decomposition to which all preconditions and effects are attached



Metric Resources

- ***Consumable*** (no replenishment)
  - Example: "*available oil reserves*"
  - Representation: **[r_l, r_u]** ranges; Constraint ***propagation***
- ***Renewable*** (complicated by substitution effects)
  - Example: "*money*" "*fuel*"
  - Representation: Algebraic; Constraint ***satisfaction***
- ***Sharable – Single Unit***
  - Example: "*a spacecraft's camera*"
  - Representation: **Mutex**; Simple ***add/delete***
- ***Sharable – Multiple Units***
  - Example: "*drill presses in a factory*"
  - Representation: ***Need explicit scheduling techniques***

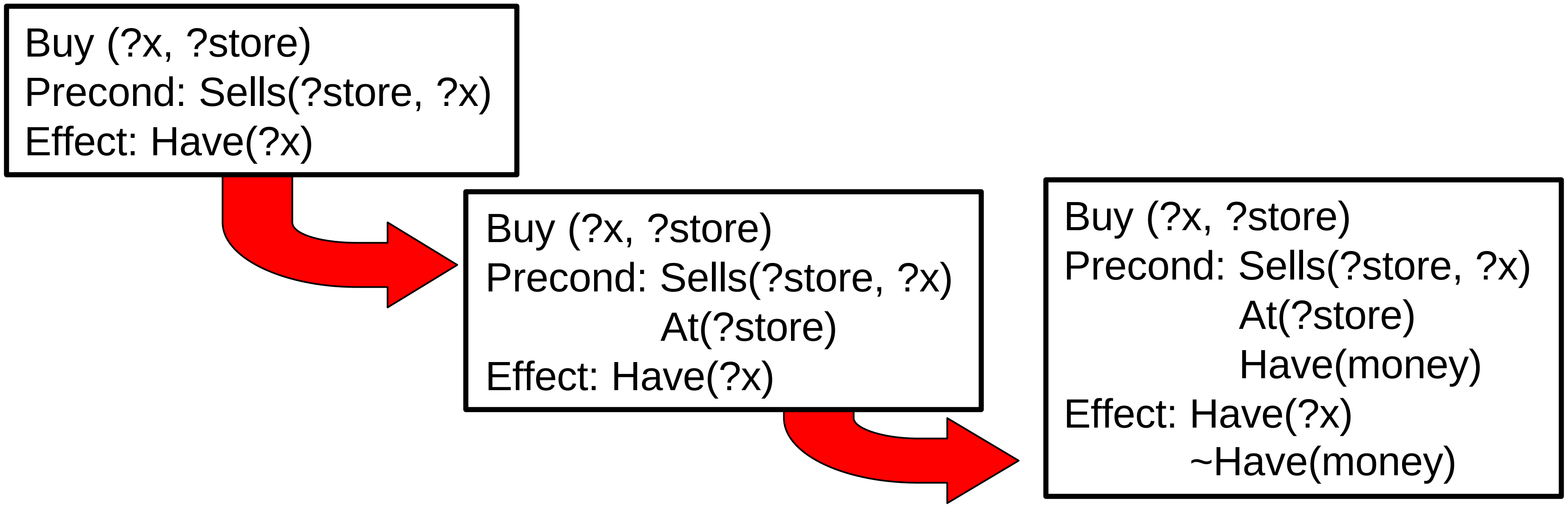
Handling Metric Time

- Time Can be Treated as a Metric Resource, but it is Special
  - Non-exclusive resource (multiple actions can occur during same time interval)
  - Non-renewable resource (cannot create more time – ***sigh***)
  - Time resources must be consistent with temporal orderings

- Representations
  - Precise time: map to **reals** or **integers**
  - Time windows: **[min, max]**
  - Algebraic: **($t1 + t2 < duration$)**

Abstraction Planning

- ***Not*** the Same as HTN
  - Although hierarchical, as is HTN, representation is ***action operators***, same as with action-based planning
  - Search space is same as with action-based planning
  - Hierarchy within ***single*** operator – achieved by removing preconditions and effects to get “simpler” operator



```
Buy (?x, ?store)
Precond: Sells(?store, ?x)
Effect: Have(?x)
```

```
Buy (?x, ?store)
Precond: Sells(?store, ?x)
              At(?store)
Effect: Have(?x)
```

```
Buy (?x, ?store)
Precond: Sells(?store, ?x)
              At(?store)
              Have(money)
Effect: Have(?x)
          ~Have(money)
```

Abstraction Planning

- **Basic Idea**: Save planning time by working first on parts of plan that will not be affected by subsequent planning

- Abstraction Hierarchy Should Maintain the ***Ordered Monotonicity*** Property (Knoblock)
  – A literal that “interacts” with another literal is at the same, or lower, abstraction level
  – Literals that interact with one another are at the same level
  – Abstraction level consists only of preconditions from that level, or higher