Planning:
Graph Planning and Hierarchical Abstraction

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:

Section 11.4 – 11.7, p. 395 – 408, Russell & Norvig 2nd edition

Lecture Outline

- **Reading for Next Class: Sections 11.4 – 11.7 (p. 395 – 408), R&N 2e**
- **Last Class: Sections 11.1 – 11.2 (p. 375 – 386), R&N 2e**
  - **Planning problem: initial conditions, actions (preconditions/effects), goal**
  - **STRIPS operators: represent actions with preconditions, ADD/DELETE list**
  - **ADL operators: allow negated preconditions, inequality**
  - **Examples: socks and shoes, blocks world, changing spare tire**
- **Today: Partial-Order Planning, Section 11.3 (p. 387 – 394), R&N 2e**
  - **Plan linearization**
  - **Extended POP example: changing spare tire**
  - **Graph planning**
  - **Hierarchical abstraction planning (ABSTRIPS)**
- **Coming Week: Robust Planning Concluded; Uncertain Reasoning**

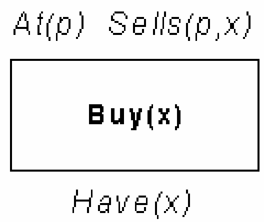
STRIPS Operators: Review

Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p,x)$
EFFECT: $Have(x)$

[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm
  Precondition: conjunction of positive literals
  Effect: conjunction of literals



$Action(Fly(p, from, to),$
  PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  EFFECT: $\neg At(p, from) \wedge At(p, to))$

Adapted from materials © 2003 – 2004 S. Russell & P. Norvig. Reused with permission.

STRIPS *vs.* ADL Representation [1]: Review

- **What STRIPS *Can* Represent**
  - **States**
  - **Goals**
  - **Actions (using action schema)**
    - ⇨ **<u>Preconditions</u>: must be true before action can be applied**
    - ⇨ **<u>Effects</u>: asserted afterwards**

$$\begin{array}{l} Action(Fly(p, from, to), \\ \quad \text{PRECOND:} At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\ \quad \text{EFFECT:} \neg At(p, from) \land At(p, to)) \end{array}$$

- **Real STRIPS: <u>ADD</u>, <u>DELETE</u> Lists for Operators**
- **<u>STRIPS Assumption</u>**
  - **Representational frame problem solution**
  - **Default is that conditions remain unchanged unless mentioned in effect**
- **What STRIPS *Cannot* Represent**
  - **Negated preconditions**
  - **Inequality constraints**
- **Richer Planning Language: <u>A</u>ction <u>D</u>escription <u>L</u>anguage (ADL)**

$$\begin{array}{l} Action(Fly(p : Plane, from : Airport, to : Airport), \\ \quad \text{PRECOND:} At(p, from) \land (from \neq to) \\ \quad \text{EFFECT:} \neg At(p, from) \land At(p, to)) \,. \end{array}$$

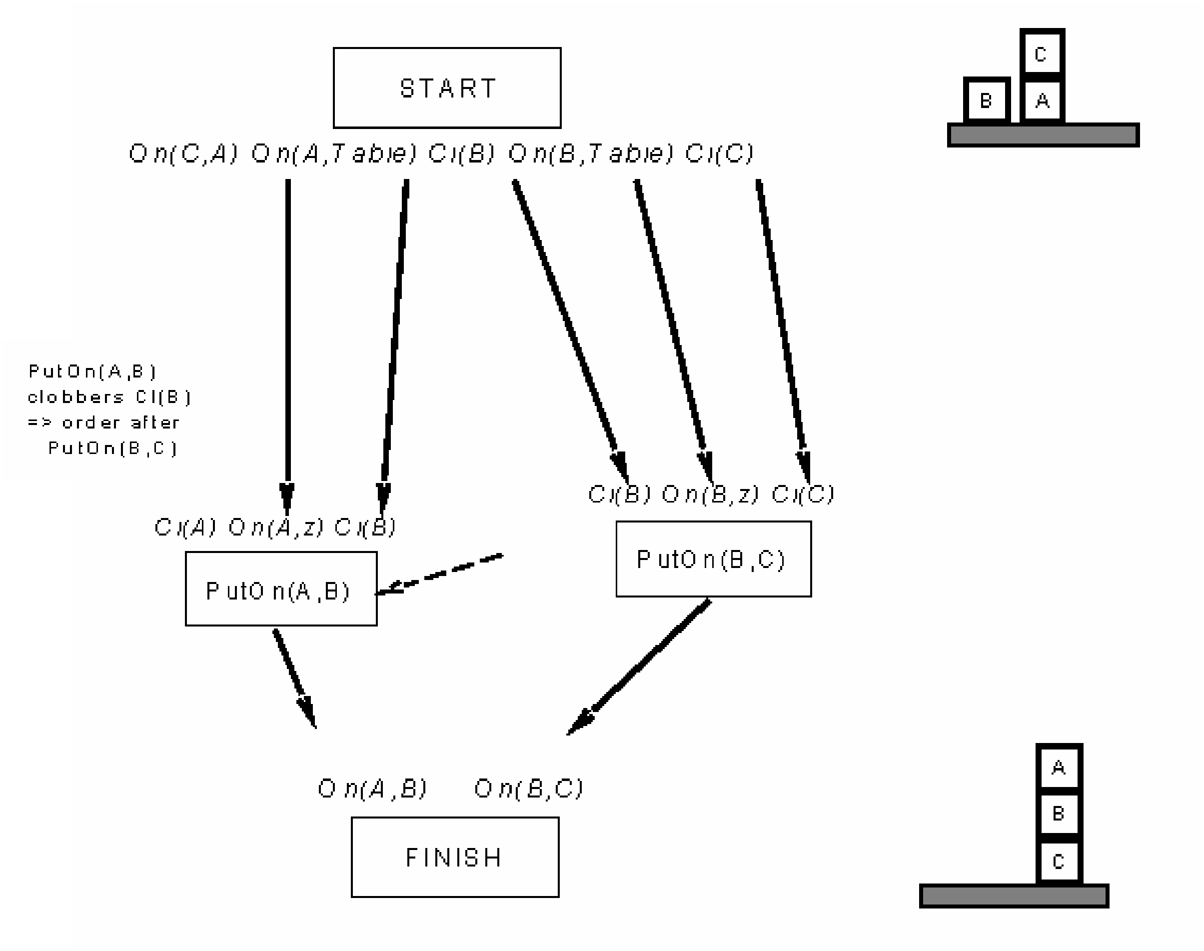
STRIPS *vs.* ADL Representation [2]: Review

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x\, At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

Figure 11.1
p. 379 R&N 2e

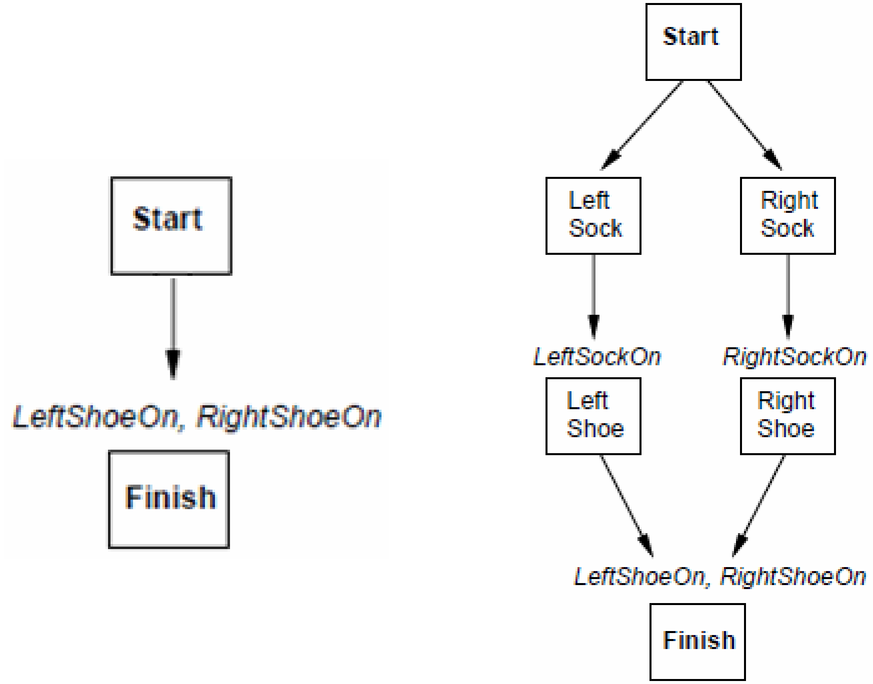
© 2003 S. Russell & P. Norvig. Reused with permission.

Sussman Anomaly: Review



Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.

Partial Order Planning – Definitions: Review



A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it

Adapted from materials © 2003 – 2004 S. Russell & P. Norvig. Reused with permission.

POP, Linearization, & Total Orderings: Review

- **Socks and Shoes Example: POP Constraints**

$$
\begin{aligned}
&\text{Actions:}\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}\\
&\text{Orderings:}\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}\\
&\text{Links:}\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,\\
&\qquad RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}\\
&\text{Open Preconditions:}\{\ \}\ .
\end{aligned}
$$

- **Plan Linearization**
  - **Total ordering**
  - **Enumerating interleavings: combinatorial explosion**
- **Theorem: Partial Order (PO) Plans**
  - **Every linearization of PO plan is total ordering (TO)**
  - **TO is guaranteed to satisfy goal condition(s) given initial condition(s)**

Adapted from materials © 2003 – 2004 S. Russell & P. Norvig. Reused with permission.

POP Algorithm – Top-Level Functions: Review

```
function POP(initial, goal, operators) returns plan

   plan ← MAKE-MINIMAL-PLAN(initial, goal)
   loop do
       if SOLUTION?(plan) then return plan
       S_need, c ← SELECT-SUBGOAL(plan)
       CHOOSE-OPERATOR(plan, operators, S_need, c)
       RESOLVE-THREATS(plan)
   end
```

```
function SELECT-SUBGOAL(plan) returns S_need, c

   pick a plan step S_need from STEPS(plan)
       with a precondition c that has not been achieved
   return S_need, c
```

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

Spare Tire Planning Example [1]: Review

On(x) ~Flat(x)

FINISH

START

~Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)

On(x)

Remove(x)

Off(x) ClearHub

Off(x) ClearHub

Puton(x)

On(x) ~ClearHub

Intact(x) Flat(x)

Inflate(x)

~Flat(x)

© 2004 S. Russell & P. Norvig. Reused with permission.

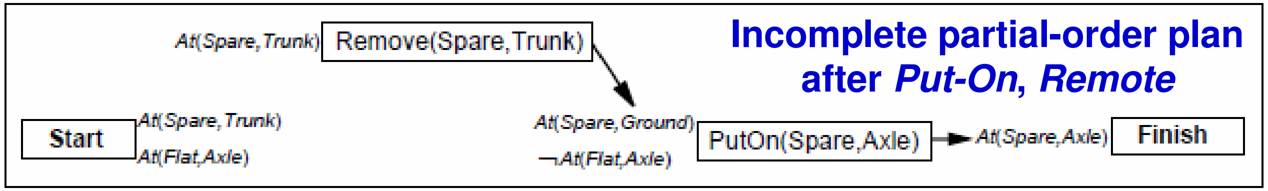
Spare Tire Planning Example [2]: Review

$Init(At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
$\quad$ PRECOND: $At(Spare, Trunk)$
$\quad$ EFFECT: $\neg At(Spare, Trunk) \land At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
$\quad$ PRECOND: $At(Flat, Axle)$
$\quad$ EFFECT: $\neg At(Flat, Axle) \land At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
$\quad$ PRECOND: $At(Spare, Ground) \land \neg At(Flat, Axle)$
$\quad$ EFFECT: $\neg At(Spare, Ground) \land At(Spare, Axle))$
$Action(LeaveOvernight,$
$\quad$ PRECOND:
$\quad$ EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
$\qquad\qquad \land \neg At(Flat, Ground) \land \neg At(Flat, Axle))$

Figure 11.3 &11.7
p. 381 (& 391)
R&N 2e

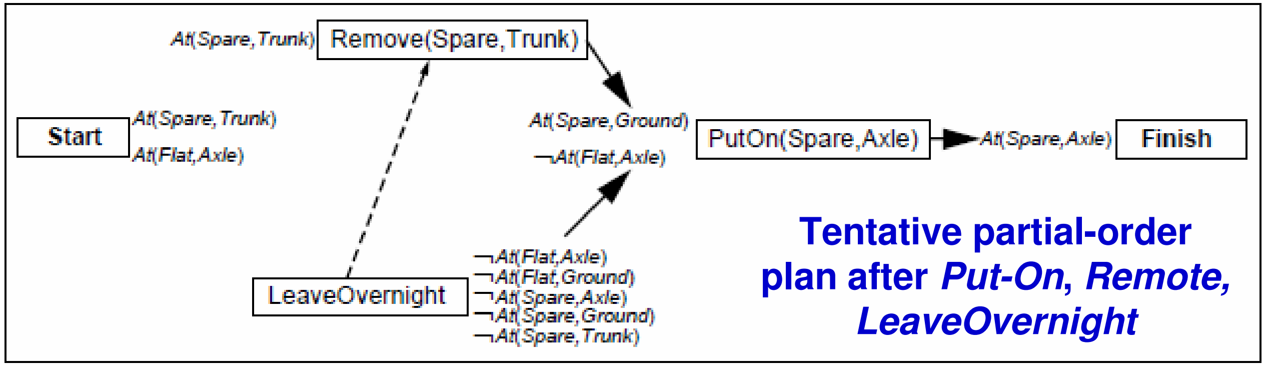
© 2003 S. Russell & P. Norvig. Reused with permission.

Spare Tire Planning Example [3]: POP Trace



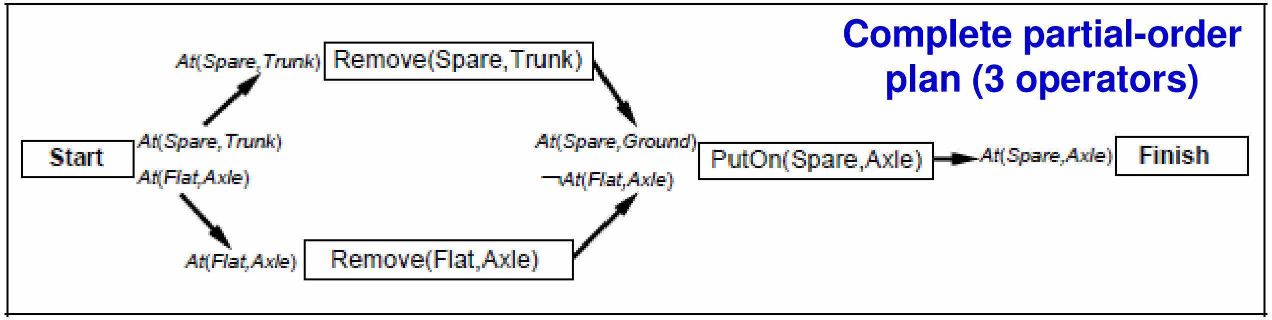
Incomplete partial-order plan after *Put-On*, *Remote*

Figure 11.8
p. 392 R&N 2e



Tentative partial-order plan after *Put-On*, *Remote*, *LeaveOvernight*

Figure 11.9
p. 392 R&N 2e



Complete partial-order plan (3 operators)

Figure 11.10
p. 393 R&N 2e

© 2003 S. Russell & P. Norvig. Reused with permission.

Instantiation of STRIPS/ADL Operators

$$
\begin{aligned}
&Action(Move(b,x,y),\\
&\quad \text{PRECOND:}On(b,x) \land Clear(b) \land Clear(y),\\
&\quad \text{EFFECT:}On(b,y) \land Clear(x) \land \neg On(b,x) \land \neg Clear(y))
\end{aligned}
$$

$$
\begin{aligned}
&Action(Move(A,x,B),\\
&\quad \text{PRECOND:}On(A,x) \land Clear(A) \land Clear(B),\\
&\quad \text{EFFECT:}On(A,B) \land Clear(x) \land \neg On(A,x) \land \neg Clear(B))
\end{aligned}
$$

$$Move(A,x,B) \xrightarrow{On(A,B)} Finish$$

© 2003 S. Russell & P. Norvig. Reused with permission.

Heuristics for Classical Planning

- **Problem: Combinatorial Explosion due to High Branch Factor**
  - **Branch factor (main problem in planning): possible operators**
  - **Fan-out: many side effects**
  - **Fan-in: many preconditions to work on at once**
- **Goal: Speed Up Planning**
- **Heuristic Design Principles**
  - **Favor general ones (domain-independent)**
  - **Treat as goals as countable or continuous instead of boolean (true/false)**
  - **Use commonsense reasoning (need commonsense knowledge)**
    - ⇨ **Counting, weighting partially-achieved goals**
    - ⇨ **Way to compute preferences (utility estimates)**
- **Domain-Independent *h*: Number of Unsatisfied Conjuncts**
  - **e.g., Have(A) $\wedge$ Have(B) $\wedge$ Have(C) $\wedge$ Have(D)**
  - **Have(A) $\wedge$ Have(C): $h = 2$**
- **Domain-Dependent *h*: May Be Based on Problem Structure**

© 2003 S. Russell & P. Norvig. Reused with permission.

Graph Planning: **Graphplan** Algorithm

- **Previous Heuristics for STRIPS/ADL**
  - Domain-independent heuristics: counting parts (conjuncts) of goal satisfied
  - Domain-dependent heuristics: based on (many) domain properties
    - ⇨ problem decomposability (intermediate goals)
    - ⇨ reusability of solution components
    - ⇨ preferences
- **Limitation: Heuristics May Not Be Accurate**
- **Objective: Better Heuristics**
  - Need: structure that clarifies problem
  - Significance: faster convergence, more manageable branch factor
- **Approach: Use Graphical Language of Constraints, Actions**
- **Notation**
  - Operators (real actions): large rectangles
  - <u>Persistence actions</u> (for each literal): small squares, denote non-change
  - Gray links: mutual exclusion (mutex)

© 2003 S. Russell & P. Norvig. Reused with permission.

Graph Planning: Cake Problem

Init(*Have*(*Cake*))
Goal(*Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Eat*(*Cake*)
  PRECOND: *Have*(*Cake*)
  EFFECT: ¬ *Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Bake*(*Cake*)
  PRECOND: ¬ *Have*(*Cake*)
  EFFECT: *Have*(*Cake*)

Figure 11.11
p. 396 R&N 2[e]

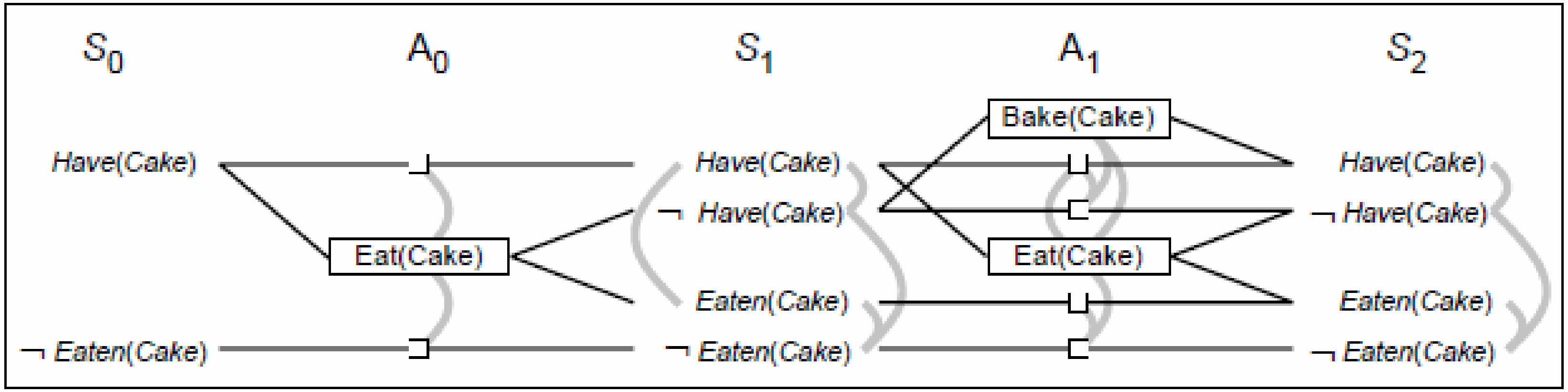
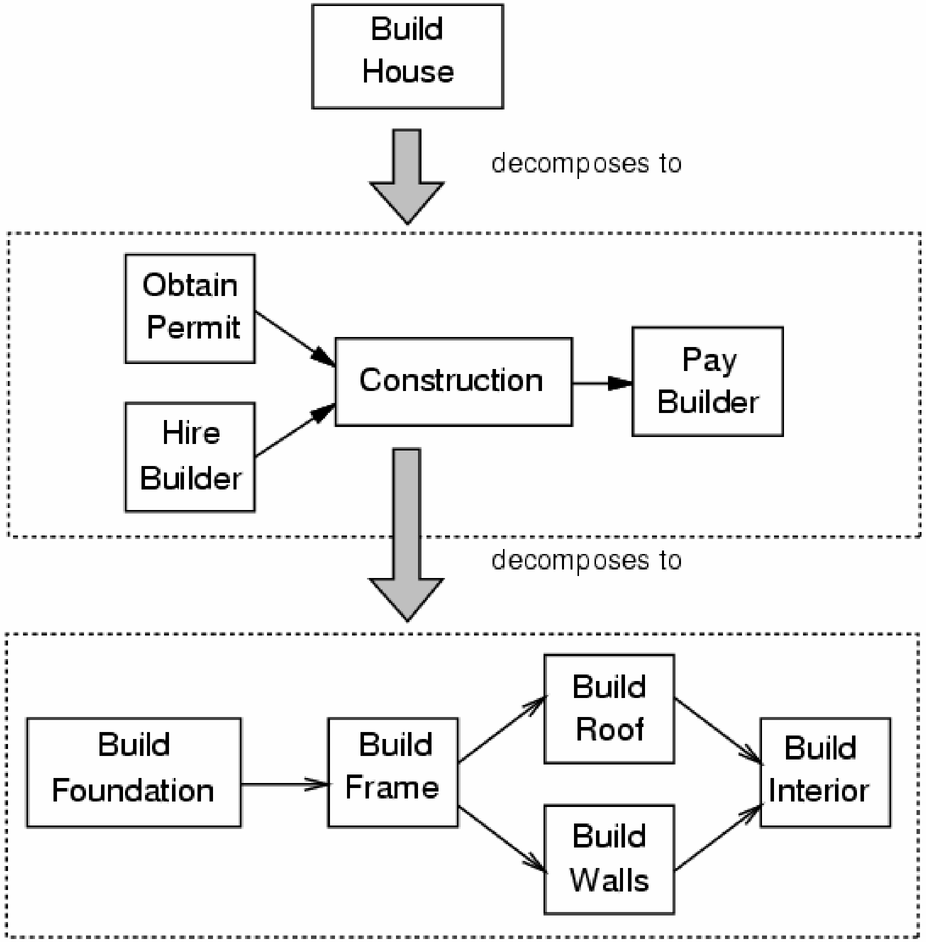


Figure 11.12
p. 396 R&N 2[e]

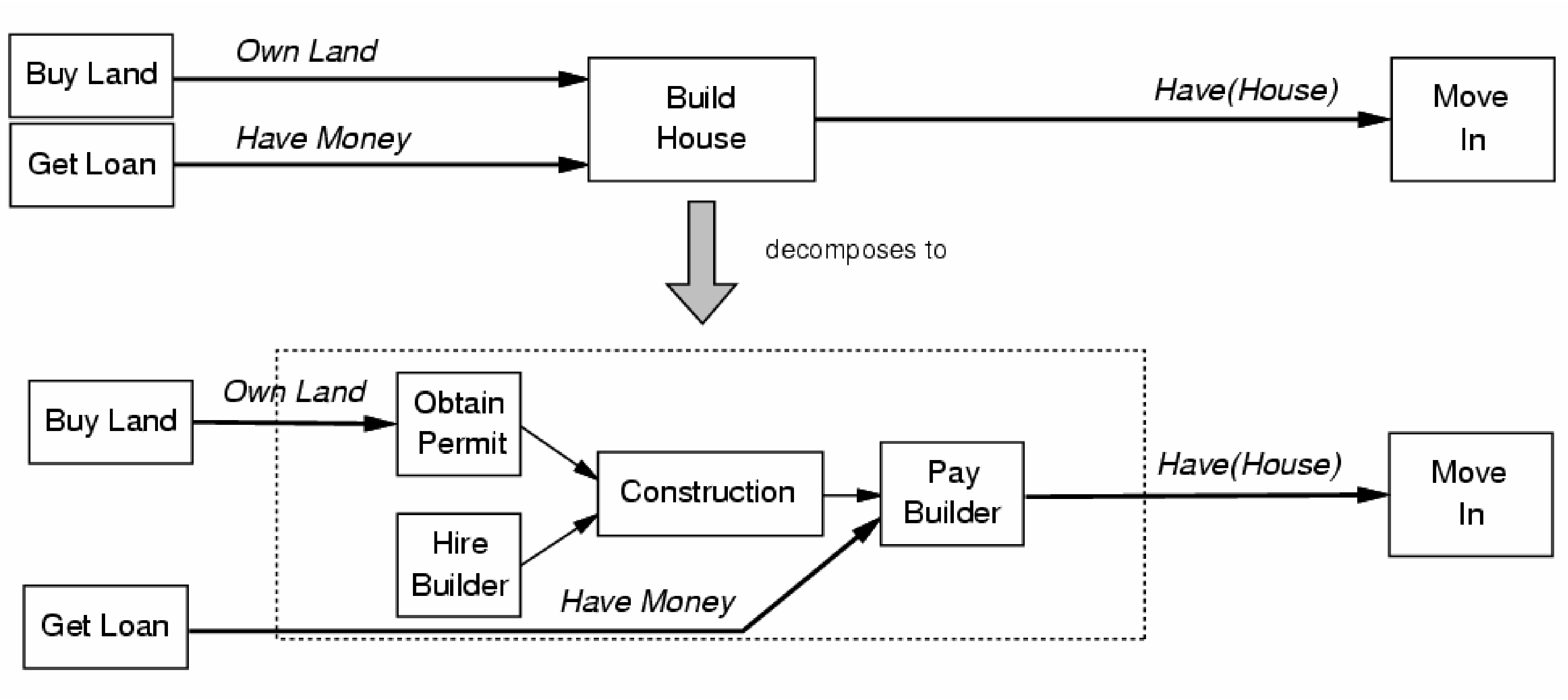
© 2003 S. Russell & P. Norvig. Reused with permission.

Hierarchical Abstraction: House-Building Example



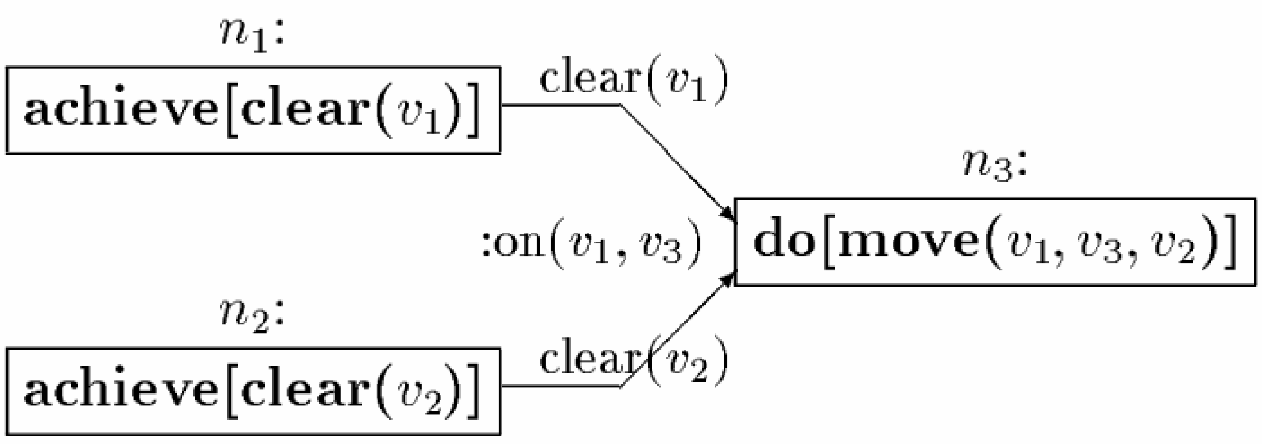
© 2003 S. Russell & P. Norvig. Redrawn by José Luis Ambite, ISI http://bit.ly/3ldmiM

Hierarchical Abstraction: House-Building Example



© 2003 S. Russell & P. Norvig. Redrawn by José Luis Ambite, ISI http://bit.ly/3ldmiM

Hierarchical Task Network (HTN) Planning



$$[(n_1 : achieve[clear(v_1)])(n_2 : achieve[clear(v_2)])(n_3 : do[move(v_1, v_3, v_2)])$$
$$(n_1 \prec n_3) \wedge (n_2 \prec n_3) \wedge (n_1, clear(v_1), n_3) \wedge (n_2, clear(v_2), n_3) \wedge (on(v_1, v_3), n_3)$$
$$\wedge \neg(v_1 = v_2) \wedge \neg(v_1 = v_3) \wedge \neg(v_2 = v_3)]$$

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

How Things Go Wrong in Planning

Incomplete information

Unknown preconditions, e.g., $Intact(Spare)$?

Disjunctive effects, e.g., $Inflate(x)$ causes

$Inflated(x) \lor SlowHiss(x) \lor Burst(x) \lor BrokenPump \lor \ldots$

Incorrect information

Current state incorrect, e.g., spare NOT intact

Missing/incorrect postconditions in operators

Qualification problem:

can never finish listing all the required preconditions and possible conditional outcomes of actions

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

Practical Planning Solutions [1]: Conditional Planning & Replanning

Conditional planning

Plan to obtain information (**observation actions**)

Subplan for each contingency, e.g.,

$[Check(Tire1), \textbf{If}(Intact(Tire1), [Inflate(Tire1)], [CallAAA])]$

Expensive because it plans for many unlikely cases

Monitoring/Replanning

Assume normal states, outcomes

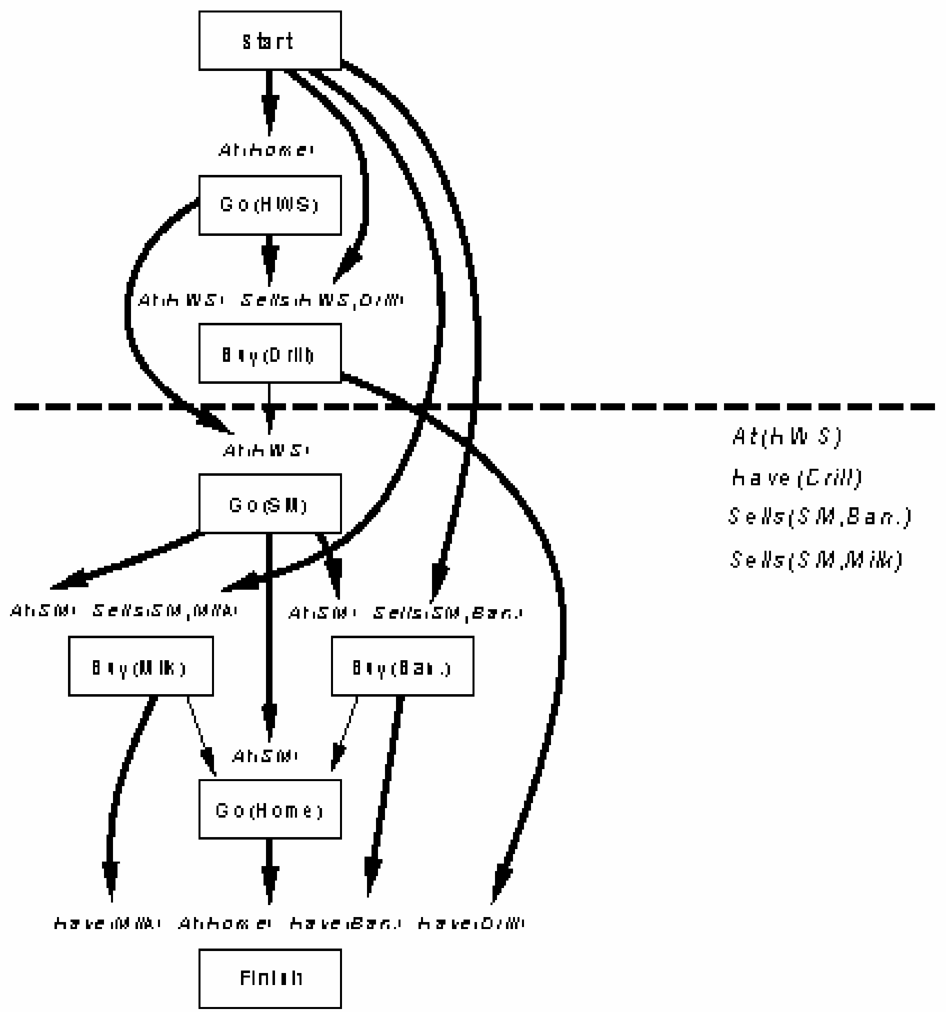
Check progress *during execution*, replan if necessary

Unanticipated outcomes may lead to failure (e.g., no AAA card)

In general, some monitoring is unavoidable

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

Practical Planning Solutions [2]: Continual Planning – Preview



© 2004 S. Russell & P. Norvig. Reused with permission.

Terminology

- **Partial-Order Planning**
  - Represent multiple possible interleavings
  - Keep track of which ones are achievable
  - Complete plans
    - Every precondition achieved
    - No clobberings by possibly intervening steps
- **Sussman Anomaly**
  - Contains threat that needs to be resolved to get to goal
  - Illustrates need for partial-order planning, promotion / demotion
- **Hierarchical Abstraction Planning: Refinement of Plans into Subplans**
- **Robust Planning**
  - Sensorless: use coercion and reaction
  - Conditional *aka* contingency: IF statement
  - Monitoring and replanning: resume temporarily failed plans
  - Continual *aka* lifelong: multi-episode, longeval or "immortal" agents

Summary Points

- **Last Class: Classical Planning – STRIPS and ADL**
  - **Planning defined: initial conditions, actions (preconditions/effects), goal**
  - **STRIPS operators: conjunction of positive preconditions**
  - **ADL operators: allow negated preconditions, unequality**
- **Today: Graph Planning, Hierarchical Abstraction**
  - **GRAPHPLAN algorithm illustrated**
  - **Hierarchical abstraction planning (ABSTRIPS)**
- **Preview: Robust Planning**
  - **Planning with plan step failures**
  - **Types**
    - ⇨ **<u>Sensorless</u>: use coercion and <u>reaction</u>**
    - ⇨ **<u>Conditional</u> *aka* <u>contingency</u>: IF statement**
    - ⇨ **<u>Monitoring and replanning</u>: resume temporarily failed plans**
    - ⇨ **<u>Continual</u> *aka* <u>lifelong</u>: multi-episode, longeval or "immortal" agents**
- **Coming Week: More Robust Planning Continued**