CS 2710 / ISSP 2610

R&N 10.3
R&N 11.1-11.3

1

Planning

- What is planning?
- Approaches to planning
  - Situation calculus
  - STRIPS
  - Partial-order planning

2

Planning problem

- Find a **sequence of actions** that achieves a given **goal** when executed from a given **initial world state**. That is, given
  - a set of operator descriptions (defining the possible primitive actions by the agent),
  - an initial state description, and
  - a goal state description or predicate,

  compute a plan, which is
  - a sequence of operator instances, such that executing them in the initial state will change the world to a state satisfying the goal-state description.
- Goals are usually specified as a conjunction of goals to be achieved

3

Planning vs. problem solving

- Planning and problem solving methods can often solve the same sorts of problems
- Planning is more powerful because of the representations and methods used
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
- Search often proceeds through *plan space* rather than *state space* (though there are also state-space planners)
- Subgoals can be planned independently, reducing the complexity of the planning problem

4

Typical assumptions

- Atomic time: Each action is indivisible
- No concurrent actions are allowed (though actions do not need to be ordered with respect to each other in the plan)
- Deterministic actions: The result of actions are completely determined—there is no uncertainty in their effects
- Agent is the sole cause of change in the world
- Agent is omniscient: Has complete knowledge of the state of the world
- Closed World Assumption: everything known to be true in the world is included in the state description. Anything not listed is false.

5

Blocks world

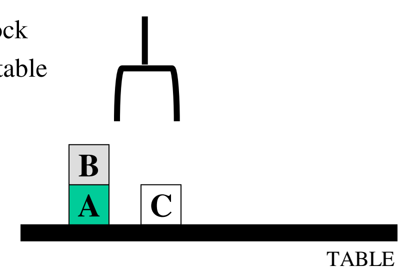
The **blocks world** is a micro-world that consists of a table, a set of blocks and a robot hand.

Some domain constraints:

- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation:

ontable(a)
ontable(c)
on(b,a)
handempty
clear(b)
clear(c)



6

Major approaches

- **Situation calculus**
- **STRIPS**
- **Partial order planning**
- Planning with constraints (SATplan, Graphplan)
- Reactive planning (not covered)

7

Situation calculus planning

- Intuition: Represent the planning problem using first-order logic
  - Situation calculus lets us reason about changes in the world
  - Use theorem proving to "prove" that a particular sequence of actions, when applied to the situation characterizing the world state, will lead to a desired result

8

Motivation

- Recall problems with propositional logic. So FOL?
- The robot is in the kitchen.
  - in(robot,kitchen)
- It walks into the living room.
  - in(robot,livingRoom)
- Ooops…
- in(robot,kitchen,2:02pm)
- in(robot,livingRoom,2:17pm)
- But what if you are not sure when it was?
- We can do something simpler than rely on time stamps…

9

Representation of Time

- Lots of other approaches besides situations, e.g.
  - Temporal Logic, Dynamic Logic
  - Maintaining Knowledge about Temporal Intervals

10

Situation Calculus

- Logic for reasoning about changes in the state of the world
- The world is described by
  - Sequences of situations of the current state
  - Changes from one situation to another are caused by actions
- The situation calculus allows us to
  - Describe the initial state and a goal state
  - Build the KB that describes the effect of actions (operators)
  - Prove that the KB and the initial state lead to a goal state
  - Extracts a plan as side-effect of the proof

11

Situation Calculus Ontology

- Actions: terms, such as “forward” and “turn(right))”
- Situations: terms; initial situation s0 and all situations that are generated by applying an action to a situation. result(a,s) names the situation resulting when action a is done in situation s.

12

Situation Calculus Ontology continued

- Fluents: functions and predicates that vary from one situation to the next. By convention, the situation is the last argument of the fluent. ~holding(robot,gold,s0)
- Atemporal or eternal predicates and functions do not change from situation to situation. gold(g1). lastName(wumpus,smith). adjacent(livingRoom,kitchen).

Modified Wumpus World

- Won't worry about agent's orientation
- Fluent predicates: at(O,X,S) and holding(O,S)
- Initial situation: at(agent,[1,1],s0) ^ at(g1,[1,2],s0)
- But we want to exclude possibilities from the initial situation too…

Initial KB

- All O,X at(O,X,s0) ←→ [O=agent ^ X = [1,1]) v (O=g1 ^ X = [1,2])]
- All O ~holding(O,s0)
- Eternals:
  - gold(g1) ^ adjacent([1,1],[1,2]) ^ adjacent([1,2],[1,1]).

15

Goal: g1 is in [1,1]

At(g1,[1,1],resultSeq(
[go([1,1],[1,2]),grab(g1),go([1,2],[1,1])],
s0)

Or, planning by answering the query:

Exists S at(g1,[1,1],resultSeq(S,s0))

So, what has to go in the KB for such queries to be answered?...

16

Axioms for our Wumpus World

- For brevity: we will omit universal quantifiers that range over entire sentence. S ranges over situations, A ranges over actions, O over objects (including agents), G over gold, and X,Y,Z over locations.

17

Possibility and Effect Axioms

- Possibility axioms:
  - Preconditions → poss(A,S)

- Effect axioms:
  - poss(A,S) → changes that result from that action

18

Possibility Axioms

- The possibility axioms that an agent can
  - go between adjacent locations,
  - grab a piece of gold in the current location, and
  - release gold it is holding
    - Holding(g,s) => Poss(Release(g),s)

19

Effect Axioms

- If an action is possible, then certain fluents will hold in the situation that results from executing the action
  - Going from X to Y results in being at Y
  - Grabbing the gold results in holding the gold
  - Releasing the gold results in not holding it
    - Poss(Release(g),s)=>~Holding(g,Result(Release(g),s))

20

Frame Problem

- We run into the frame problem
- Effect axioms say what changes, but don't say what stays the same
- A real problem, because (in a non-toy domain), each action affects only a tiny fraction of all fluents

21

Frame Problem (continued)

- One solution approach is writing explicit frame axioms, such as:

At(O,X,S) ^ ~(O=agent) ^ ~holding(O,S) → at(O,X,result(Go(Y,Z),S))

22

Representational Frame Problem

- What stays the same?
- A actions, F fluents, and E effects/action (worst case). Typically, E << F
- Want O(AE) versus O(AF) solution

23

Solving the Representational Frame Problem

- Instead of writing the effects of each action, consider how each fluent predicate evolves over time
- Successor-state axioms:
- Action is possible →

  (fluent is true in result state ←→

   action's effect made it true v

   it was true before and action left it alone)

24

Example

- **Initial state**: a logical sentence about (situation) S_0
  At(Home, S_0) ^ ~Have(Milk, S_0) ^ ~ Have(Bananas, S_0) ^ ~Have(Drill, S_0)

- **Goal state**:
  (∃s) At(Home,s) ^ Have(Milk,s) ^ Have(Bananas,s) ^ Have(Drill,s)

- **Operators** :
  ∀(a,s) Have(Milk,Result(a,s)) <=> ((a=Buy(Milk) ^ At(Grocery,s)) ∨
  (Have(Milk, s) ^ a~=Drop(Milk)))

25

Ramification Problem

- Implicit effects, such as: if an agent moves from X to Y, then any gold it is carrying will move too
- For our specific domain, we can solve this by writing a more general successor-state axiom for "at"

26

Qualification Problem

- Ensuring that all necessary conditions for an action's success have been specified. No complete solution.

27

Blocks world example

- A situation calculus rule for the blocks world:
  - Clear (X, Result(A,S)) ↔
    [Clear (X, S) ∧
      (¬(A=Stack(Y,X) ∨ A=Pickup(X))
      ∨ (A=Stack(Y,X) ∧ ¬(holding(Y,S))
      ∨ (A=Pickup(X) ∧ ¬(handempty(S) ∧ ontable(X,S) ∧ clear(X,S))))]
    ∨ [A=Stack(X,Y) ∧ holding(X,S) ∧ clear(Y,S)]
    ∨ [A=Unstack(Y,X) ∧ on(Y,X,S) ∧ clear(Y,S) ∧ handempty(S)]
    ∨ [A=Putdown(X) ∧ holding(X,S)]
- English translation: A block is clear if (a) in the previous state it was clear and we didn't pick it up or stack something on it successfully, or (b) we stacked it on something else successfully, or (c) something was on it that we unstacked successfully, or (d) we were holding it and we put it down.
- Whew!!! There's gotta be a better way!

28

Situation calculus planning: Analysis

- This is fine in theory, but remember that problem solving (search) is exponential in the worst case
- Also, resolution theorem proving only finds *a* proof (plan), not necessarily a good plan
- So we restrict the language and use a special-purpose algorithm (a planner) rather than general theorem prover

29

Basic representations for planning

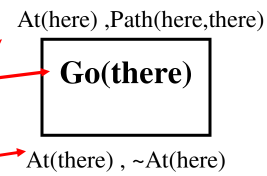
- Classic approach first used in the **STRIPS** planner circa 1970
- States represented as a conjunction of ground literals
  - at(Home) ^ ~have(Milk) ^ ~have(bananas) ...
- Goals are conjunctions of literals, but may have variables which are assumed to be existentially quantified
  - at(?x) ^ have(Milk) ^ have(bananas) ...
- Do not need to fully specify state
  - Non-specified either don't-care or assumed false
  - Represent many cases in small storage
  - Often only represent changes in state rather than entire situation
- Unlike theorem prover, not seeking whether the goal is true, but is there a sequence of actions to attain it

30

Operator/action representation

- Operators contain three components:
  - **Action description**
  - **Precondition** - conjunction of positive literals
  - **Effect** - conjunction of positive or negative literals which describe how situation changes when operator is applied
- Example:

  Op[Action: Go(there),
      Precond: At(here) ^ Path(here,there),
      Effect: At(there) ^ ~At(here)]



- All variables are universally quantified
- Situation variables are implicit
  - preconditions must be true in the state immediately before operator is applied; effects are true immediately after

31

Blocks world operators

- Here are the classic basic operations for the blocks world:
  - stack(X,Y): put block X on block Y
  - unstack(X,Y): remove block X from block Y
  - pickup(X): pickup block X
  - putdown(X): put block X on the table
- Each will be represented by
  - a list of preconditions
  - a list of new facts to be added (add-effects)
  - a list of facts to be removed (delete-effects)
  - optionally, a set of (simple) variable constraints
- For example:

  preconditions(stack(X,Y), [holding(X),clear(Y)])
  deletes(stack(X,Y), [holding(X),clear(Y)]).
  adds(stack(X,Y), [handempty,on(X,Y),clear(X)])
  constraints(stack(X,Y), [X ~= Y,Y ~= table,X ~= table])

32

Blocks world operators II

operator(stack(X,Y),
    Precond [holding(X),clear(Y)],
    Add [handempty,on(X,Y),clear(X)],
    Delete [holding(X),clear(Y)],
    Constr [X ~=Y,Y ~=table,X ~= table]).

operator(unstack(X,Y),
    [on(X,Y), clear(X), handempty],
    [holding(X),clear(Y)],
    [handempty,clear(X),on(X,Y)],
    [X ~= Y,Y ~= table, X ~= table]).

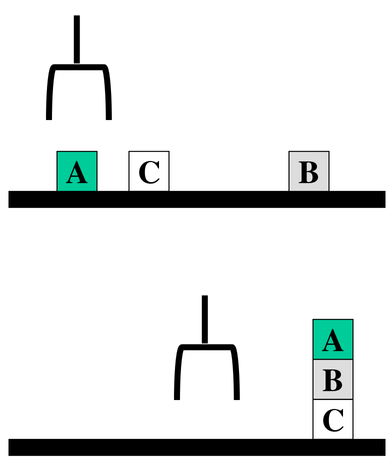
operator(pickup(X),
    [ontable(X), clear(X), handempty],
    [holding(X)],
    [ontable(X),clear(X),handempty],
    [X ~= table]).

operator(putdown(X),
    [holding(X)],
    [ontable(X),handempty,clear(X)],
    [holding(X)],
    [X ~= table]).

Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(b,c)
- on(a,b)
- ontable(c)



A plan:
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

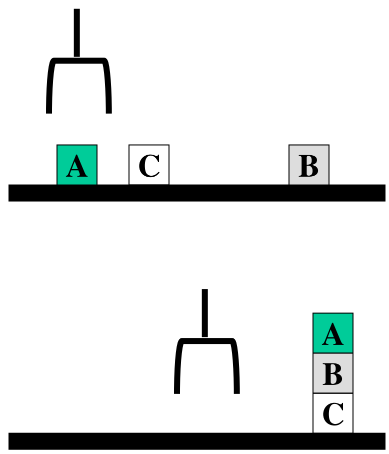
Another BW planning problem

Initial state:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Goal:

on(a,b)
on(b,c)
ontable(c)



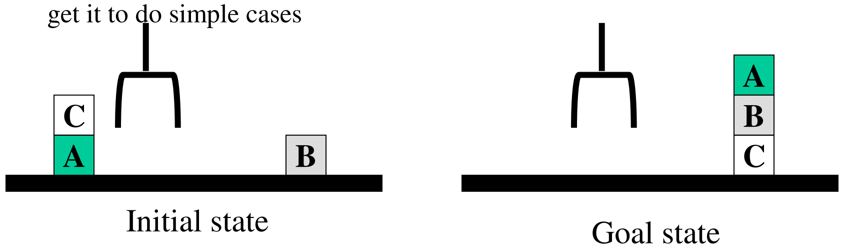
A plan:

pickup(a)
stack(a,b)
unstack(a,b)
putdown(a)
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

35

Goal interaction

- Simple planning algorithms assume that the goals to be achieved are independent
  - Each can be solved separately and then the solutions concatenated
- This planning problem, called the "Sussman Anomaly," is the classic example of the goal interaction problem:
  - Solving on(A,B) first (by doing unstack(C,A), stack(A,B) will be undone when solving the second goal on(B,C) (by doing unstack(A,B), stack(B,C)).
  - Solving on(B,C) first will be undone when solving on(A,B)
- Classic STRIPS could not handle this, although minor modifications can get it to do simple cases



36

State-space planning

- We initially have a space of situations (where you are, what you have, etc.)
- The plan is a solution found by "searching" through the situations to get to the goal
- A **progression planner** searches forward from initial state to goal state
- A **regression planner** searches backward from the goal
  - This works if operators have enough information to go both ways
  - Ideally this leads to reduced branching –you are only considering things that are relevant to the goal

37

Plan-space planning

- An alternative is to **search through the space of *plans***, rather than situations.
- Start from a **partial plan** which is expanded and refined until a complete plan that solves the problem is generated.
- **Refinement operators** add constraints to the partial plan and modification operators for other changes.
- We can still use STRIPS-style operators:

  Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

  Op(ACTION: RightSock, EFFECT: RightSockOn)

  Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

  Op(ACTION: LeftSock, EFFECT: leftSockOn)

could result in a partial plan of

  [RightShoe, LeftShoe]

38

Partial-order planning

- A **linear planner** builds a plan as a **totally ordered sequence** of plan steps
- A **non-linear planner (aka partial-order planner)** builds up a plan as a set of steps with some temporal constraints
  - constraints of the form S1<S2 if step S1 must comes before S2.
- One **refines** a partially ordered plan (POP) by either:
  - **adding a new plan step**, or
  - **adding a new constraint** to the steps already in the plan.
- A POP can be **linearized** (converted to a totally ordered plan) by topological sorting

39

Least commitment

- Non-linear planners embody the principle of **least commitment**
  - only choose actions, orderings, and variable bindings that are absolutely necessary, leaving other decisions till later
  - avoids early commitment to decisions that don't really matter
- A linear planner always chooses to add a plan step in a particular place in the sequence
- A non-linear planner chooses to add a step and possibly some temporal constraints

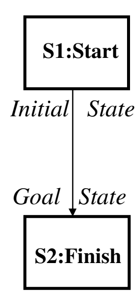
40

Non-linear plan

- A non-linear plan consists of
  - (1) A set of **steps** $\{S_1, S_2, S_3, S_4 \ldots\}$
    - Each step has an operator description, preconditions and post-conditions
  - (2) A set of **causal links** $\{ \ldots (S_i,C,S_j) \ldots\}$
    - Meaning a purpose of step S_i is to achieve precondition C of step S_j
  - (3) A set of **ordering constraints** $\{ \ldots S_i<S_j \ldots \}$
    - if step S_i must come before step S_j
- A non-linear plan is **complete** iff
  - Every step mentioned in (2) and (3) is in (1)
  - If S_j has prerequisite C, then there exists a causal link in (2) of the form (S_i,C,S_j) for some S_i
  - If (S_i,C,S_j) is in (2) and step S_k is in (1), and S_k threatens (S_i,C,S_j) (makes C false), then (3) contains either $S_k<S_i$ or $S_j>S_k$

The initial plan

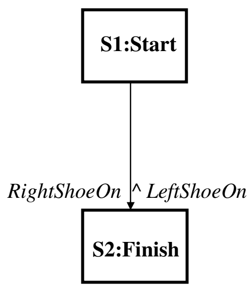
Every plan starts the same way



Trivial example

Operators:

Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
Op(ACTION: RightSock, EFFECT: RightSockOn)
Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
Op(ACTION: LeftSock, EFFECT: leftSockOn)



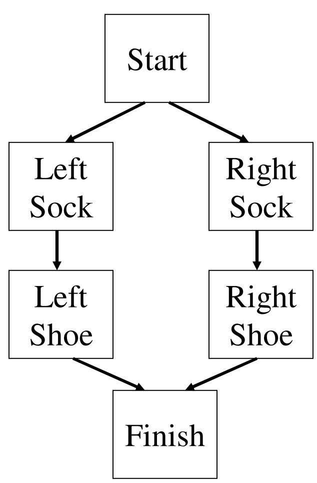
Steps: {S1:[Op(Action:Start)],
S2:[Op(Action:Finish,
Pre: RightShoeOn^LeftShoeOn)]}

Links: {}

Orderings: {S1<S2}

43

Solution



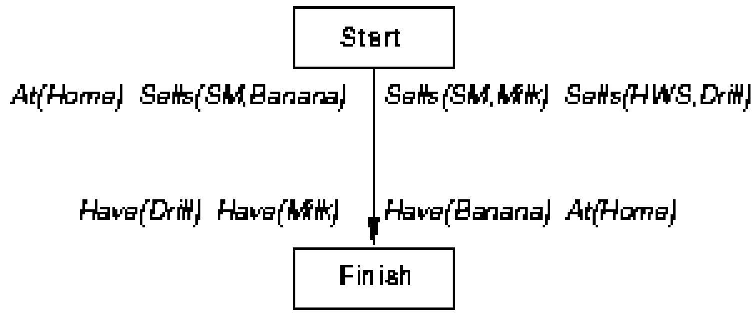
44

POP constraints and search heuristics

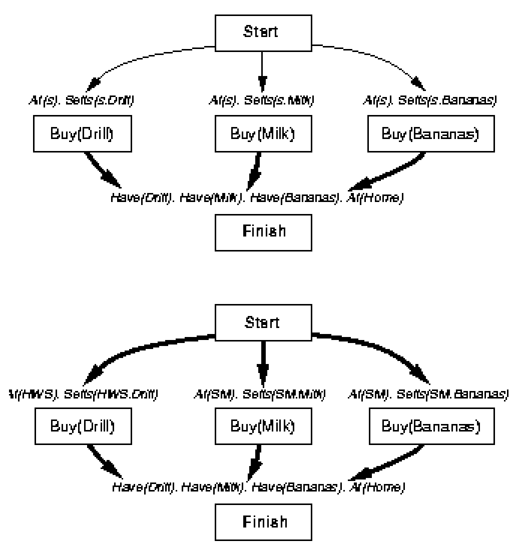
- Only add steps that achieve a currently unachieved precondition
- Use a least-commitment approach:
  - Don't order steps unless they need to be ordered
- Honor causal links $S_1 \xrightarrow{c} S_2$ that **protect** a condition *c*:
  - Never add an intervening step S_3 that violates *c*
  - If a parallel action **threatens** *c* (i.e., has the effect of negating or **clobbering** *c*), resolve that threat by adding ordering links:
    - Order S_3 before S_1 (**demotion**)
    - Order S_3 after S_2 (**promotion**)

Partial-order planning example

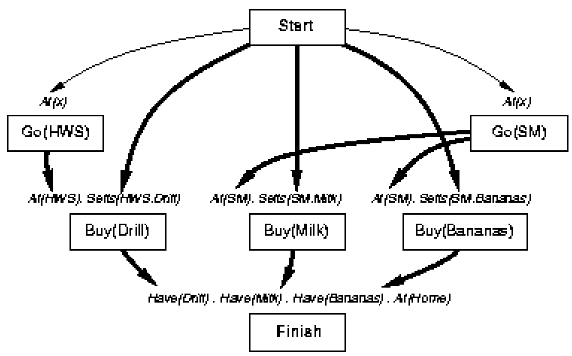
- Goal: Have milk, bananas, and a drill


Start
At(Home) Sells(SM,Banana) Sells(SM,Milk) Sells(HWS,Drill)
Have(Drill) Have(Milk) Have(Banana) At(Home)
Finish

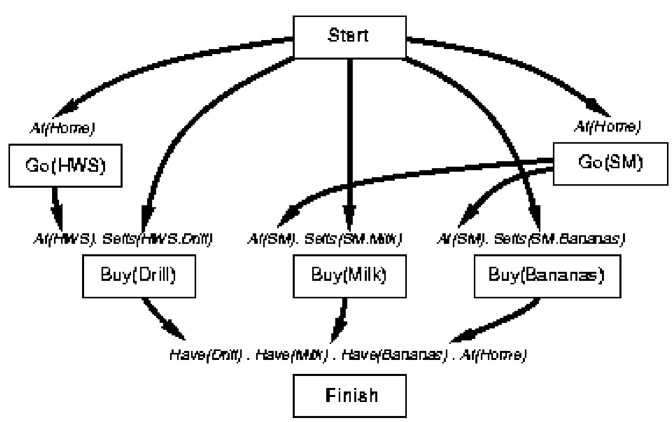
47


Start
At(s). Sells(s,Drill)
At(s). Sells(s,Milk)
At(s). Sells(s,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill). Have(Milk). Have(Bananas). At(Home)
Finish
Start
t(HWS). Sells(HWS,Drill)
At(SM). Sells(SM,Milk)
At(SM). Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill). Have(Milk). Have(Bananas). At(Home)
Finish

48

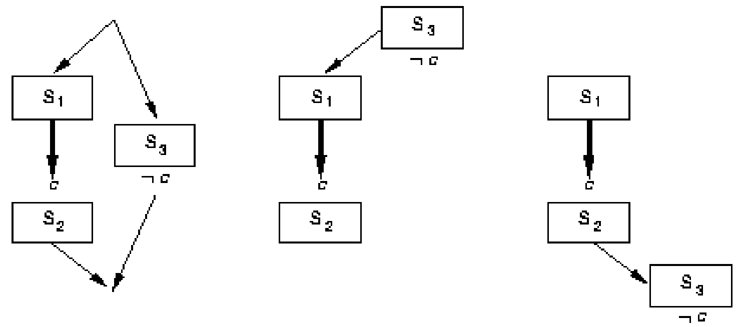

Start
At(x)
At(x)
Go(HWS)
Go(SM)
At(HWS). Sells(HWS,Drill)
At(SM). Sells(SM,Milk)
At(SM). Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) . Have(Milk) . Have(Bananas) . At(Home)
Finish

49

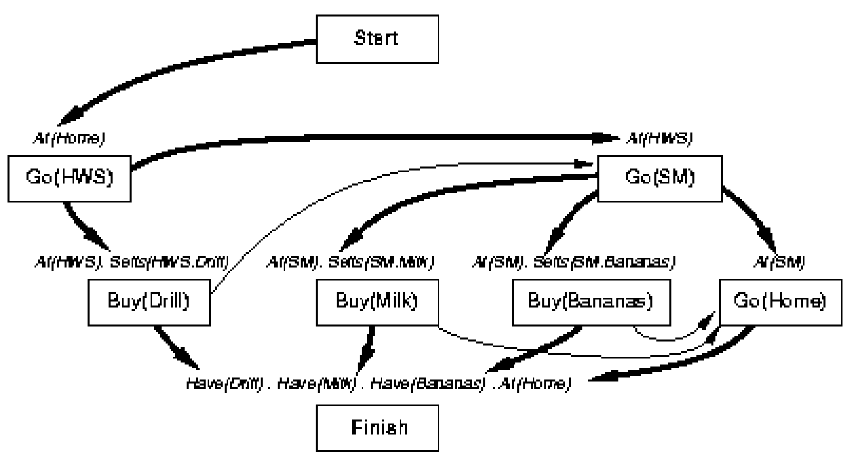

Start
At(Home)
At(Home)
Go(HWS)
Go(SM)
At(HWS). Sells(HWS,Drill)
At(SM). Sells(SM,Milk)
At(SM). Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) . Have(Milk) . Have(Bananas) . At(Home)
Finish

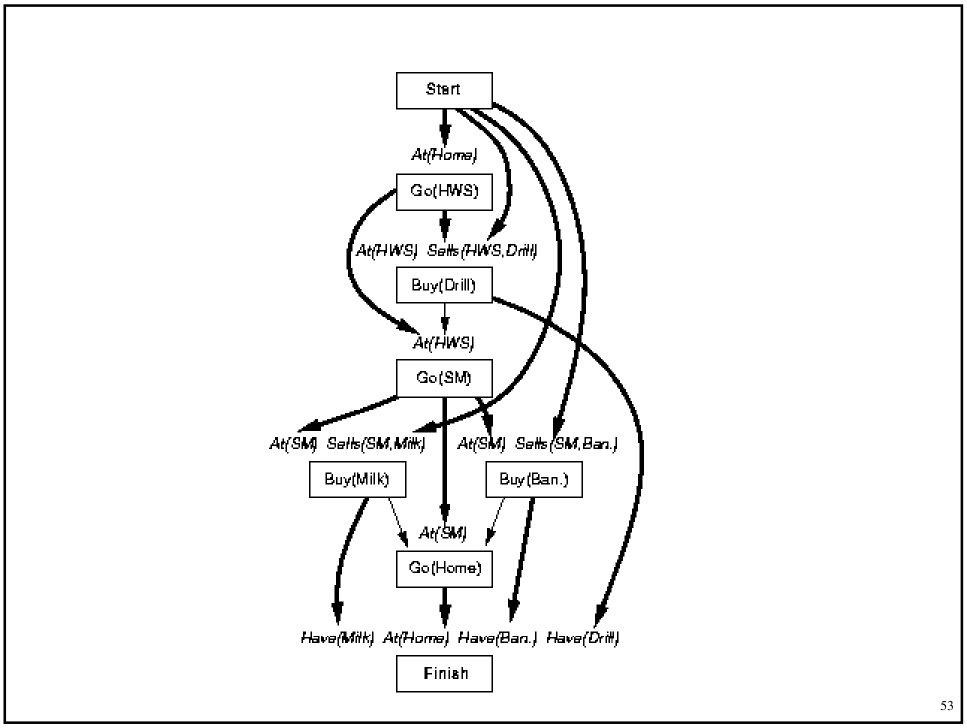
50

Resolving threats



(a) Threat (b) Demotion (c) Promotion




Start
At(Home)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM) Sells(SM,Milk)
At(SM) Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish
53