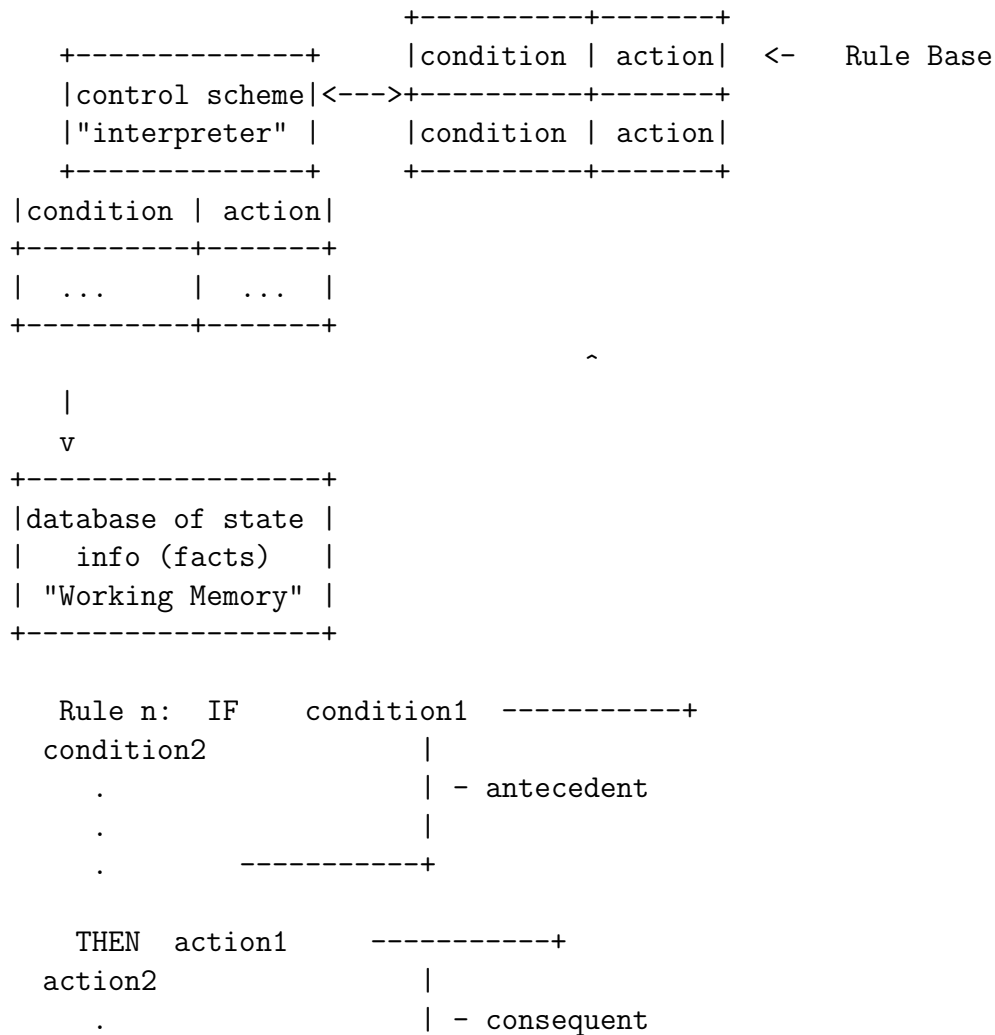
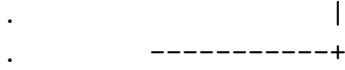

Expert Systems

- ES perform tasks for specific domains that require human expertise
Medical diagnosis, fault diagnosis, status monitoring, data interpretation, computer configuration, etc.
 - ES solve problems using domain-specific knowledge
 - Domain knowledge is acquired by interviewing human experts
 - ES cannot operate in situations requiring common sense
-

Rule-Based Systems





When one part of the IF portion matches a fact in working memory, the antecedent is SATISFIED. When all antecedents are satisfied, the rule is TRIGGERED. When the consequent of a rule is performed, the rule is FIRED.

Three Phases

1. Match phase

Match left side of rules (antecedents) with facts in working memory

Unification

2. Conflict resolution

Of all the rules that are triggered, decide which rule to fire.

Some strategies include:

- Do not duplicate rule with same instantiated arguments twice
- Prefer rules that refer to recently created WM elements
- Prefer rules that are more specific (antecedents are more constraining)

`prefer`

`Mammal(x) & Human(x) -> add Legs(x,2)`

`over`

`Mammal(x) -> add Legs(x,4)`

- If rules are ranked, fire according to ranking

3. Act phase

Add or delete facts to WM as specified by rule consequent

Expert Systems at a Glance

ES are one of AI's early showpieces.

ES use rule-based systems along with additional capabilities

Components of an ES include:

Evidence gathering

Control of inference
Uncertainty reasoning
Explanation generation and tutoring
User interface
Validation
Social issues

Some Famous Early Expert Systems

- Dendral (Stanford, Ed Feigenbaum, 1965) - organic chemical analysis
 - Macsyma (MIT, 1965) - symbolic math problems
 - Mycin (Stanford, 1972) - diagnose blood diseases
 - Prospector (SRI, 1972) - mineral exploration
 - Caduceus (1975) - internal medicine
 - Xcon and R1 (1980, 1982) - computer system configuration (for DEC)
 - Harpy - document retrieval
 - Hearsay - Speech understanding
 - Max (Nynex, 1992) - telephone network troubleshooting bacterial infection diagnosis
-

Examples

- - Some expert system shells
 -
 - KEE
 -
-

Mycin Example

Rules

```
PREMISE ($AND (SAME CNTXT GRAM GRAMNEG)
              (SAME CNTXT MORH ROD)
              (SAME CNTXT AIR AEROBIC))
ACTION: (CONCLUDE CNTXT CLASS ENTEROBACTERIACEAE .8)
```

```
If   the stain of the organism is gramneg, and
     the morphology of the organism is rod, and
     the aerobiticity of the organism is aerobic
Then there is strongly suggestive evidence (.8) that
     the class of organism is enterobacteriaceae
```

Data

```
ORANISM-1:
GRAM = (GRAMNEG 1.0)
MORP = (ROD .8) (COCCUS .2)
AIR = (AEROBIC .6) (FACUL .4)
```

Forward Chaining and Backward Chaining

There are two main types of control schemes that are applied to rule-based systems.

```
Z1   If    ?x has hair
     Then  ?x is a mammal
Z2   If    ?x gives milk
     Then  ?x is a mammal
Z3   If    ?x has feathers
     Then  ?x is a bird
Z6   If    ?x is a mammal
     ?x has pointed teeth
     ?x has claws
     ?x has forward-pointing eyes
     Then  ?x is a carnivore
Z8   If    ?x is a mammal
     ?x chews cud
     Then  ?x is an ungulate
Z11  If    ?x is an ungulate
     ?x has long legs
     ?x has long neck
```

?x has tawny color
?x has dark spots
Then ?x is a giraffe

...

Database

F1) Stretch has hair
F2) Stretch chews cud
F3) Stretch has long legs
F4) Stretch has a long neck
F5) Stretch has tawny color
F6) Stretch has dark spots

Forward Chaining

Reasons FORWARD from facts/rules to (hopefully) a needed goal

Uses modus ponens Know A -> B
 Know A

 Add B

Rule antecedents are compared with facts from database.

If match, add consequents to database

Repeat as long as needed

Forward chaining is “data driven”

* Match Z1 & F1
 Add: Stretch is a mammal

* Match Z8/1 & F7 ((?x Stretch))
 Match Z8/2 & F2
 Add: Stretch is an ungulate

Note: Actually, Z5/1, Z6/1, and Z7/1 would all be matched before Z8/1

* Match Z11/1 & F8 ((?x Stretch))
 Match Z11/2 & F3
 Match Z11/3 & F4
 Match Z11/4 & F5

Match Z11/5 & F6
Add: Stretch is a giraffe

Backward Chaining

Reasons BACKWARD from goal through rules to facts

Uses modus ponens	To Prove	B
	Know	A \rightarrow B

	Prove	A

Start at goals
Match goals to consequents or facts
If match consequents, antecedents become new subgoals
Repeat until
All subgoals proven
or
At least one subgoal cannot be proven
Backward chaining is “goal driven”

Backward Chaining Example

Goal 1: Stretch is a giraffe
Match: Goal 1 and Z11/C (does not match with any facts)
Subgoal 2: Stretch is an ungulate
Subgoal 3: Stretch has long legs
Subgoal 4: Stretch has long neck
Subgoal 5: Stretch has tawny color
Subgoal 6: Stretch has dark spots
Match: Subgoal 2 and Z8/C (does not match with any facts)
Subgoal 7: Stretch is a mammal
Subgoal 8: Stretch chews cud
Match: Subgoal 7 and Z1/C (does not match with any facts)
Subgoal 9: Stretch has hair
Match: Subgoal 9 and F1
Subgoals 9, 7, met
Match: Subgoal 8 and F2
Subgoals 8, 2 met
Match: Subgoal 3 and F3
Subgoal 3 met

Match: Subgoal 4 and F4
Subgoal 4 met
Match: Subgoal 5 and F5
Subgoal 5 met
Match: Subgoal 6 and F6
Subgoal 6 met, Goal 1 met

Forward Chaining vs. Backward Chaining

High fan out - use backward chaining

1) Human(Albert)
2) Human(Alfred)
3) Human(Barry)
4) Human(Charlie)
...
50) Human(Highlander)
...
100) Human(Shaun)
...
500) Human(Zelda)
501) Human(x) -> Mortal(x)
502) Mortal(x) -> CanDie(x)

Can we kill Shaun Connery?

FC

--

503) Mortal(Albert)
504) Mortal(Alfred)
...
1003) CanDie(Albert)
1004) CanDie(Alfred)
...
1100) CanDie(Shaun)

BC

--

Prove: CanDie(Shaun)
Match: Goal and 502/C
Prove: Mortal(Shaun)
Match: Mortal(Shaun) and 501/C
Prove: Human(Shaun)

Match: Human(Shaun) and 100
Done!

Forward Chaining may generate a lot of useless facts

If ?x has feathers	If ?x has feathers
Then ?x can fly	Then ?x can be used as a pen

If ?x has feathers
Then ?x is a bird
...

1 condition - many actions USE BC!

More Forward Chaining vs. Backward Chaining

High fan in - use forward chaining

Potential Problem with BC - if many subgoals, each must be examined

If	?x has feathers
	?x is brown
	?x sings @#!@#! patterns
	?x sleeps on one foot
	?x makes a good pot pie
	?x lives in tree
Then	?x is a bird

Many conditions - 1 action

- If time is crucial and you only have 1 goal to prove, use BC
- If you have spare time and want to be prepared for future questions, use FC to generate all possible facts

Or, use bi-directional search

Limitation of ES

- A lot of matches!

- May perform many matches for Rule 1, then have to perform same matches for Rule 2 (should share partial match information)
- May match first few antecedents, but fail on last. If last antecedent added later, have to start again at beginning of rule (should save partial match information)

One solution: RETE net (RETE stands for “network”)

Contains: alpha nodes (one for each antecedent)

beta nodes (combination of matches)

terminal nodes (one for each rule)

Explicitly store shared and partial match information

Planning

What is planning?

Strategies required to achieve a goal or to solve a particular problem

Need for Planning

If our goal is to create an autonomous, intelligent entity, we need to study planning.

Logic, natural language, learning, vision, are all useful, and they would make nice human-driven tools.

But what is the entity going to DO???

It has to decide WHAT it wants to do, and HOW to do it.

Planning Search Space

Suppose problem is “Get a quart of milk and a two eggs (to make pancakes) and a variable-speed cordless drill (to fix house when done)”.

- We search through a space of states
 - We try each possible action
 - Thousands of actions, millions of states
 - The heuristic function can direct search, but doesn’t prune states
-

Planning Key 1

In planning we will modify our representation of the search space (states, goals, and actions) to allow more efficient reasoning.

Instead of search through states from initial state to goal state, we can look for actions that seem relevant and are accessible.

If goal includes Have(Milk)

and Buy(x) achieves Have(x)

then agent should consider plan that includes Buy(Milk).

Do not consider Buy(WhippingCream) or GoToSleep.

Planning Key 2

The planner can add actions to the plan wherever they are needed, instead of always at the end (or beginning) of the plan.

Agent may want to Buy(Milk), but doesn't know how to get there, how to buy it, what to do after that.

Make "obvious" or "important" decisions first.

Planning Key 3

Parts of the world are independent from other parts.

We can solve one part of conjunctive goal with one plan, another part of the goal with another plan.

Planning Representation

- States are represented as a conjunction of instantiated literals (no functions)

$At(Home) \wedge \neg Have(Milk) \wedge \neg Have(Bananas) \wedge \neg Have(Drill) \wedge \dots$

States can be incomplete - if don't specify, then negation is assumed.

If Have(Unicorn) is not included, assume $\neg Have(Unicorn)$.

This is the **Closed-World Assumption** .

- Goals are described as conjunction of literals (possibly with variables).

$At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$

or

$At(x) \wedge Sells(x, Milk)$ if goal is to be at store that sells milk.

Variables must be existential (like theorem prover goal).
Here, we are not seeing if database entails goal.
We are trying to transform state to one that includes goal.

Planning Representation

- STRIPS Operators

Precondition: Conjunction of atoms (positive literals) that must be true to apply operator

Effect: Conjunction of literals that describes how the situation changes when operator is applied

Example OP Go(?there)

Precondition: $At(?here) \wedge Path(?here, ?there)$

Effect: $At(?there) \wedge \neg At(?here)$

State Space Planner

Operators can have variables - then we **unify** goals with facts.

We call this type of planner a “situation space” planner, or “state space” planner, because nodes in search space represent states or situations.

This type of planner completely solves one goal, then tacks on plan for next goal.

“Progression planner” if search forward (A^*) This prunes options if high fan-in

“Regression planner” if search backward (GPS) This prunes options if high fan-out

Example State Space Planner

General Problem Solver, GPS

Uses Means-Ends Analysis

Limitations

Notice what happens when we apply STRIPS planning to the following problem.

	+----+	OPERATOR:	PUTON(X, Y)
	A		
+----+	+----+	Precondition:	Clear(X)

C		B		Y=TABLE or Clear(Y)
+----+	+----+	+----+		
A	B	C	Add:	On(X, Y)
-+----+----+----+		----+----+----	Delete:	On(X, ~)
Initial State:		Goal State:		
On(C,A)		ON(A,B)		
ON(A, TABLE)		ON(B,C)		
ON(B, TABLE)				
CLEAR(C)				
CLEAR(B)				

In which order do we try to achieve the goals?

If 1) ON(A,B)

Try: PUTON(C, TABLE) to clear A
 PUTON(A, B) to achieve first goal

2) ON(B,C)

To achieve this goal, B will be re-cleared, undoing the first goal.

If 1) ON(B,C)

Try: PUTON(B,C)

2) ON(A,C)

We are farther from this goal now than we were in the initial state!

This planning problem is known as "Sussman's Anomaly".

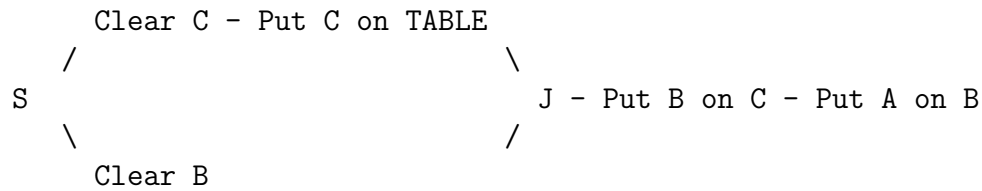
Sussman's Anomaly

This problem can be solved, but it cannot be attacked by first applying all the operators to achieve one goal, and then applying operators to achieve another goal.

The problem is that we have forced an ORDERING on the operators. Sometimes steps for multiple goals need to be interleaved.

Partial-order planning is a type of plan generation in which ordering is imposed on operators ONLY when it has to be imposed in order to achieve the goals.

Plan Space Planning



- Node (state space): world state
- Node (plan space): partial plan
- Operator (state space): next step in plan sequence
- Operator (plan space)
 - Add a link from existing action to open condition (precondition that is not yet fulfilled)
 - Add a plan step to fulfill open condition
 - Order a step in the sequence

Gradually move from incomplete/vague plans to complete plans.
We call planners that use this type of approach [partial-order planners](#)

Partially-Ordered Plans

The initial plan has only two steps (Start and Finish), 1 ordering (Start before Finish).
A sock must be put on before a shoe.

Partially-Ordered Plans

One partial-order plan can have many linearizations (total orderings), as seen in this example.

Partially-Ordered Plans

A plan in this scenario consists of the following components:

1. Set of plan steps
2. Set of ordering constraints (Step i must occur some time before Step j)
3. Variable binding constraints ($v=x$)

4. Causal links ($S_i \xrightarrow{c} S_j$) which reads “Step i achieves condition c for Step j” .
This is also referred to as a **protection interval** . If Step i achieves condition c for Step j, no one better remove c during this interval.
-

Clobbering and Promotion / Demotion

A **clobberer** is a potentially intervening step that destroys the condition achieved by a causal link.

We can prevent clobbering by ordering the clobberer *before* the start of the link or *after* the end of the link.

Example

In this example, Go(Home) clobbers At(HWS).

Example

Here is the shopping problem initial incomplete plan.

Example

The top plan achieves three of the four Finish preconditions, the heavy arrows show causal links. The bottom plan refines the top plan by adding causal links to achieve the Sells preconditions of the Buy steps.

Example

This partial plan achieves the At preconditions of the three Buy actions.

Example

This is a flawed plan for directing the agent to the hardware store and the supermarket.

There is no way to resolve the threat that each Go step poses to the other.

We now backtrack in the search process.

Example

The next choice is to achieve the $At(x)$ precondition of the $Go(SM)$ step by adding a causal link from $Go(HWS)$ to $Go(SM)$.

The $Go(SM)$ step now threatens the $At(HWS)$ precondition of the $Buy(Drill)$ step, and is resolved by promotion.

Example

The $At(Home)$ precondition of the $Finish$ step is still unachieved.

- If the planner links $At(Home)$ in the initial state to $Finish$, there will be no way to resolve the threats raised by $Go(HWS)$ and $Go(SM)$.
- If the planner tries to link $At(x)$ to $Go(HWS)$, there will be no way to resolve the threat posed by $Go(SM)$, which is ordered after $Go(HWS)$.
- The planner links $Go(SM)$ to $At(x)$, so x is bound to SM and $Go(Home)$ deletes the $At(SM)$ condition, resulting in threats to the $At(SM)$ precondition for $Buy(Milk)$ and $Buy(Bananas)$. These threats are resolved by promoting $Go(Home)$.

Here is the final plan.

Sussman's Anomaly Partial Plan

Plan 0:

```
Steps:  Start(0), Finish(1: precond=(On A B) & (On B C))
Order:  0 before 1
Bind:   none (Ignore bindings for now)
Links:  none
```

What is wrong with this plan? Preconds of 1 are not met. Try adding step.

Plan 1: (other sibling would add $Puton(B,C)$)

```
Steps:  Start{0}, Finish{1: pre=(On A B)-2 & (On B C)}
        Puton(A, B){2: pre=(Clear A) & (Clear B)-0}
Order:  0 before 1, 2 before 1
Links:  2 establishes (On A B) for 1 - No step removes (On A B), ok
        0 establishes (Clear B) for 2
```

The other pre of 1 is not met.

Plan 2: (siblings would address pre of 2)

Steps: Start{0}, Finish{1: pre=(On A B)-2 & (On B C)-3}
 Puton(A, B){2: pre=(Clear A) & (Clear B)-0}
 Puton(B, C){3: pre=(Clear B)-0 & (Clear C)}

Order: 0 before 1, 2 before 1, 0 before 2, 0 before 3,
 3 before 2, 3 before 1

Links: 2 establishes (On A B) for 1, order 2 before 1 -
 No step removes (On A B), ok
 0 establishes (Clear B) for 2, order 0 before 2
 0 establishes (Clear B) for 3, order 0 before 3
 2 clobbers (Clear B), order 3 before 2
 3 establishes (On B C) for 1, order 3 before 1
 No step removes (On B C), ok

One pre of 2 is not met.

Plan 3:

Steps: Start{0}, Finish{1: pre=(On A B)-2 & (On B C)-3}
 Puton(A, B){2: pre=(Clear A)-4 & (Clear B)-0}
 Puton(B, C){3: pre=(Clear B)-0 & (Clear C)-0}
 Puton(C, Table){4: pre=(Clear C)-0}

Order: 0 before 1, 2 before 1, 0 before 2, 0 before 3,
 3 before 2, 3 before 1, 4 before 2, 4 before 3

Links: 2 establishes (On A B) for 1, order 2 before 1
 No step removes (On A B), ok
 0 establishes (Clear B) for 2, order 0 before 2
 0 establishes (Clear B) for 3, order 0 before 3
 2 clobbers (Clear B), order 3 before 2
 3 establishes (On B C) for 1, order 3 before 1
 No step removes (On B C), ok
 4 establishes (Clear A) for 2, order 4 before 2
 No step removes (Clear A), ok
 3 clobbers (Clear C), order 4 before 3
 0 establishes (Clear C) for 4, order 0 before 4

0->4->3->2->1

Shakey

STRIPS was originally designed to control SRI's Shakey robot, using the operators Go(x), Push(b,x,y), Climb(b), Down(b), TurnOnLight(s), and TurnOffLight(s).

Planning Graphs

More efficient and accurate planning technique.

The idea is to build a graph with n layers (the maximum plan length is thus n). Every level in the graph corresponds to a plan time step.

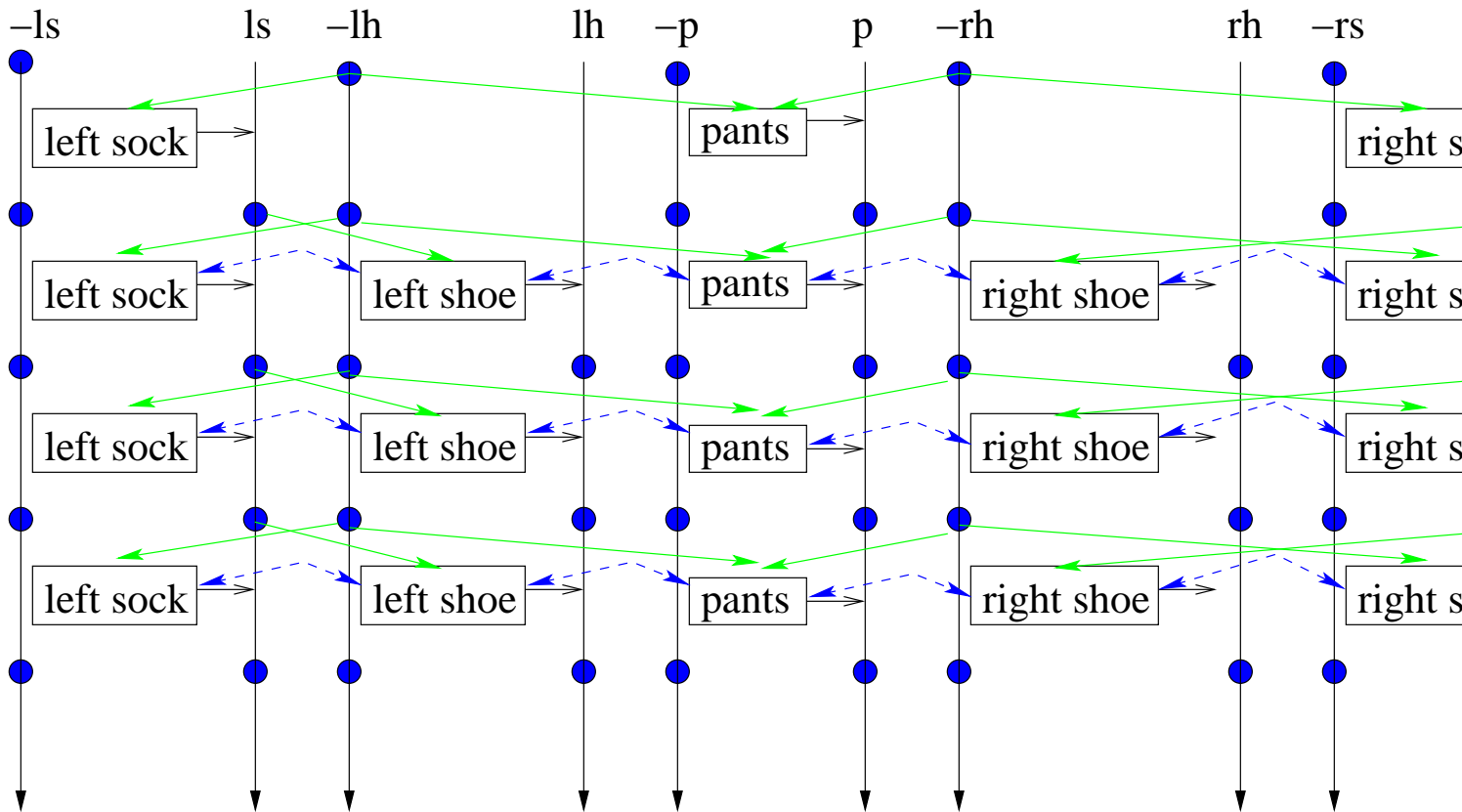
The initial state is level 0

Graph

- Each level contains a set of propositions and a set of actions
 - Propositions at level i could be true in situation S_i depending on actions selected earlier
 - Actions at level i could be executed (preconditions satisfied), depending on which propositions are true in S_i
 - Every level i contains all literals that result from any possible choice of actions in the previous level
 - Constraints (negative interactions among actions) are noted
-

Example

- Propositions
 - ls, rs (wearing left/right sock)
 - lh, rh (wearing left/right shoe)
 - p (wearing pants)
- Actions
 - left/right sock
 - preconditions = -ls/-rs, -lh/-rh
 - delete = -ls/-rs
 - add = ls/rs
 - left/right shoe
 - preconditions = -lh/-rh, ls/rs
 - delete = -lh/-rh
 - add = lh/rh
 - pants
 - preconditions = -p, -lh/-rh
 - delete = -p
 - add = p



Generating Planning Graph

- First level is initial state
- List possible actions, generate next level as resulting propositions
- Persistence actions are used to represent propositions that remain true from one step to the next
- Mutual exclusion (mutex) links connect actions that cannot co-occur
 - Mutex two actions if one action negates an effect of the other (inconsistent effects)
 - Mutex two actions if one action effect is the negation of a precondition of the other action (interference)
 - Mutex two actions if the preconditions of the actions are contradictory (competing needs)
- Mutex links also connect literals that cannot appear together
 - One literal is the negation of the other
 - The actions that generate the two literals are mutex (inconsistent support)

- Continue generating graph until two consecutive levels are identical (graph has leveled off)
 - Alternatively, continue generating graph until a level contains the goal
-

Search Through Graph

One graph is generated, search backward from goal to initial state for plan.

Goals at time t

- ls, rs, lh, rh, p

Select set of non-mutex actions at time t

- pants, left sock, right sock
- left shoe, right shoe

Goals at time t-1

- -p, -ls, -rs, -lh, -rh, lh, rh
These goals are contradictory, so cannot use
 - -lh, -rh, ls, rs, p
-

Search Through Graph

Select set of non-mutex actions at time t-1

- left sock, right sock, pants

Goals at time t-2

- -p, -ls, -rs, -lh, -rh

These are satisfied by initial conditions.

The final plan is

1. left sock, right sock, pants
 2. left shoe, right shoe
-

Analysis

Efficient

Graphplan is Complete. If Graphplan does not generate a solution, then no solution exists.

Hierarchical Planning

Reduce complexity of planning by planning in multiple levels

Example: Attending IJCAI in Stockholm

Classical Planner:

Plan to call taxi

Plan to get to front door

Plan to get to taxi

Plan route to airport

Plan route to ticket counter

...

Plan airplane route from Dallas to Japan

...

ABSTRIPS Methodology

- Plan at most abstract level
 - Completely ignore predicates labeled as “less critical”
 - Using abstract plan, fill in details for next level of abstraction
 - Complete preconditions, adding predicates for next lower level of criticality
 - Repeat until reached ground (know exactly how to execute it) level
 - Issues
 - How form hierarchies?
 - How do levels interact?
-

Example

Operator: `PushThruDoor(bx, dx, rx)`

Preconditions:

(6) `Pushable(bx)`

(6) `Isa(dx, Door)`

(6) Isa(rx, Room)
(2) Status(dx, Open)
(1) NextTo(bx, dx)
(1) NextTo(Robot, bx)
(5) Inroom(bx, ry)
(6) Inroom(Robot, ry)
(6) Connects(dx, ry, rx)

Similar example with lightbulbs

STRIPS: 119 nodes, 23 on successful path, 30 minutes CPU

ABSTRIPS: 60 nodes, 54 on successful path, 5:28 CPU (1/5 time)

Additional Planning Considerations

- Resource Constraints
 - Conditional Planning
 - Monitoring / Replanning
 - Scalability
GraphPlan
-

Planning in Practice

- NASA New Millenium Program
Programs on board spacecraft perform science planning and scheduling
Execute plans without human intervention
- Hitachi's O-PLAN
Process planning and scheduling of 30-day assembly schedules
- Desert Storm Planning
Plan for transportation of troops, supplies, weapons