

Automatic Planning - Linear Planning

- Solving problem methodology
- Problems are described using a predicate calculus logic formalism (situation calculus)
- A problem is composed by:
 - A description of the initial state
 - A description of the goals
 - A set of planning operators
- A search algorithm will be used to obtain a sequence of operators

Lineal Planning (STRIPS)

Operator

- An operator is defined by:
 - A set of preconditions
 - The predicates that will be true in the state (add list)
 - The predicates that will be false in the state (delete list)
- Preconditions and effects are expressed in predicate calculus

Plan

- A plan is defined by:
 - An initial state and a goal state
 - An ordered list of instantiated operators
- The search is performed in the state space
- A search strategy has to be defined (for instance: depth first, breadth first, means end analysis, ...)

Linear Planning algorithm (I)

```
function LP ( $e_{initial}$ ,  $e_{goal}$ , operators) returns plan  
  state  $\leftarrow e_{initial}$   
  stack.push( $G(e_{goal})$ )  
  while  $\neg$ stack.empty?() do  
    goal  $\leftarrow$  stack.pop()  
    switch  
      case: goal is composed goal  
        if  $\neg$  valid(goal, estate) then  
          stack.push(goal)  
          goals[]  $\leftarrow$  choose_new_order_for_subgoals(goal)  
          for each g in goals[] do stack.push( $G(g)$ )  
        eof  
      ecase
```

Linear Planning algorithm (II)

case: *goal* is simple goal

if $\neg\text{valid}(\textit{goal}, \textit{state})$ **then**

Choose an operator not used (*op*) that adds *goal*

if that operator exists **then**

Unify variables of the operator so *goal* holds

stack.push(obj)

stack.push(A(op))

stack.push(G(preconditions(op)))

endif

endif

ecase

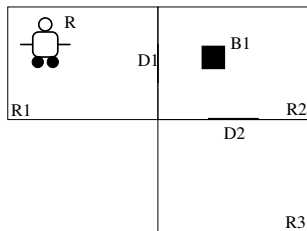
Lineal Planning algorithm (III)

```
case: goal is apply operator
  if preconditions of goal hold in the current state then
    state ← apply(goal,state)
    plan.insert(goal)
  endif
ecase
eswitch
ewhile
return(plan)
efunction
```

Operators

- `go(robot, door, room1, room2)`
 - Precondition: `connected(door, room1, room2), in(room1, robot)`
 - Delete: `in(room1, robot)`
 - Add: `in(room2, robot)`
- `push(box, door, room1, room2)`
 - Precondition: `connected(door, room1, room2), in(room1, robot), in(room1, box)`
 - Delete: `in(room1, robot), in(room1, box)`
 - Add: `in(room2, robot), in(room2, box)`

Initial and goal state



$\text{in}(R1, \text{Robot})$

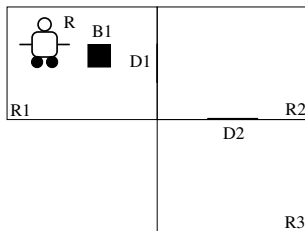
$\text{in}(R2, B1)$

$\text{connected}(D1, R1, R2)$

$\text{connected}(D1, R2, R1)$

$\text{connected}(D2, R2, R3)$

$\text{connected}(D2, R3, R2)$



$\text{in}(R1, B1)$

Resolution of the plan (1)

1. Goal = $\text{in}(\text{R1}, \text{B1})$

The operator that allows to move a box from one place to another is **push**. If we unify with the add list $[\text{in}(\text{R1}, \text{B1}) = \text{in}(\text{room2}, \text{obj})] \rightarrow [\text{room2}=\text{R1}, \text{obj}=\text{B1}]$

The preconditions that we have are: $\text{connected}(\text{door}, \text{room1}, \text{R1})$, $\text{in}(\text{room1}, \text{R})$ and $\text{in}(\text{room1}, \text{B1})$, we look for an unification that gives the new goals (for instance: $[\text{room1}=\text{R2}, \text{door}=\text{D1}]$)

We add the operator $\text{push}(\text{B1}, \text{D1}, \text{R2}, \text{R1})$

2. Goal = $\text{connected}(\text{D1}, \text{R2}, \text{R1})$, $\text{in}(\text{R2}, \text{B1})$, $\text{in}(\text{R2}, \text{R})$

We create a goal for each subgoal

3. Goal = $\text{connected}(\text{D1}, \text{R2}, \text{R1}) \rightarrow$ holds in the current state
4. Goal = $\text{in}(\text{R2}, \text{B1}) \rightarrow$ holds in the current state

Resolution of the plan (2)

5. Goal = $\text{in}(\text{R2}, \text{R})$

The operator that moves the robot is **go**. If we unify with the add list $[\text{in}(\text{R2}, \text{R}) = \text{in}(\text{room2}, \text{R})] \rightarrow [\text{room2} = \text{R2}]$

We will have the preconditions: $\text{connected}(\text{door}, \text{room1}, \text{R2})$, $\text{in}(\text{room1}, \text{R})$, we look for an unification to obtain the new goals (for instance $[\text{room1} = \text{R1}, \text{door} = \text{D1}]$)

We add the operator $\text{go}(\text{D1}, \text{R1}, \text{R2})$

6. Goal = $\text{connected}(\text{D1}, \text{R1}, \text{R2})$, $\text{in}(\text{R1}, \text{R})$

We create a goal for each subgoal

7. Goal = $\text{connected}(\text{D1}, \text{R1}, \text{R2}) \rightarrow$ holds in the current state

8. Goal = $\text{in}(\text{R1}, \text{R}) \rightarrow$ holds in the current state

Resolution of the plan (and 3)

9. We execute the action $go(D1,R1,R2)$ modifying the state, now $in(R2,R)$ will be true in the state and $in(R1,R)$ will be no longer true
10. Goal = $in(R2,R) \rightarrow$ holds in the current state
11. We execute the action $push(B1,D1,R2,R1)$ modifying the state, now $in(R1,R)$ and $in(R1,B1)$ will be true in the state and $in(R2,R)$ and $in(R2,B1)$ will be no longer true
12. Goal = $in(R1,B1) \rightarrow$ holds in the current state

Now the goal stack is empty, so we have arrived to the goal state with the plan:

$$E_{initial} \rightarrow go(D1,R1,R2) \rightarrow push(B1,D1,R2,R1) \rightarrow E_{goal}$$

Building the triangular table

- First column, first row = initial state
- For the rest
 - (column $i+1$, row i) = Operator i of the plan
 - (column 1 to i , row $i+1$) = (column 1 to i , row i) except the predicates the operator has deleted
 - column $i+1$, row $i+1$ = The predicates that the operator adds

Plan - Triangular table

in(R1,R) in(R2,B1) connected(D1,R1,R2) connected(D1,R2,R1)	go(D1,R1,R2)	
in(R2,B1) connected(D1,R1,R2) connected(D1,R2,R1)	in(R2,R)	push(B1,D1,R2,R1)
connected(D1,R1,R2) connected(D1,R2,R1)		in(R1,R) in(R1,B1)

Plan - Only relevant predicates

in(R1,R) connected(D1,R1,R2)		go(D1,R1,R2)
in(R2,B1) connected(P1,H2,H1)	in(R2,R)	push(B1,D1,R2,R1)
		in(R1,R) in(R1,B1)

We delete from the table all predicates that are not used to prove the preconditions of the operators.

Variabilization of the triangular table

- We assign different variables to each constant in the predicate from the first column
- We variabilize the parameters of the operators and propagate the substitutions accordingly in the corresponding column

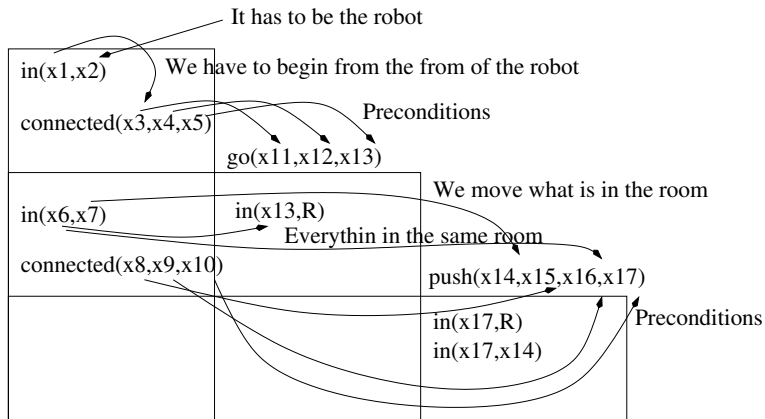
Plan - Variabilization

in(x1,x2) connected(x3,x4,x5)		
	go(x11,x12,x13)	
in(x6,x7) connected(x8,x9,x10)	in(x13,R)	
		push(x14,x15,x16,x17)
		in(x17,R) in(x17,x14)

Constraint of the triangular table

- We unify the variables of the operators using the preconditions as guide
- The restriction has to be performed so the original proofs of the precondition still hold

Plan - Constraints



$x1=x4=x12$ $x2=R$ $x3=x11$ $x5=x13=x6=x9=x16$
 $x7=x14$ $x8=x15$ $x10=x17$

Plan - Final table

in(x1,R) connected(x3,x1,x5)	go(x3,x1,x5)	
in(x5,x7) connected(x8,x5,x10)	in(x5,R)	push(x7,x8,x5,x10)
		in(x10,R) in(x10,x7)

New operator

op: move(x1,x3,x5,x7,x8,x10)

prec: in(x1,R), connected(x3,x1,x5),in(x5,x7),connected(x8,x5,x10)

add: in(x10,R),in(x10,x7)

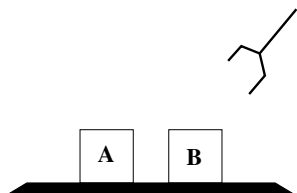
del: in(x1,R),in(x5,x7)

- Precondition: Relevant predicate from the first column
- Add: Predicates from the last row except the ones from the first column
- Delete: Differences among the initial state and the predicates from the last row of the first column

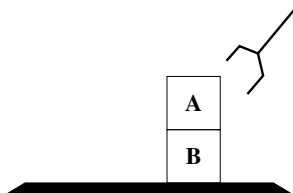
Operators

pick(x):	Precondition: on_table(x), clear(x), emptyhand Delete: on_table(x), clear(x), emptyhand Add: on_hand(x)
unpick(x):	Precondition: on_hand(x) Delete: on_hand(x) Add: on_table(x), clear(x), emptyhand
stack(x,y):	Precondition: on_hand(x), clear(y) Delete: on_hand(x), clear(y) Add: emptyhand, on(x,y), clear(x)
unstack(x,y):	Precondition: emptyhand, clear(x), on(x,y) Delete: emptyhand, clear(x), on(x,y) Add: on_hand(x), clear(y)

Initial and goal states



clear(A)
 clear(B)
 emptyhand
 on_table(A)
 on_table(B)



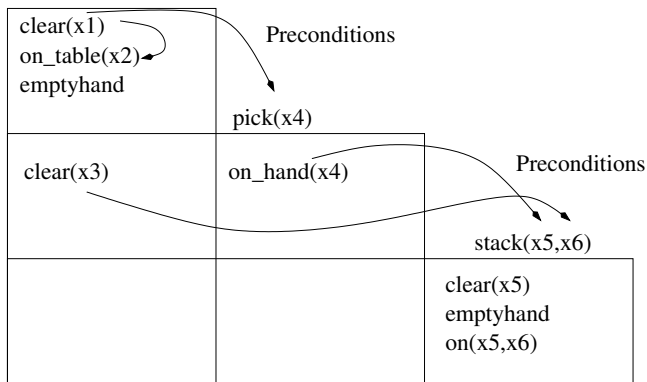
on(A,B)

Plan: pick(A) \rightarrow stack(A,B)

Plan - Triangular table

clear(A) clear(B) on_table(A) on_table(B) emptyhand	pick(A)	
clear(B) on_table(B)	on_hand(A)	stack(A,B)
on_table(B)		clear(A) emptyhand on(A,B)

Plan - Generalized table



$x1=x2=x4=x5$ $x3=x6$

Plan - Constrained table

clear(x1) on_table(x1) emptyhand		
	pick(x1)	
clear(x3)	on_hand(x1)	
		stack(x1,x3)
		clear(x1) emptyhand on(x1,x3)

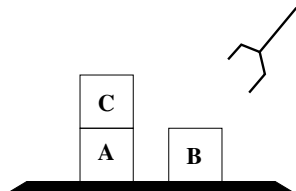
New operator

op: on-block(x1,x3)
prec: clear(x1),on_table(x1),emptyhand,clear(x3)
add: on(x1,x3),clear(x1),emptyhand
del: clear(x3),clear(x1),on_table(x1),emptyhand

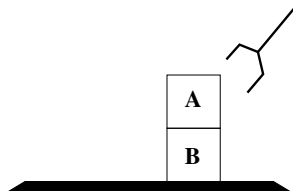
Sometimes the operator can be simplified

op: on-block(x1,x3)
prec: clear(x1),on_table(x1),emptyhand,clear(x3)
add: on(x1,x3)
del: clear(x3),on_table(x1)

Initial and goal states



clear(B)
 clear(C)
 emptyhand
 on_table(A)
 on_table(B)
 on(C,A)



on(A,B)
 clear(A)

Plan: unstack(C,A) \rightarrow unpick(C) \rightarrow pick(A) \rightarrow stack(A,B)

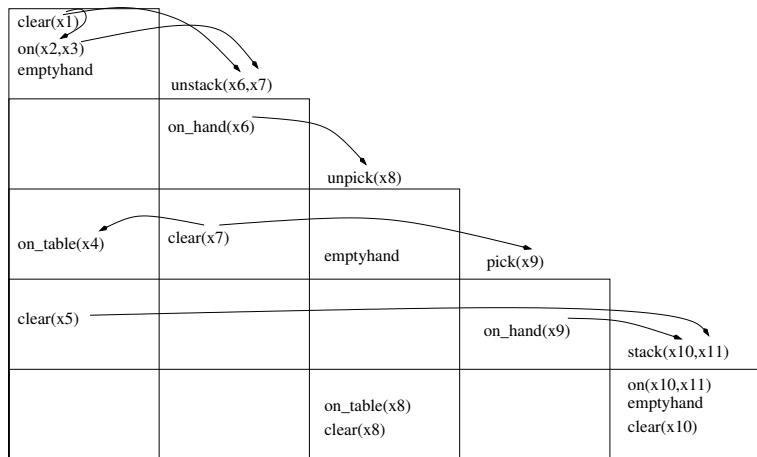
Plan - Triangular table

clear(C) clear(B) on_table(A) on_table(B) emptyhand on(C,A)	unstack(C,A)			
clear(B) on_table(B) on_table(A)	on_hand(C) clear(A)	unpick(C)		
clear(B) on_table(B) on_table(A)	clear(A)	on_table(C) clear(C) emptyhand	pick(A)	
on_table(B) clear(B)		on_table(C) clear(C)	on_hand(A)	stack(A,B)
on_table(B)		on_table(C) clear(C)		on(A,B) emptyhand clear(A)

Plan - Table - relevant predicates

clear(C) on(A,C) emptyhand	unstack(C,A)			
	on_hand(C)			
		unpick(C)		
on_table(A)	clear(A)	emptyhand	pick(A)	
clear(B)			on_hand(A)	
				stack(A,B)
		on_table(C) clear(C)	Not deleted by any operator	on(A,B) emptyhand clear(A)

Plan - Generalized table



$x1=x2=x6=x8$ $x3=x7=x4=x9=x10$ $x5=x11$

Plan - Constrained table

clear(x1) on(x1,x3) emptyhand	unstack(x1,x3)			
	on_hand(x1)			
		unpick(x1)		
on_table(x3)	clear(x3)	emptyhand	pick(x3)	
clear(x5)			on_hand(x3)	stack(x3,x5)
		on_table(x1) clear(x1)		on(x4,x5) emptyhand clear(x4)

New operator

op: clear-put(x1,x3,x5)
prec: clear(x1),on_table(x5),emptyhand,clear(x5),on(x1,x3)
add: on(x3,x5),clear(x3),emptyhand,clear(x1),on_table(x1)
del: clear(x5),on(x1,x3),on_table(x3),emptyhand,clear(x1)

Effect of learning new operators

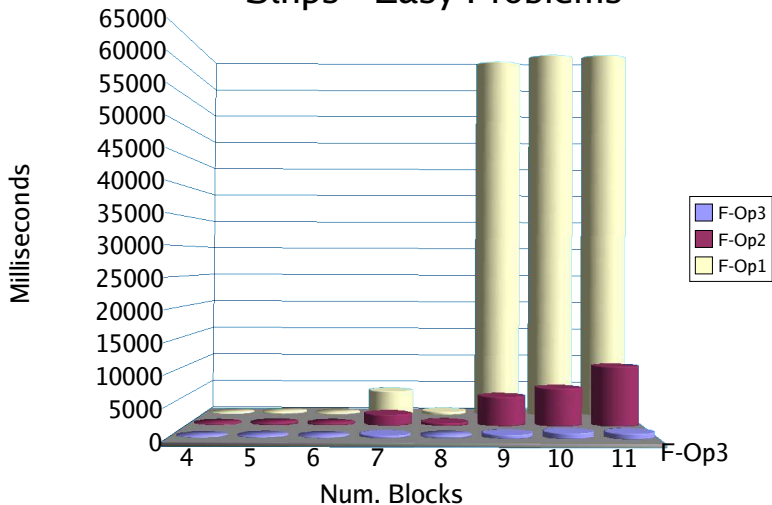
- To learn new operators from simple plans can improve the resolution time
- The problems used to learn the new operators have to be selected carefully
- The cost of having to check the new operators during the resolution have to be compensated by the resolution time saved

Effect of learning new operators - Example

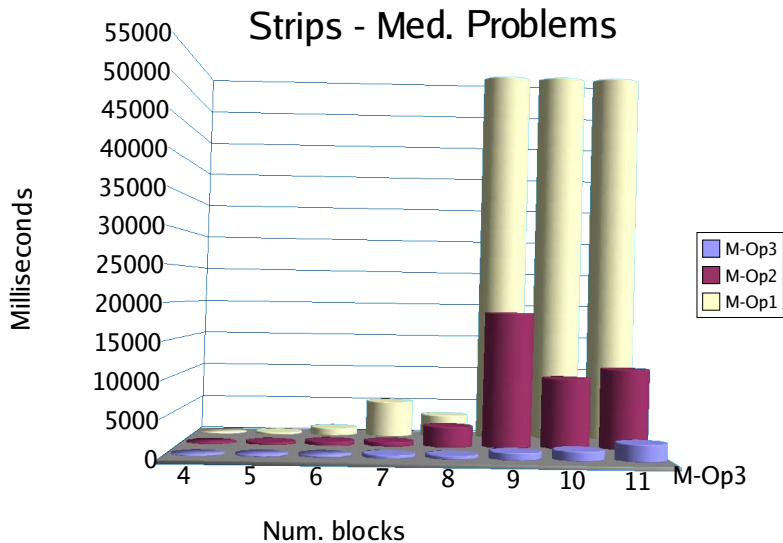
- In the previous example we can add to the initial operators the one learned **on-block** and another with the opposite effect (**off-block**)
- We can measure the time we save solving different problems of increasing difficulty with the original operators, adding the new operators and only using the new operators

Easy problems

Strips - Easy Problems



Medium problems



Difficult problems

