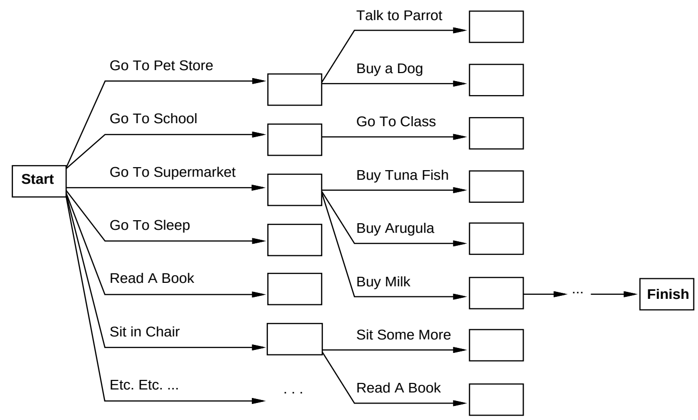
Planning

Planning

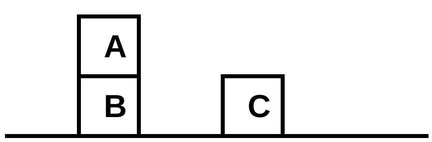
- Planning is a particular type of problem solving in which actions and goals are declaratively specified in logic and generally concerns performing actions in the real world.
- By “opening up” the representation of states, goals, and actions (instead of treating them as black boxes) a planner can make more direct connections between them.



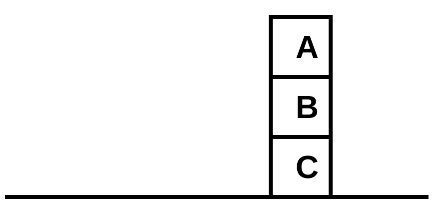
- Frequently, subgoals are independent and problems can be solved by divide and conquer.

Sample Problem
Blocks World

Initial state



Goal State



Situation Calculus

- Formalization of actions in first-order logic (McCarthy & Hayes, 1969). Search performed by logical inference.
- Situations are logical terms denoting states of the world.
- Actions and facts are represented as logical terms called **fluents**.

  puton(A,B): The action of putting block A on block B.
  on(A,B): The proposition that block A is on block B.

- Propositional fluents are asserted to be true in a particular state by using the predicate: holds

  holds(on(A,B), s): A is on B in situation s

- Situations resulting from performing an action in another situation are represented using the function: result

  result(puton(A,B), s): The situation resulting from putting A on B in situation s.

Situation Calculus (cont)

- Axioms used to represent **preconditions** and **effects** of actions.

$\forall s \forall x \forall y[\text{holds}(\text{clear}(x),s) \wedge \text{holds}(\text{clear}(y),s) \rightarrow$
$\text{holds}(\text{on}(x,y), \text{result}(\text{puton}(x,y),s))\ ]$

- However, must also explicitly state what does **not** change when an action is performed.

$\forall s \forall x \forall y \forall c[\text{holds}(\text{color}(x,c),s) \rightarrow$
$\text{holds}(\text{color}(x,c),\text{result}(\text{puton}(x,y),s))]$

- These are called **frame axioms** and the fact that so many must be provided is called the **frame problem**.

- Other more sophisticated logics such as temporal and modal logics have also been developed for reasoning about actions.

Deductive Planning

- For situation calculus, prove a theorem of the following form to construct a plan for initial state φ and goal ψ:

$\forall s[\text{holds}(\varphi,s) \rightarrow \exists z(\text{holds}(\psi,z) \wedge \text{reachable}(z,s)]$

- For goal of having A on B when initially both are on the table:

$\forall s[\text{holds}(\text{on}(A,\text{Table}),s) \wedge \text{holds}(\text{on}(B,\text{Table}),s) \rightarrow$
$\exists z(\text{holds}(\text{on}(A,B),z) \wedge \text{reachable}(z,s))]$

- Need a constructive prove that produces a value for z that represents a plan.

z= result(puton(A,B),s)

z = result(puton(A,B),result(puton(B,C),s))

- Approach first implemented by Green (1969)

Situation Calculus in Prolog

```
holds(on(A,B),result(puton(A,B),S)) :-
    holds(clear(A),S), holds(clear(B),S), neq(A,B).

holds(clear(C),result(puton(A,B),S)) :-
    holds(clear(A),S), holds(clear(B),S), holds(on(A,C),S),
    neq(A,B).

holds(on(X,Y),result(puton(A,B),S)) :-
    holds(on(X,Y),S), neq(X,A), neq(Y,A), neq(A,B).

holds(clear(X),result(puton(_A,B),S)) :-
    holds(clear(X),S), neq(X,B).

holds(clear(table),_S).

neq(a,table).
neq(table,a).
neq(b,table).
neq(table,b).
neq(c,table).
neq(table,c).
neq(a,b).
neq(b,a).
neq(a,c).
neq(c,a).
neq(b,c).
neq(c,b).
```


Situation Calculus Planner

```
plan([],_,_).
plan([G1|Gs], S0, S) :-
    holds(G1,S),
    plan(Gs, S0, S),
    reachable(S,S0).

reachable(S,S).
reachable(result(_,S1),S) :-
    reachable(S1,S).
```

However, what will happen if we try to make plans using normal Prolog depth-first search?

Situation Calculus Results

- Stack of 3 blocks

holds(on(a,b), s0).
holds(on(b,table), s0).
holds(on(c,table),s0).
holds(clear(a), s0).
holds(clear(c), s0).

| ?- cpu_time(db_prove(6,plan([on(a,b),on(b,c)],s0,S)), T).

S =
result(puton(a,b),result(puton(b,c),result(puton(a,table),s0)))
T = 1.3433E+01

- Invert stack

holds(on(a,table), s0).
holds(on(b,a), s0).
holds(on(c,b),s0).
holds(clear(c), s0).

?- cpu_time(db_prove(6,plan([on(b,c),on(a,b)],s0,S)),T).

S =
result(puton(a,b),result(puton(b,c),result(puton(c,table),s0)))
,T = 7.034E+00

Situation Calculus Results (Cont)

- O.K. Let's try a simple four block stack.

holds(on(a,table), s0).
holds(on(b,table), s0).
holds(on(c,table),s0).
holds(on(d,table),s0).
holds(clear(c), s0).
holds(clear(b), s0).
holds(clear(a), s0).
holds(clear(d), s0).

| ?-
cpu_time(db_prove(7,plan([on(b,c),on(a,b),on(c,d)],s0,S)),T
).

S =
result(puton(a,b),result(puton(b,c),result(puton(c,d),s0))),
T = 2.765935E+04

7.5 hours!

STRIPS

- Developed at SRI (Stanford Research Institute) in early 1970's.
- Just using theorem proving with situation calculus was found to be too inefficient.
- Introduced STRIPS action representation.
- Combines ideas from problem solving and theorem proving.
- Basic backward chaining in state space but solves subgoals independently and then tries to reachieve any clobbered subgoals at the end.

STRIPS Representation

- Attempt to address the frame problem by defining actions by a *precondition*, and *add list,* and a *delete list*. (Fikes & Nilsson, 1971).

  Precondition: logical formula that must be true in order to execute the action.
  Add list: List of formulae that become true as a result of the action.
  Delete list: List of formulae that become false as result of the action.

  Puton(x,y)
  Precondition: Clear(x) $\wedge$ Clear(y) $\wedge$ On(x,z)
  Add List: {On(x,y), Clear(z)}
  Delete List: {Clear(y), On(x,z)}

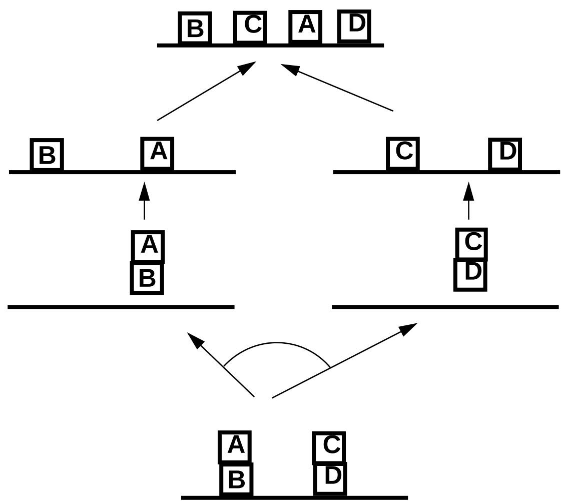
- STRIPS assumption: Every formula that is satisfied before an action is performed and does not belong to the delete list is satisfied in the resulting state.

  Although Clear(z) implies that On(x,z) must be false, it must still be listed in the delete list explicitly.

  For action Kill(x,y) must put Alive(y), Breathing(y), Heart-Beating(y), etc. must all be included in the delete list although these deletions are implied by the fact of adding Dead(y)

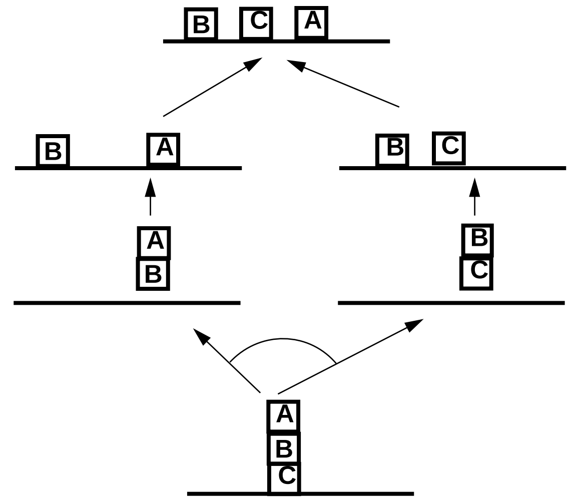
Subgoal Independence

- If the goal state is a conjunction of subgoals, search is simplified if goals are assumed independent and solved separately (divide and conquer)



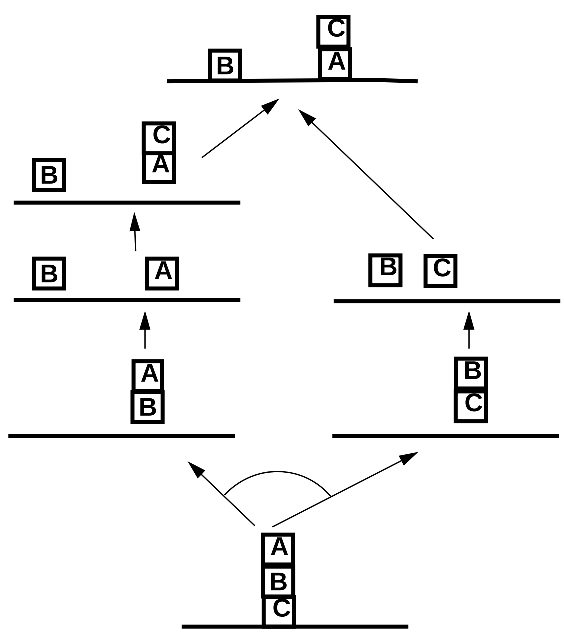
Subgoal Interaction

Achieving different subgoals may interact, the order in which subgoals are solved in this case is important.



If do puton(B,C) first, no problem.
If do puton(A,B) first, cannot do puton(B,C) without undoing (clobbering) subgoal: on(A,B)

Sussman Anomoly



Either way of ordering the subgoals causes clobbering.

STRIPS Approach

- Use resolution theorem prover to try and prove that goal or subgoal is statisfied in the current state.
- If it is not, use the incomplete proof to find a set of differences between the current and goal state (a set of subgoals).
- Pick a subgoal to solve and an operator that will achieve that subgoal.
- Add the precondition of this operator as a new goal and recursively solve it.

STRIPS Algorithm

```
STRIPS(init-state, goals, ops)
   Let current-state be init-state;
   For each goal in goals do
       If goal cannot be proven in current state
           Pick an operator instance, op, s.t. goal ∈ adds(op);
           ;; Solve preconditions
           STRIPS(current-state, preconds(op), ops);
           ;; Apply operator
           current-state := current-state + adds(op) - dels(ops);
   ;; Patch any clobbered goals
   Let rgoals be any goals which are not provable in
   current-state;
   STRIPS(current-state, rgoals, ops).
```

The “pick operator instance” step involves a nondeterministic choice that is backtracked to if a dead-end is ever encountered.

Employs **chronological backtracking** (depth-first search), when reach dead-end, backtrack to last decision point and pursue the next option.

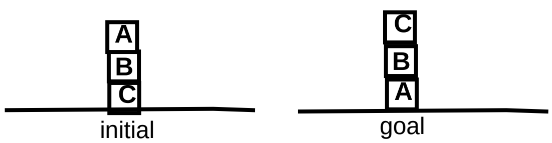
Norvig’s Implementation

- Simple propositional (no variables) Lisp implementation of STRIPS.

```
#S(OP ACTION (MOVE C FROM TABLE TO B)
   PRECONDS ((SPACE ON C) (SPACE ON B) (C ON TABLE))
   ADD-LIST ((EXECUTING (MOVE C FROM TABLE TO B)) (C ON B))
  DEL-LIST ((C ON TABLE) (SPACE ON B)))
```

- Commits to first sequence of actions that achieves a subgoal (incomplete search).
- Prefers actions with the most preconditions satisfied in the current state.
- I modified to to try and reachieve any clobbered subgoals (only once).

STRIPS Results




```
; Invert stack (good goal ordering)
> (gps '((a on b)(b on c) (c on table) (space on a) (space on table))
    '((b on a) (c on b)))
Goal: (B ON A)
Consider: (MOVE B FROM C TO A)
  Goal: (SPACE ON B)
  Consider: (MOVE A FROM B TO TABLE)
    Goal: (SPACE ON A)
    Goal: (SPACE ON TABLE)
    Goal: (A ON B)
  Action: (MOVE A FROM B TO TABLE)
  Goal: (SPACE ON A)
  Goal: (B ON C)
Action: (MOVE B FROM C TO A)
Goal: (C ON B)
Consider: (MOVE C FROM TABLE TO B)
  Goal: (SPACE ON C)
  Goal: (SPACE ON B)
  Goal: (C ON TABLE)
Action: (MOVE C FROM TABLE TO B)
((START)
 (EXECUTING (MOVE A FROM B TO TABLE))
 (EXECUTING (MOVE B FROM C TO A))
 (EXECUTING (MOVE C FROM TABLE TO B)))
```


More STRIPS Results

```
; Invert stack (bad goal ordering)
> (gps '((a on b)(b on c) (c on table) (space on a) (space on table))
    '((c on b)(b on a)))
Goal: (C ON B)
Consider: (MOVE C FROM TABLE TO B)
  Goal: (SPACE ON C)
  Consider: (MOVE B FROM C TO TABLE)
    Goal: (SPACE ON B)
    Consider: (MOVE A FROM B TO TABLE)
      Goal: (SPACE ON A)
      Goal: (SPACE ON TABLE)
      Goal: (A ON B)
    Action: (MOVE A FROM B TO TABLE)
    Goal: (SPACE ON TABLE)
    Goal: (B ON C)
  Action: (MOVE B FROM C TO TABLE)
  Goal: (SPACE ON B)
  Goal: (C ON TABLE)
Action: (MOVE C FROM TABLE TO B)
Goal: (B ON A)
Consider: (MOVE B FROM TABLE TO A)
  Goal: (SPACE ON B)
  Consider: (MOVE C FROM B TO TABLE)
    Goal: (SPACE ON C)
    Goal: (SPACE ON TABLE)
    Goal: (C ON B)
  Action: (MOVE C FROM B TO TABLE)
  Goal: (SPACE ON A)
  Goal: (B ON TABLE)
Action: (MOVE B FROM TABLE TO A)
```

Must reachieve clobbered goals: ((C ON B))
Goal: (C ON B)
Consider: (MOVE C FROM TABLE TO B)
  Goal: (SPACE ON C)
  Goal: (SPACE ON B)
  Goal: (C ON TABLE)
Action: (MOVE C FROM TABLE TO B)
((START)
 (EXECUTING (MOVE A FROM B TO TABLE))
 (EXECUTING (MOVE B FROM C TO TABLE))
 (EXECUTING (MOVE C FROM TABLE TO B))
 (EXECUTING (MOVE C FROM B TO TABLE))
 (EXECUTING (MOVE B FROM TABLE TO A))
 (EXECUTING (MOVE C FROM TABLE TO B)))

STRIPS on the Sussman Anomaly

```
> (gps '((c on a)(a on table)( b on table) (space on c) (space on b)
        (space on table))  '((a on b)(b on c)))
Goal: (A ON B)
Consider: (MOVE A FROM TABLE TO B)
  Goal: (SPACE ON A)
  Consider: (MOVE C FROM A TO TABLE)
    Goal: (SPACE ON C)
    Goal: (SPACE ON TABLE)
    Goal: (C ON A)
  Action: (MOVE C FROM A TO TABLE)
  Goal: (SPACE ON B)
  Goal: (A ON TABLE)
Action: (MOVE A FROM TABLE TO B)
Goal: (B ON C)
Consider: (MOVE B FROM TABLE TO C)
  Goal: (SPACE ON B)
  Consider: (MOVE A FROM B TO TABLE)
    Goal: (SPACE ON A)
    Goal: (SPACE ON TABLE)
    Goal: (A ON B)
  Action: (MOVE A FROM B TO TABLE)
  Goal: (SPACE ON C)
  Goal: (B ON TABLE)
Action: (MOVE B FROM TABLE TO C)
Must reachieve clobbered goals: ((A ON B))
Goal: (A ON B)
Consider: (MOVE A FROM TABLE TO B)
  Goal: (SPACE ON A)
  Goal: (SPACE ON B)
  Goal: (A ON TABLE)
Action: (MOVE A FROM TABLE TO B)
((START) (EXECUTING (MOVE C FROM A TO TABLE))
 (EXECUTING (MOVE A FROM TABLE TO B)) (EXECUTING (MOVE  A
FROM B TO TABLE)) (EXECUTING (MOVE B FROM TABLE TO C))
(EXECUTING (MOVE A FROM TABLE TO B)))
```


Larger Problems

How long do four block problems take?

```
;; Stack four clear blocks (good goal ordering)
> (time (gps '((a on table)(b on table) (c on table) (d on table)(space on a)
                (space on b) (space on c) (space on d)(space on table))
    '((c on d)(b on c)(a on b))))
User Run Time     = 0.00 seconds
((START)
(EXECUTING (MOVE C FROM TABLE TO D))
 (EXECUTING (MOVE B FROM TABLE TO C))
 (EXECUTING (MOVE A FROM TABLE TO B)))
```

```
;; Stack four clear blocks (bad goal ordering)
> (time (gps '((a on table)(b on table) (c on table) (d on table)(space on a)
         (space on b) (space on c) (space on d)(space on table))
    '((a on b)(b on c) (c on d))))
User Run Time     = 0.06 seconds
((START)
 (EXECUTING (MOVE A FROM TABLE TO B))
 (EXECUTING (MOVE A FROM B TO TABLE))
 (EXECUTING (MOVE B FROM TABLE TO C))
 (EXECUTING (MOVE B FROM C TO TABLE))
 (EXECUTING (MOVE C FROM TABLE TO D))
 (EXECUTING (MOVE A FROM TABLE TO B))
 (EXECUTING (MOVE A FROM B TO TABLE))
 (EXECUTING (MOVE B FROM TABLE TO C))
 (EXECUTING (MOVE A FROM TABLE TO B)))
```


A Problem STRIPS Cannot Solve

- Due to the “hack” used to solve clobbered goals, STRIPS cannot even solve certain problems.
- Consider the problem of switching the contents of two program variables.

Operator: Assign(X,Y)
    Preconditions: Value(X,A), Value(Y,B)
    Delete List: Value(X,A)
    Add List: Value(X,B)

Initial state: Value(M,1), Value(N,0), Value(L,2)
Goal state: Value(M,0), Value(N,1)

STRIPS will first do Assign(M,N) to achieve Value(M,0); however, then it is unable to achieve Value(N,1) since the 1 value has already been lost.

Of course the other goal ordering has an analagous problem.

Need the “white knight” action Assign(L,M) first to save the 1 value for later use by Assign(N,L).

Partial-Order Planners

- Don’t commit to ordering of actions until necessary by allowing partially ordered plans.