Logic and Agents

Situation Calculus

intuition: represent planning problem with FOL

- lets us reason about changes in the world
- use theorem proving
  - to “prove” that a particular sequence of actions,
  - when applied to the initial situation
  - leads to desired result

The Idea

goal: draw conclusions from a set of data (observations, beliefs, etc)

logic is

- a powerful and well developed approach
- also a strong formal system suited for algorithms

challenges

- formalizing all real world facts (especially on a true/false basis)
- computational complexity

Planning
(with Logic)

Planning

- find a **sequence of actions**
- that achieves a given **goal**
- when executed from a given **initial world state**

- i.e., given
  - a set of *operator descriptions* defining the possible (primitive) actions by the agent
  - an *initial state* description, and a *goal state* description
- compute a plan
  - which is a sequence of operator instances
  - which after executing them in the initial state
  - changes the world to a goal state

Typical Assumptions

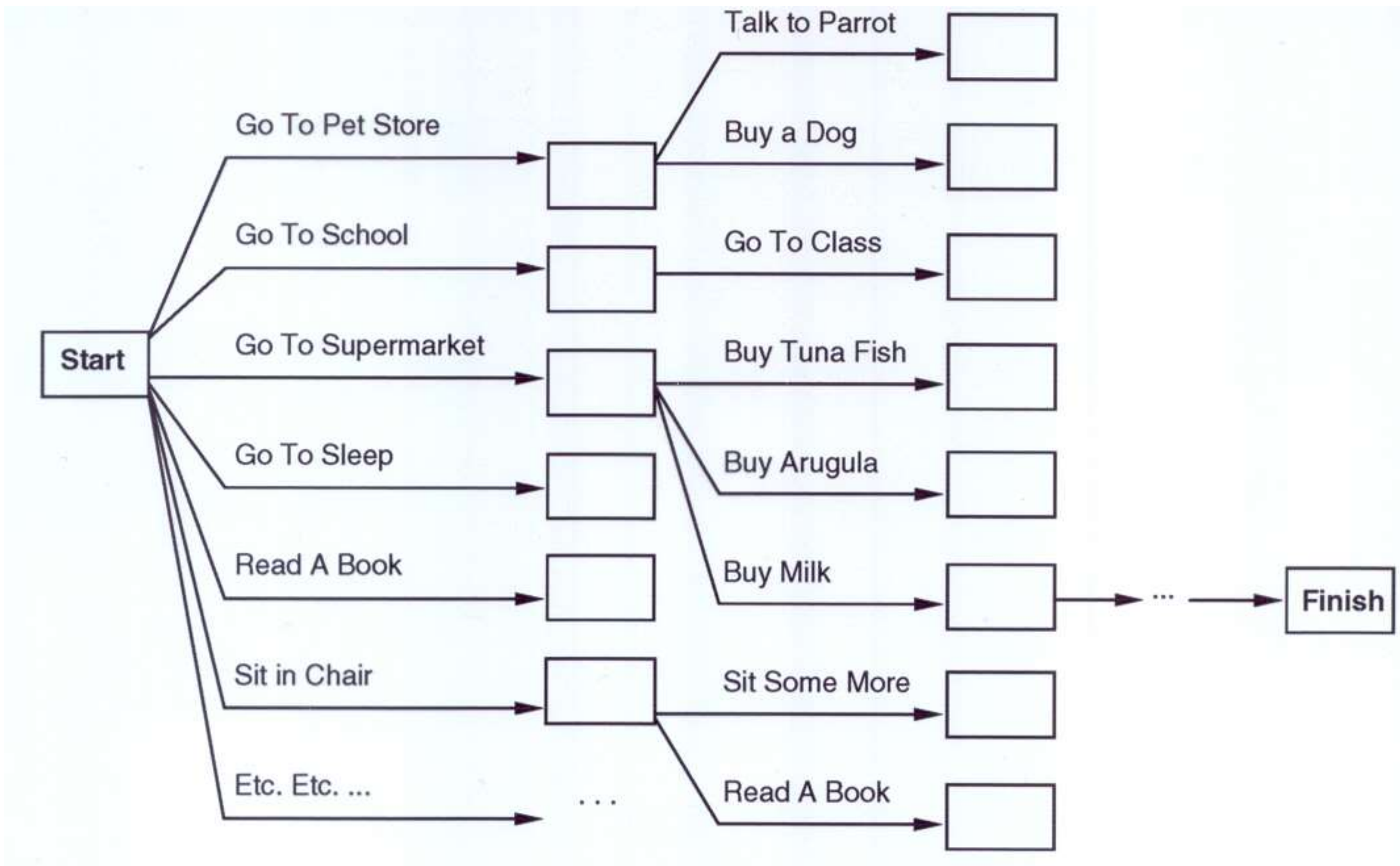
- atomic time: each action is indivisible
- no concurrent actions (but actions need not be ordered w.r.t each other in the plan)
- deterministic actions: action results completely determined (no uncertainty in their effects)
- agent is the sole cause of change in the world
- agent is omniscient with 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
  - and anything not listed is false

Planning as Search

planning, e.g., as just another search problem

- **actions:** generate successor states
- **states:** completely described & only used for successor generation, heuristic fn. evaluation & goal testing
- **goals:** represented as a goal test (and using a heuristic fct)
- **plan representation:** sequence of actions forward from initial states (or backward from goal state)

"Get a quart of milk, a bunch of bananas and a variable-speed cordless drill."



treating planning as a (generic) search problem typically gets computationally hard...

General Problem Solver (GPS)

- early planner (Newell, Shaw, and Simon, 1957)
  - mainly of historic interest
  - generates **actions** that **reduce** the **difference** between some **state** and a **goal state** (using search)
- ***Means-Ends Analysis***
  - **compare** given to desired states
  - **select** a best action that should be done next
  - **table of differences** to identify procedures to reduce types of differences
- is a **state space planner**
  - operates in the domain of state space
  - problems specified by an initial state, some goal states, and a set of operations

Situation Calculus

Initial state

$At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \neg\ Have(Bananas, S_0) \wedge \neg\ Have(Drill, S_0)$

Goal state

$(\exists s)\ At(Home,s) \wedge Have(Milk,s) \wedge Have(Bananas,s) \wedge Have(Drill,s)$

Operators

descriptions of how world changes as a result of actions

$\forall(a,s)\ Have(Milk,Result(a,s)) \Leftrightarrow$

$((a=Buy(Milk) \wedge At(Grocery,s)) \vee (Have(Milk, s) \wedge a \neq Drop(Milk)))$

Result(a,s)

names situation resulting from executing action a in situation s

Action sequences

Result*(l,s) is result of executing the list of actions (l)

$(\forall s)\ Result^*([],s) = s$

$(\forall a,p,s)\ Result^*([a|p]s) = Result^*(p,Result(a,s))$

Situation Calculus

solution:

- a plan ***p*** as list of actions
- that yields situation satisfying the goal

$$\begin{aligned}&At(Home, Result^*(p,S_0))\\&\quad\wedge Have(Milk, Result^*(p,S_0))\\&\quad\wedge Have(Bananas, Result^*(p,S_0))\\&\quad\wedge Have(Drill, Result^*(p,S_0))\end{aligned}$$

e.g.,

p = [Go(Grocery), Buy(Milk), Buy(Bananas),
Go(HardwareStore), Buy(Drill), Go(Home)]

(Logical) Planning with General Inference

situation calculus

- fine, but can be exponential in the worst case
- resolution theorem finds *a* proof (plan), not necessarily a good plan

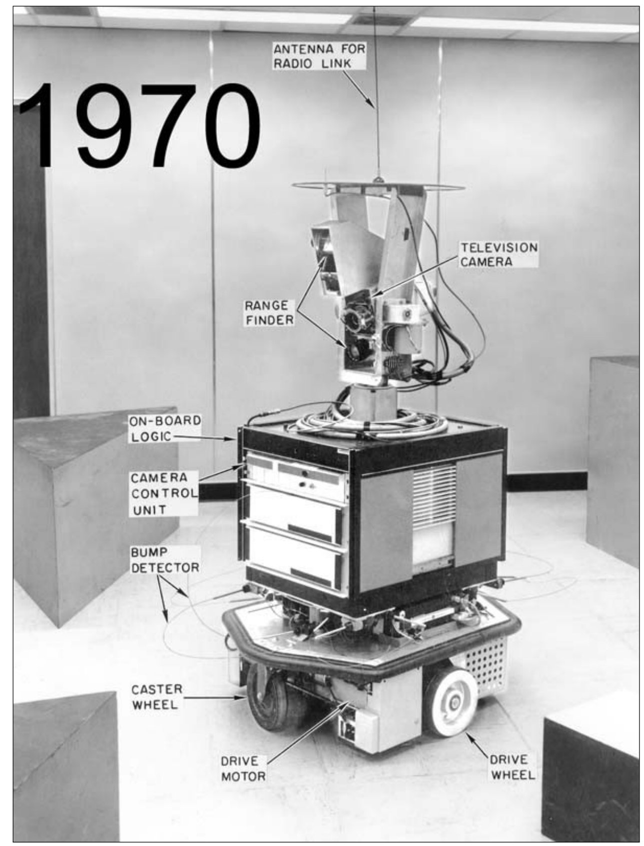
hence typically for planning

- restriction on the language
- and use of special-purpose algorithms (i.e., specialized planners)
- it is a quite large research area (here only glimpse into the topic)

STRIPS Planning

(Stanford Research Institute Problem Solver)

- classic approach
- used for Shakey the robot



STRIPS Planning

- state
  - conjunction of ground literals
  - e.g., at(Home) $\wedge$ $\neg$have(Milk) $\wedge$ $\neg$have(bananas) ...
- goals
  - conjunctions of literals
  - may have variables (existentially quantified)
  - e.g., at(X) $\wedge$ have(Milk) $\wedge$ have(bananas) ...
- no need to fully specify state
  - non-specified conditions: don't-care or assumed false
  - often represent changes rather than entire situation

Operator/Action Representation

operators contain three components

- **Action** description
- **Precondition**
  - conjunction of positive literals
- **Effect**
  - conjunction of positive or negative literals
  - describe how the situation changes

e.g.: Op[Action: Go(there),
Precond: $At(here) \wedge Path(here,there)$,
Effect: $At(there) \wedge \neg At(here)$]

Operator/Action Representation

- 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

Example Blocks World

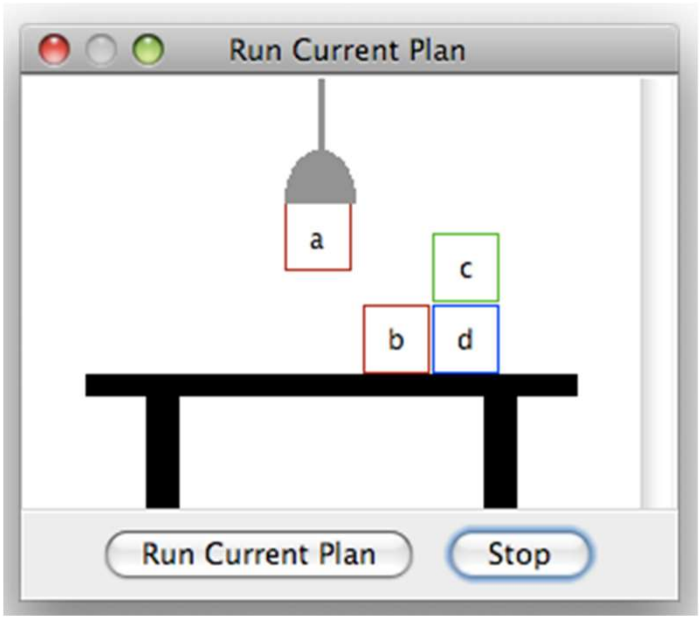
a table, a set of blocks and a robot hand

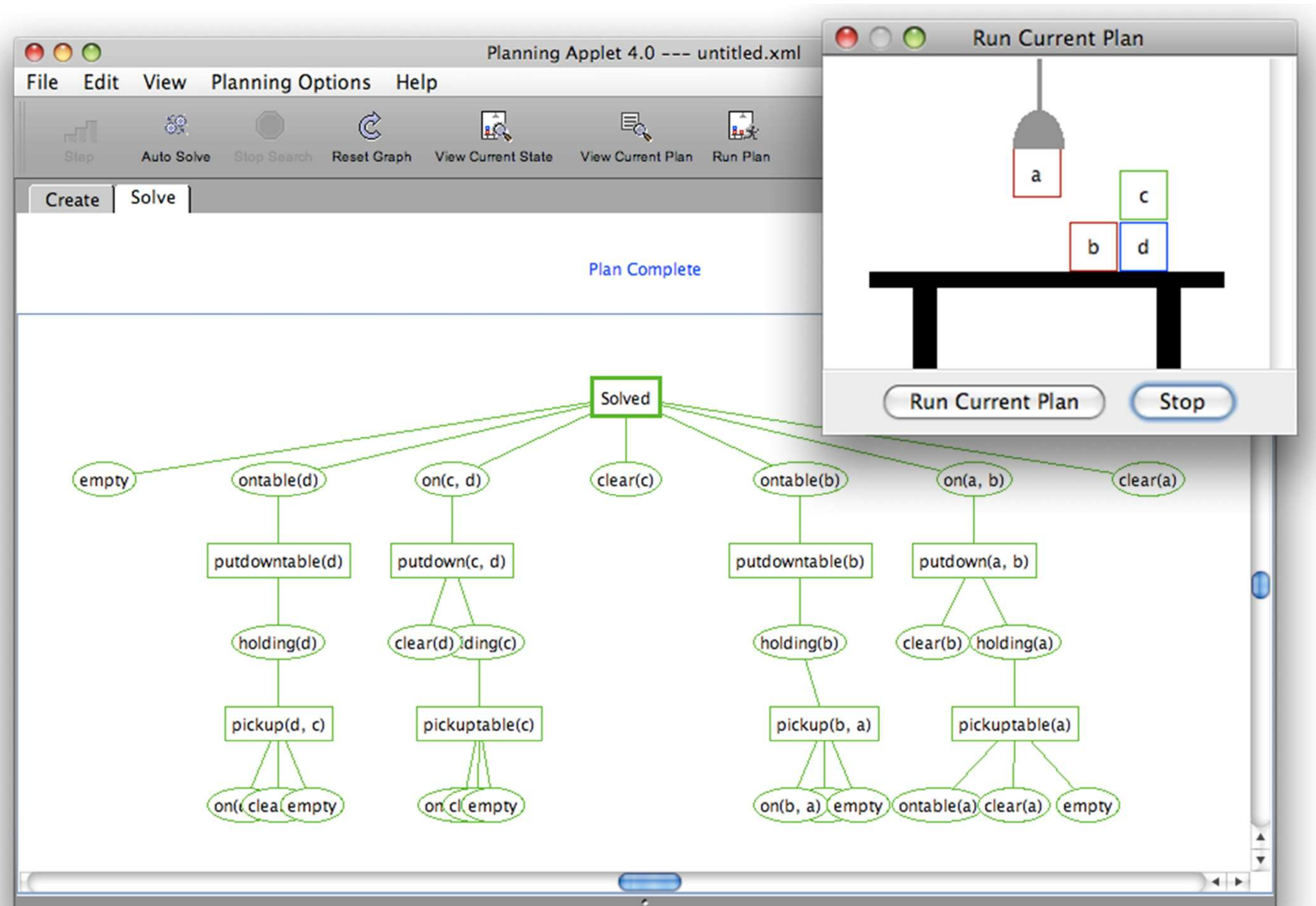
some domain constraints:

- 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(b)	ontable(d)
on(c,d)	holding(a)
clear(b)	clear(c)





demo at http://aispace.org/planning/

Blocks World Operators

- classic basic operations
  - **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 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**

Blocks World Operators

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]).

STRIPS planning

- two additional data structures:
  - **State List**: all currently true predicates
  - **Goal Stack**: a push down stack of goals to be solved, with current goal on top of stack.
- current goal is not satisfied by present state
  - examine add lists of operators
  - push suited operator and its preconditions list on stack (subgoals)
- a current goal is satisfied: POP it from stack
- an operator is on top of the stack
  - check preconditions: if one is unfulfilled, re-introduce it on stack
  - record the application of that operator on the plan
  - use the operator’s add and delete lists to update the current state

can be implemented with Prolog

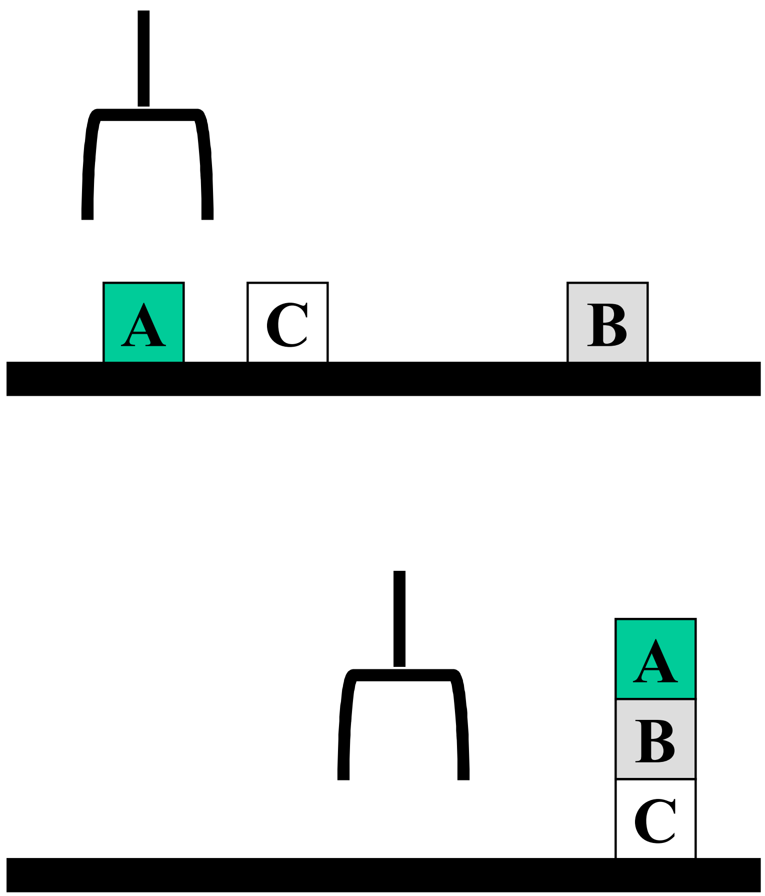
Example

Initial state:

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

Goal:

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



Plan:

- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

STRIPS planning

State List (SL): all currently true predicates

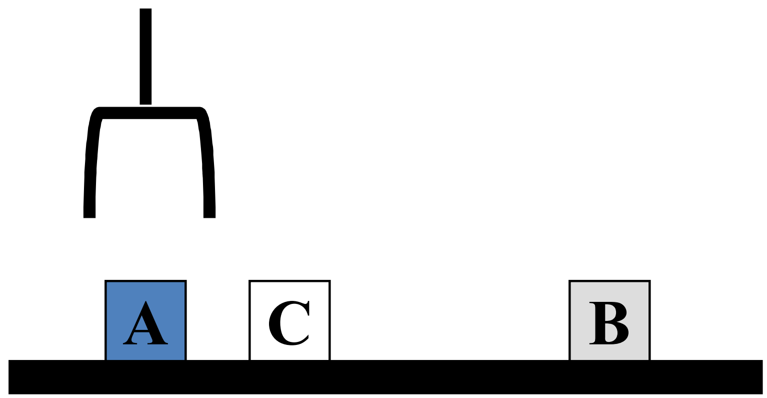
Goal Stack (GS): a push down stack of goals to be solved

- current goal is not satisfied by present state
  - examine add lists of operators
  - push suited operator and its preconditions list on stack (subgoals)
- a current goal is satisfied: POP it from stack
- an operator is on top of the stack
  - check preconditions: if one is unfulfilled, re-introduce it on stack
  - record the application of that operator on the plan
  - use the operator's add and delete lists to update the current state

STRIPS: very simple Blocks World example

Initial state:

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

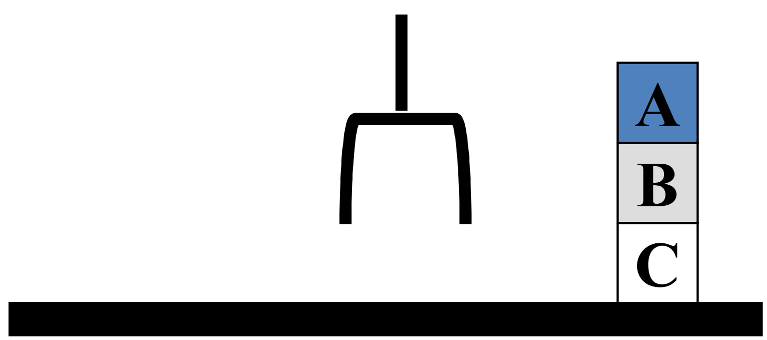


Plan:

- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Goal:

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



Blocks World Operators

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]).

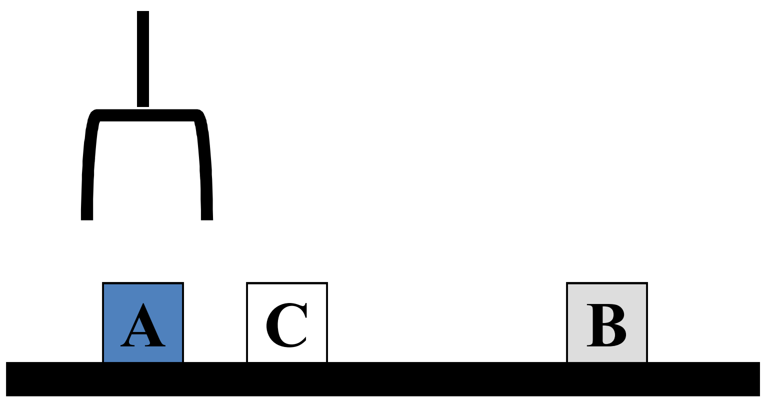
operator(putdown(X),
    [holding(X)],
    [ontable(X), handempty, clear(X)],
    [holding(X)],
    [X≠table]).

STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

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

Plan:



STRIPS: very simple Blocks World example

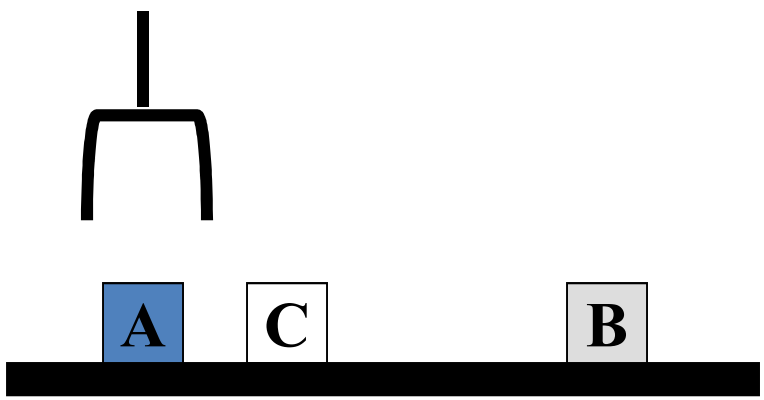
SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

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

<- try to fulfill first unfulfilled subgoal first

Plan:

green = subgoal is fulfilled
read = not fulfilled



STRIPS planning

State List (SL): all currently true predicates

Goal Stack (GS): a push down stack of goals to be solved

- **current goal is not satisfied by present state**
  - **examine add lists of operators**
  - **push suited operator and its preconditions list on stack (subgoals)**
- a current goal is satisfied: POP it from stack
- an operator is on top of the stack
  - check preconditions: if one is unfulfilled, re-introduce it on stack
  - record the application of that operator on the plan
  - use the operator’s add and delete lists to update the current state

STRIPS: very simple Blocks World example

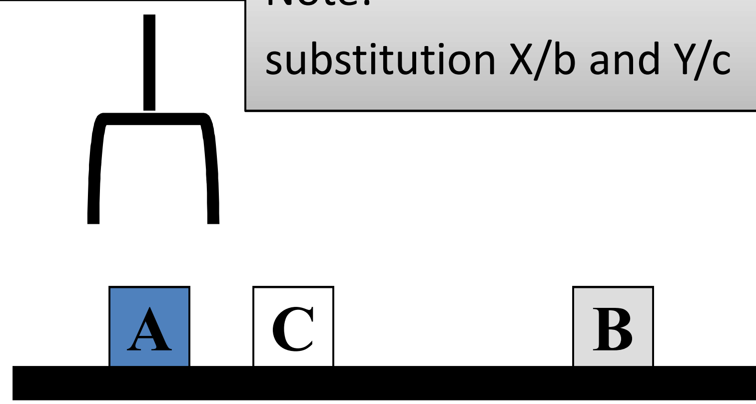
SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

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

Plan:

operator(stack(X,Y),
    [holding(X), clear(Y)],
    [handempty, **on(X,Y)**, clear(X)],
    [holding(X), clear(Y)],
    [X≠Y, Y≠table, X≠table]).

Note:
substitution X/b and Y/c



STRIPS: very simple Blocks World example

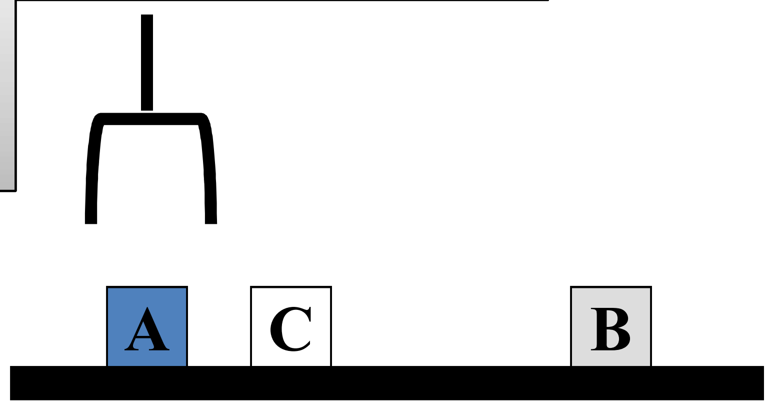
SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

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

Plan:

operator(**stack(b,c)**,
[**holding(b)**, **clear(c)**],
[handempty, **on(b,c)**, clear(b)],
[holding(b), clear(c)],
[$b \neq c$, $c \neq table$, $b \neq table$]).

push action and
its pre-conditions
on the stack



STRIPS: very simple Blocks World example

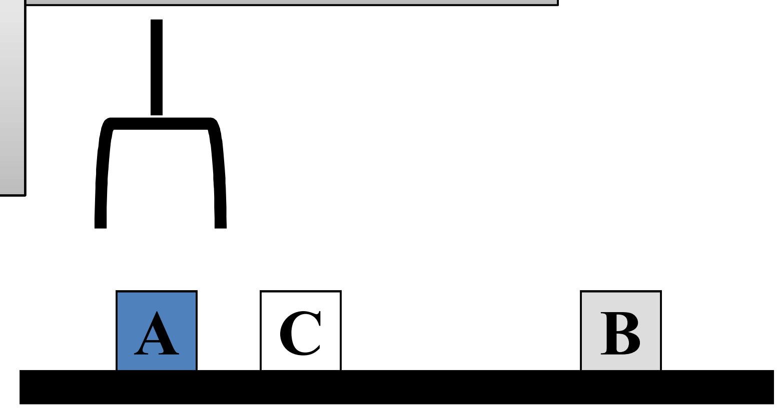
SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

Plan:

operator(**stack(b,c)**,
[**holding(b)**, **clear(c)**],
[handempty, on(b,c), clear(b)],
[holding(b), clear(c)],
[b≠c, c≠table, b≠table]).

push action and
its pre-conditions
on the stack



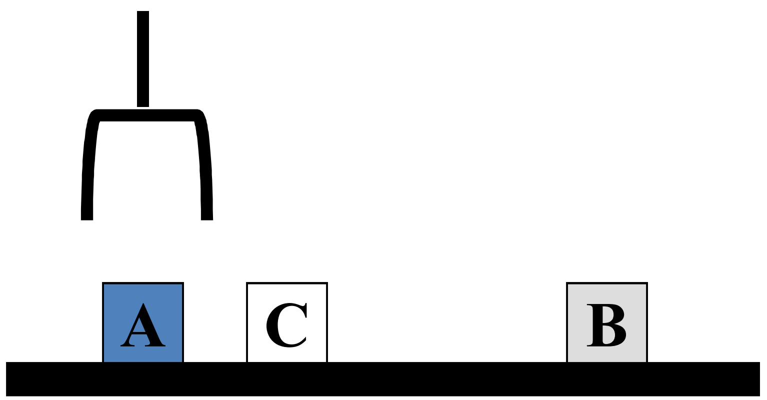
STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

Plan:

operator(pickup(X),
    [ontable(X), clear(X), handempty],
    [**holding(X)**],
    [ontable(X), clear(X), handempty],
    [X≠table]).



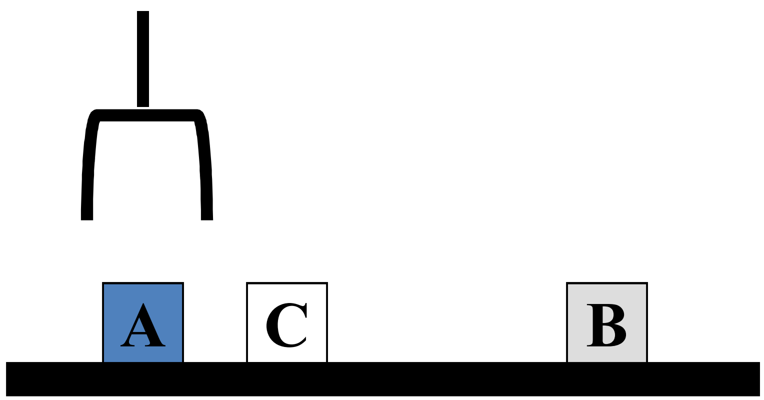
STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
ontable(b)
clear(b)
handempty
pickup(b)
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

push action and its pre-conditions on the stack

Plan:



STRIPS planning

State List (SL): all currently true predicates

Goal Stack (GS): a push down stack of goals to be solved

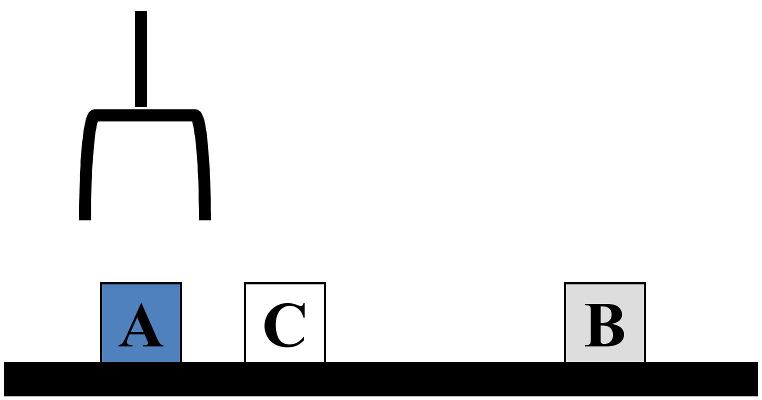
- current goal is not satisfied by present state
  - examine add lists of operators
  - push suited operator and its preconditions list on stack (subgoals)
- a current goal is satisfied: POP it from stack
- an operator is on top of the stack
  - check preconditions: if one is unfulfilled, re-introduce it on stack
  - record the application of that operator on the plan
  - use the operator’s add and delete lists to update the current state

STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
~~ontable(b)~~
clear(b)
handempty
pickup(b)
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

goal on top of stack is fulfilled
=> pop it off the stack

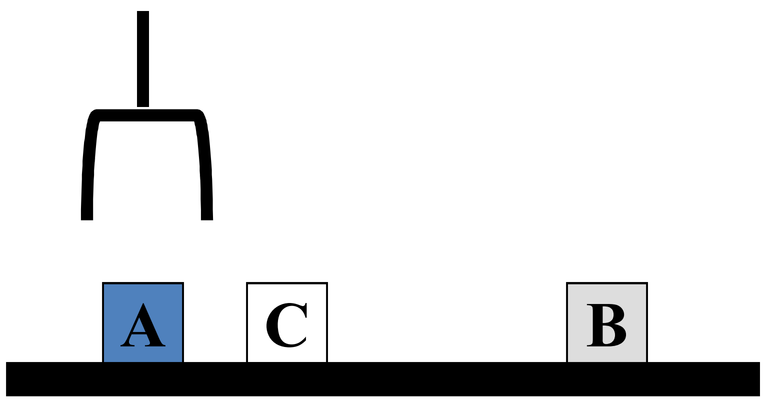


STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
~~clear(b)~~
handempty
pickup(b)
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

goal on top of stack is fulfilled
=> pop it off the stack

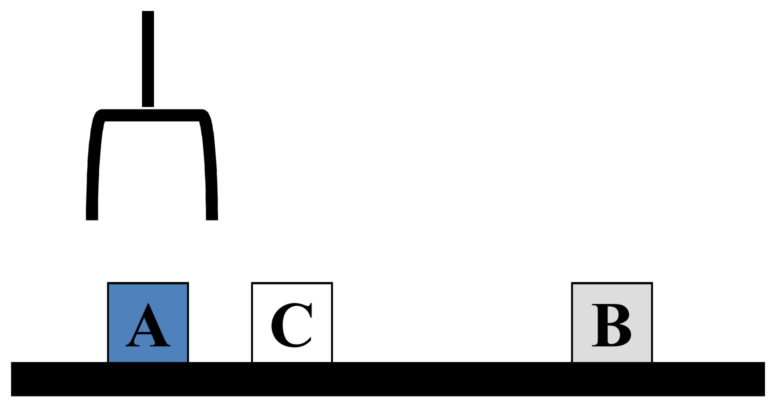


STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
~~handempty~~
pickup(b)
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

goal on top of stack is fulfilled
=> pop it off the stack



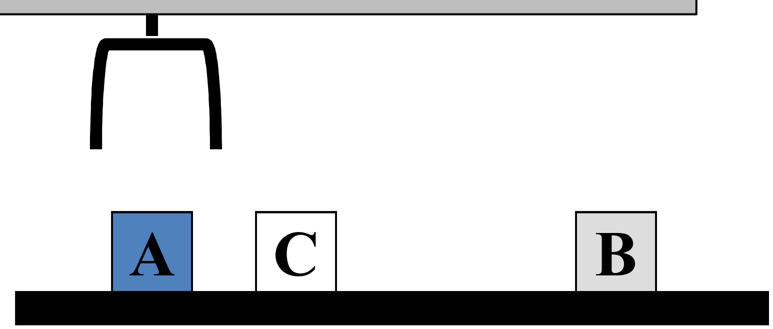
STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty
}

GS:
pickup(b)
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

check SL:
all pre-conditions
for ***pickup()***
are fulfilled!!!

operator(pickup(X),
[**ontable(X), clear(X), handempty**],
[holding(X)],
[ontable(X), clear(X), handempty],
[X≠table]).



STRIPS planning

State List (SL): all currently true predicates

Goal Stack (GS): a push down stack of goals to be solved

- current goal is not satisfied by present state
  - examine add lists of operators
  - push suited operator and its preconditions list on stack (subgoals)
- a current goal is satisfied: POP it from stack
- an operator is on top of the stack
  - check preconditions: if one is unfulfilled, re-introduce it on stack
  - record the application of that operator on the plan
  - use the operator's add and delete lists to update the current state

STRIPS: very simple Blocks World example

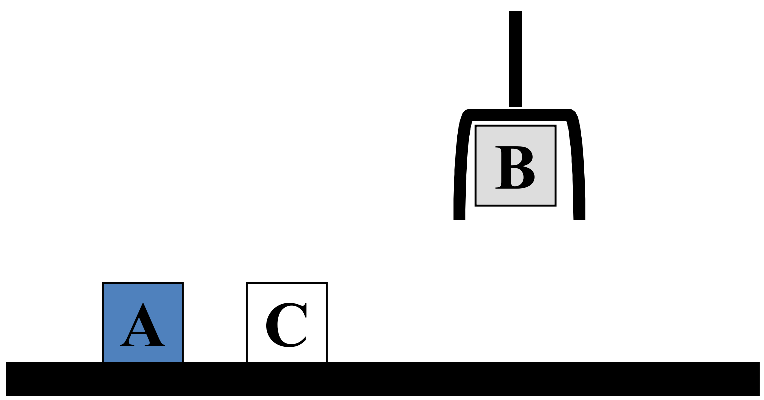
SL = {
clear(a)
~~clear(b~~)
clear(c)
ontable(a)
~~ontable(b)~~
ontable(c)
~~handempty~~
holding(b)
}

GS:
~~pickup(b)~~
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)

operator(pickup(X),
[ontable(X), clear(X), handempty],
[**holding(X)**],
[**ontable(X), clear(X), handempty**],
[X≠table]).

pickup(): change state list
(take substitutions into account)



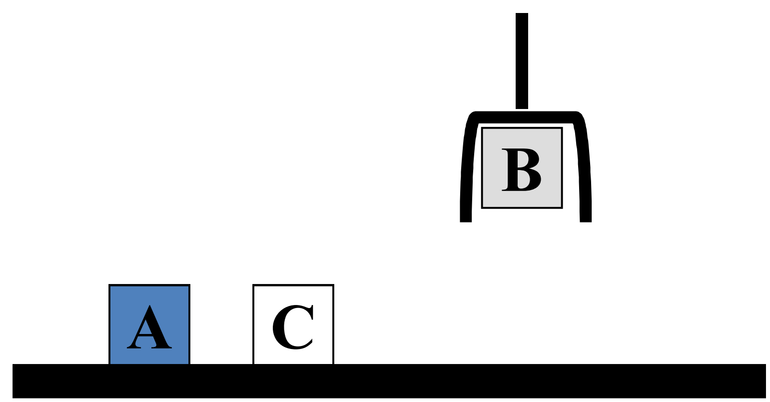
STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(c)
ontable(a)
ontable(c)
holding(b)
}

GS:
holding(b)
clear(c)
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

pop true sub-goals

Plan:
p(b)



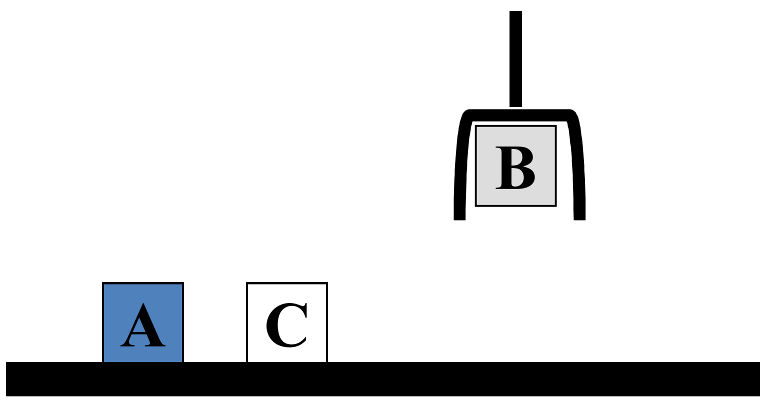
STRIPS: very simple Blocks World example

SL = {
clear(a)
clear(c)
ontable(a)
ontable(c)
holding(b)
}

GS:
~~holding(b)~~
~~clear(c)~~
stack(b,c)
on(b,c)
on(a,b)
ontable(c)

all pre-conditions for ***stack()*** fulfilled

Plan:
pickup(b)



STRIPS: very simple Blocks World example

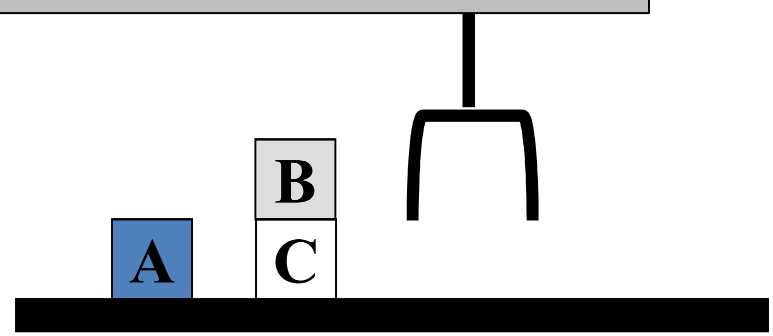
SL = {
clear(a)
~~clear(c)~~
ontable(a)
ontable(c)
~~holding(b)~~
handempty
on(b,c)
clear(b)
}

GS:
~~stack(b,c)~~
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)

operator(stack(X,Y),
[holding(X), clear(Y)],
[**handempty, on(X,Y), clear(X)**],
[**holding(X), clear(Y)**],
[X≠c, X≠table, Y≠table]).

execute ***stack()***
change state list



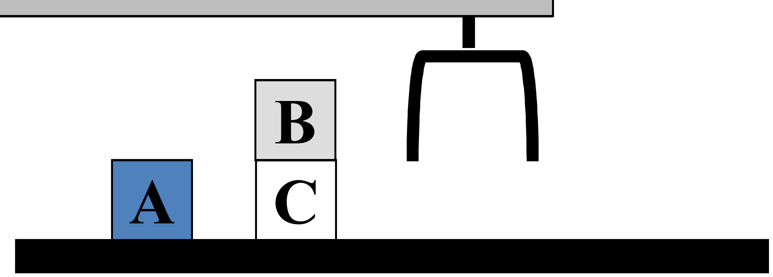
STRIPS: very simple Blocks World example

SL = {
clear(a)
ontable(a)
ontable(c)
handempty
on(b,c)
clear(b)
}

GS:
~~on(b,c)~~
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)

```
operator(stack(X,Y),
        [holding(X), clear(Y)],
        [handempty, on(X,Y), clear(X)],
        [holding(X), clear(Y)],
        [X≠Y, X≠table, Y≠table]).
```



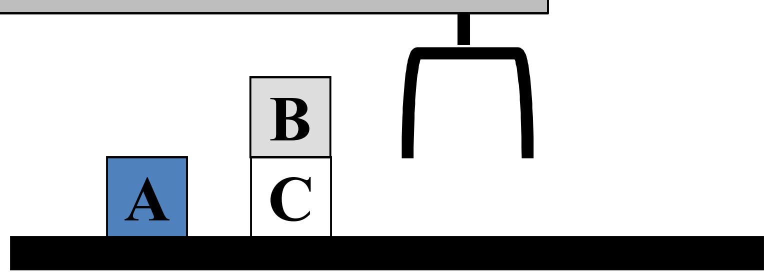
STRIPS: very simple Blocks World example

SL = {
clear(a)
ontable(a)
ontable(c)
handempty
on(b,c)
clear(b)
}

GS:
holding(a)
clear(b)
stack(a,b)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)

operator(**stack(X,Y)**,
[**holding(X), clear(Y)**],
[handempty, on(X,Y), clear(X)],
[holding(X), clear(Y)],
[X≠Y, X≠table, Y≠table]).



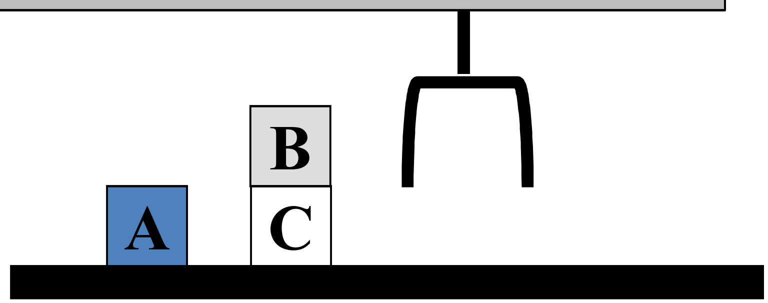
STRIPS: very simple Blocks World example

SL = {
clear(a)
ontable(a)
ontable(c)
handempty
on(b,c)
clear(b)
}

GS:
holding(a)
clear(b)
stack(a,b)
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)

operator(pickup(X),
[ontable(X), clear(X), handempty],
[**holding(X)**],
[ontable(X), clear(X), handempty],
[X≠table]).



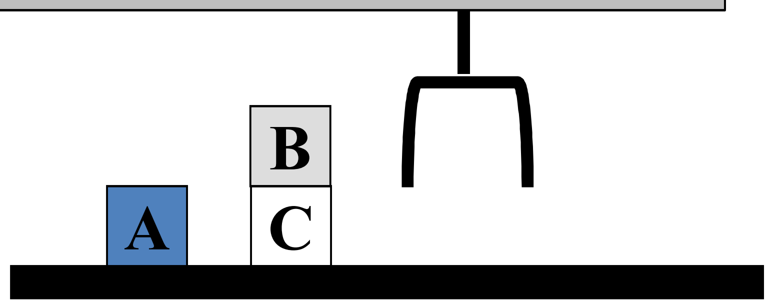
STRIPS: very simple Blocks World example

SL = {
clear(a)
ontable(a)
ontable(c)
handempty
on(b,c)
clear(b)
}

GS:
ontable(a)
clear(a)
handempty
pickup(a)
holding(a)
clear(b)
stack(a,b)
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)

operator(**pickup(X)**,
[**ontable(X), clear(X), handempty**],
[holding(X)],
[ontable(X), clear(X), handempty],
[X≠table]).



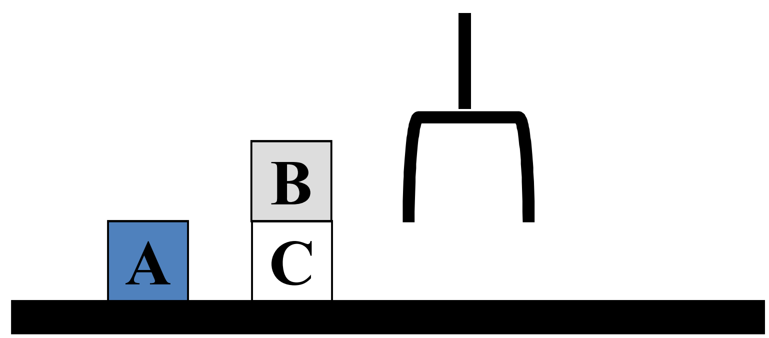
STRIPS: very simple Blocks World example

SL = {
clear(a)
ontable(a)
ontable(c)
handempty
on(b,c)
clear(b)
}

GS:
~~ontable(a)~~
~~clear(a)~~
~~handempty~~
pickup(a)
holding(a)
clear(b)
stack(a,b)
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)

all pre-conditions for ***pickup()*** fulfilled



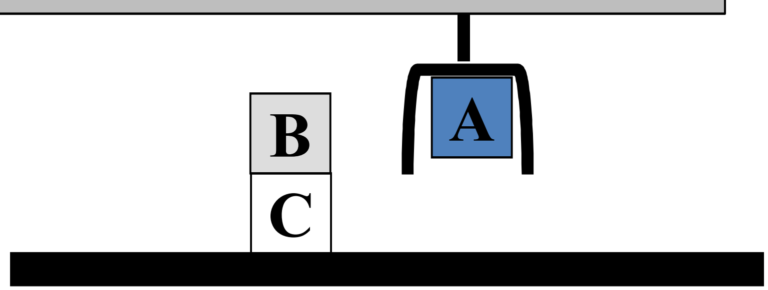
STRIPS: very simple Blocks World example

SL = {
~~clear(a)~~
~~ontable(a)~~
ontable(c)
~~handempty~~
on(b,c)
clear(b)
holding(a)
}

GS:
~~pickup(a)~~
holding(a)
clear(b)
stack(a,b)
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)
pickup(a)

operator(pickup(X),
[ontable(X), clear(X), handempty],
[**holding(X)**],
[**ontable(X), clear(X), handempty**],
[X≠table]).

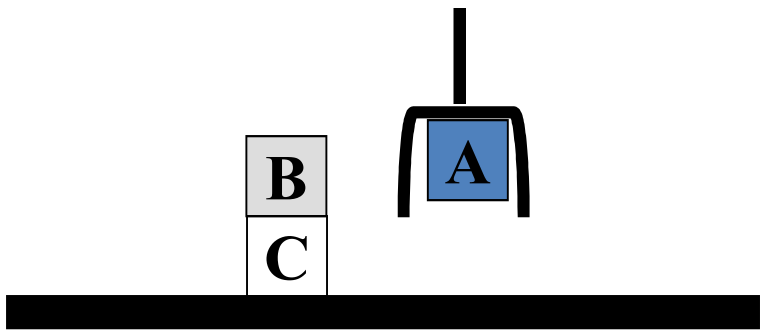


STRIPS: very simple Blocks World example

SL = {
ontable(c)
on(b,c)
clear(b)
holding(a)
}

GS:
~~holding(a)~~
~~clear(b)~~
stack(a,b)
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)
pickup(a)



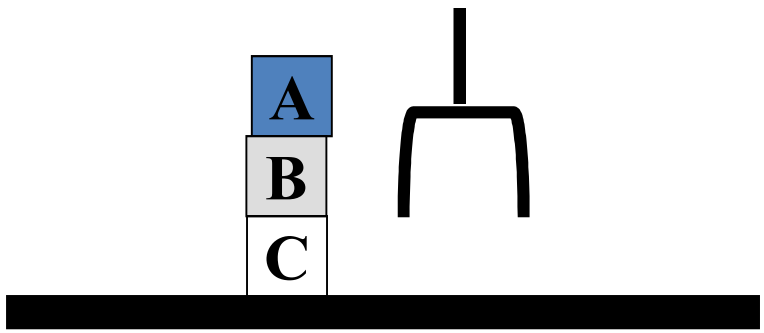
STRIPS: very simple Blocks World example

SL = {
ontable(c)
on(b,c)
~~clear(b)~~
~~holding(a)~~
handempty
on(a,b)
clear(a)
}

GS:
~~stack(a,b)~~
on(b,c)
on(a,b)
ontable(c)

Plan:
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

operator(stack(X,Y),
[holding(X), clear(Y)],
[**handempty, on(X,Y), clear(X)**],
[**holding(X), clear(Y)**],
[X≠c, X≠table, Y≠table]).



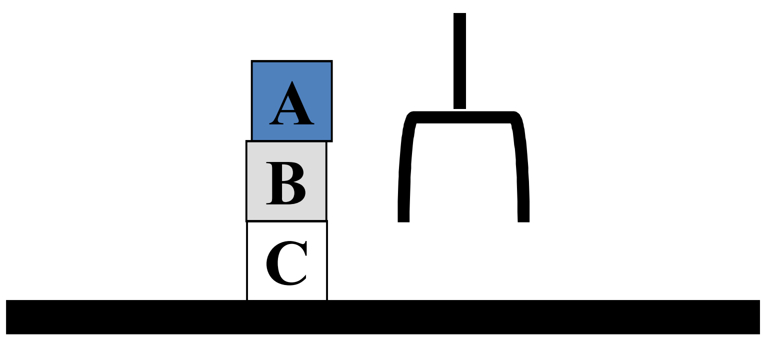
STRIPS: very simple Blocks World example

SL = {
ontable(c)
on(b,c)
handempty
on(a,b)
clear(a)
}

GS:
~~on(b,c)~~
~~on(a,b)~~
~~ontable(c)~~

all (sub-)goals fulfilled!!!
=>
done

Plan:
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)



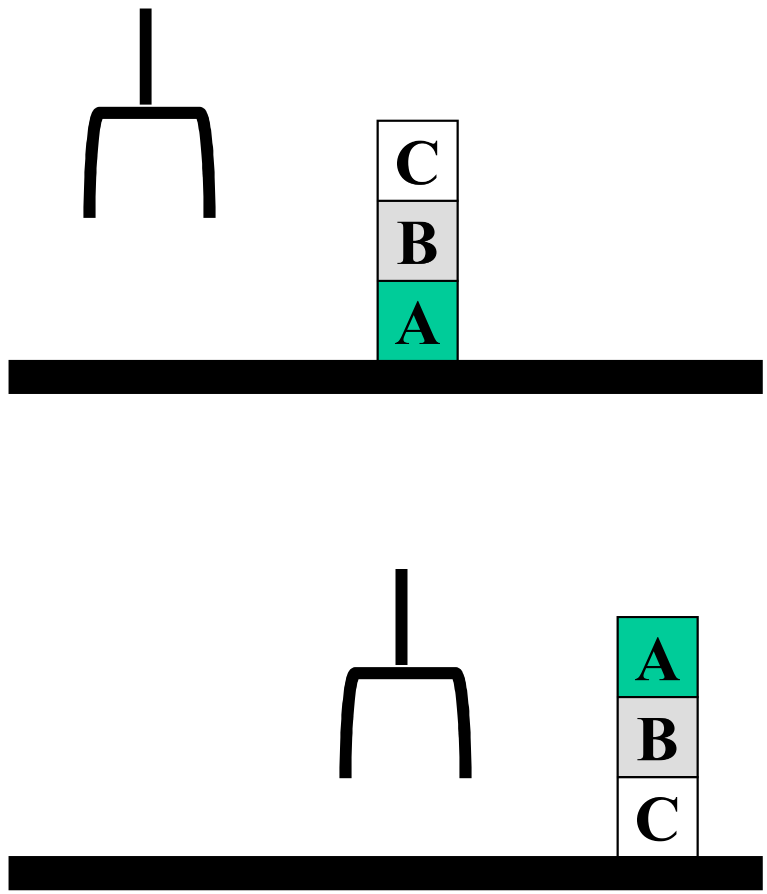
More Complex Example (just the result)

Initial state:

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

Goal:

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

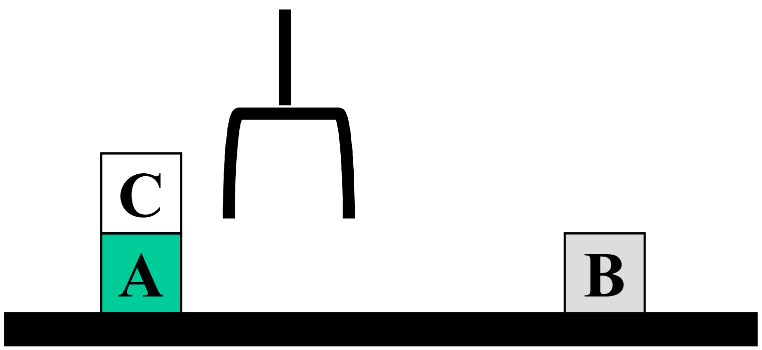


Plan:

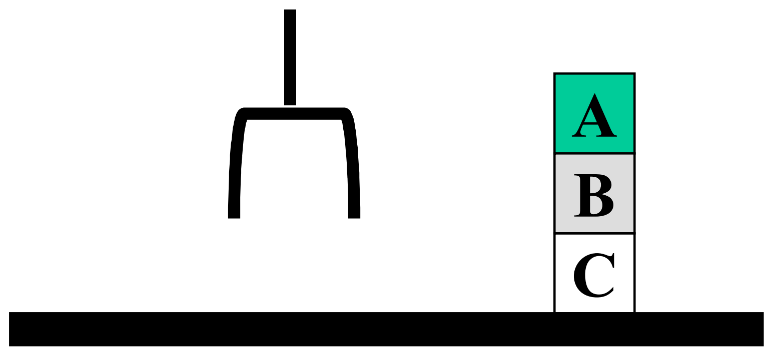
- unstack(c,b)
- putdown(c)
- unstack(b,a)
- putdown(b)
- pickup(b)
- stack(b,a)
- unstack(b,a)
- putdown(b)
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Limitations: e.g., Goal Interaction

- simple planning assumes independent sub-goals
  - solve each separately and concatenate the solutions
- "Sussman Anomaly" (classic example)
  - solving on(A,B) first (via unstack(C,A), stack(A,B))
  - is undone when solving 2nd goal on(B,C) (via unstack(A,B), stack(B,C))
  - solving on(B,C) first will be undone when solving on(A,B)



initial state



goal state

Limitations: e.g., Goal Interaction

- classic STRIPS could not handle “Sussman Anomaly”
  - hacks in STRIPS to treat simple cases
- in general: design of efficient, yet general planners not easy
  - choosing the right planner (and fully understanding its capabilities & limitations) is not trivial
  - wide-spread use of (logical) planning in real applications still missing
- and there are also general challenges on the knowledge representation side...

initial state

goal state

General Challenges of Knowledge Representation

Representing change: The frame problem

Frame axioms

- if property x does not change
- as a result of applying action a in state s
- then it stays the same

e.g.,

$On(x, z, s) \wedge Clear(x, s) \rightarrow$
$On(x, table, Result(Move(x, table), s)) \wedge \neg On(x, z, Result(Move(x, table), s))$

$On(y, z, s) \wedge y \neq x \rightarrow On(y, z, Result(Move(x, table), s))$

The proliferation of frame axioms
becomes very cumbersome in complex domains

The frame problem

- **successor-state axiom**: general statement to characterize every way in which a predicate can become true
  - either it can be **made true**
  - or it can **already be true and not be changed**
  - e.g, On (x, table, Result(a,s)) $\leftrightarrow$
    [On (x, z, s) $\wedge$ Clear (x, s) $\wedge$ a = Move(x, table)] $\wedge$
    [On (x, table, s) $\wedge$ a $\neq$ Move (x, z)]
- in complex worlds with reasoning about longer chains of action, even these types of axioms are too cumbersome
  - planning systems use special-purpose inference methods
  - to reason about the expected state of the world at any point in time during a multi-step plan

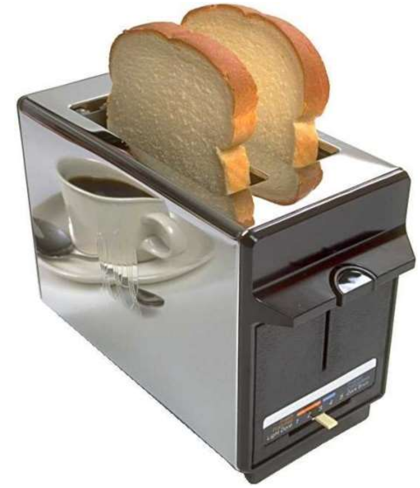
Qualification problem

How can you possibly characterize
every single effect of an action,
or every single exception that might occur?

e.g., when I put my bread into the toaster, and push the button, it will become toasted after two minutes, unless…

- the toaster is broken, or…
- the power is out, or…
- I blow a fuse, or…
- a neutron bomb explodes nearby and fries all electrical components, or…
- a meteor strikes the earth, and the world we know it ceases to exist, or…

Ramification problem



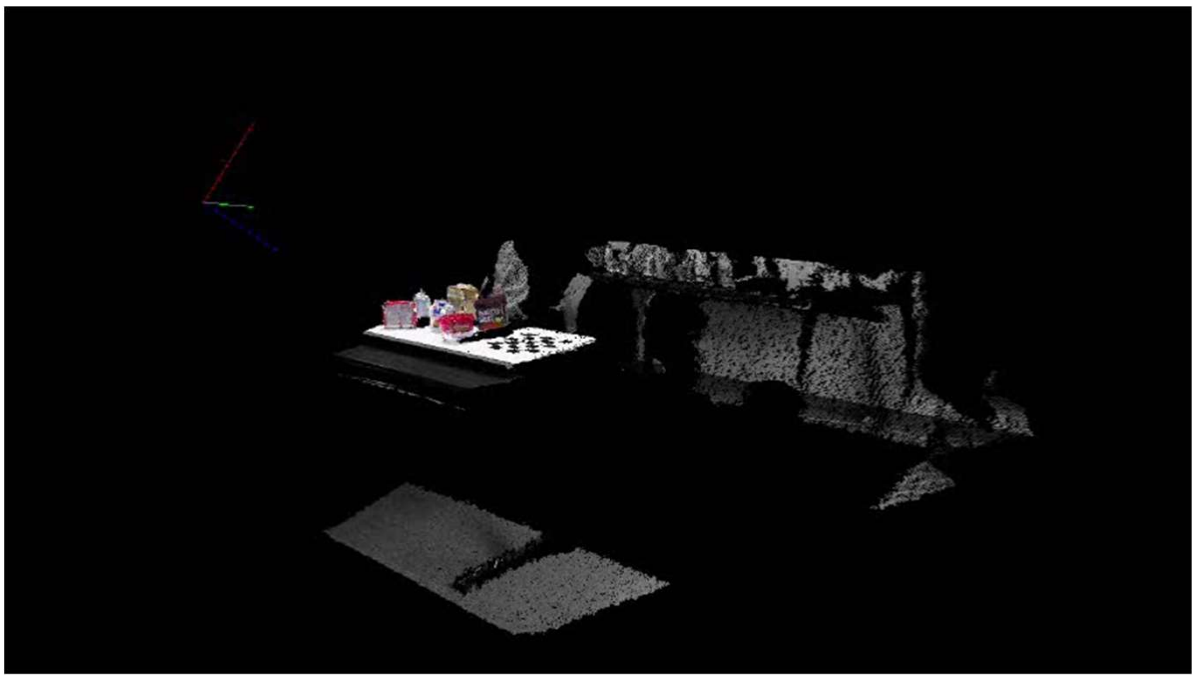
Similarly, it is just about impossible to characterize every side effect of every action at every possible level of detail

When I put my bread into the toaster, and push the button, the bread will become toasted after two minutes, and…

- The crumbs that fall off the bread onto the bottom of the toaster over tray will also become toasted, and…
- Some of the aforementioned crumbs will become burnt, and…
- The outside molecules of the bread will become “toasted,” and…
- The inside molecules of the bread will remain more “breadlike,” and…
- The toasting process will release a small amount of humidity into the air because of evaporation, and…
- The heating elements will become a tiny fraction more likely to burn out the next time I use the toaster, and…
- The electricity meter in the house will move up slightly, and…

Symbol Grounding

- from “raw” data to symbolic placeholders (and back)
- sensor data -> ‘bottle’, ‘green(X)’, ‘step-forward’ (of an other agent), ...
- ‘step-forward’ (own action), ... -> motor data



Symbol Grounding

- “only” an engineering challenge?!?!
- or the actual “hard” part (especially perception)?!?!
- or even fundamentally unsolvable?!?!

Searle’s Chinese Room

popular argument against “hard” AI

- operator O. in a room
- Chinese symbols come in which O. does not understand
- he has explicit instructions (a program)
  - how to generate output from input via pattern-matching and rules
  - allows to generate “answers” from “questions”

He understands nothing even though Chinese speakers who see the output find it correct and indistinguishable from a “real” “cognitive” agent

Knowledge Engineering

- hard to model the “right” conditions and the “right” effects at the “right” level of abstraction
- entire field (like Software Engineering) to investigate procedures and standards

- hope for automated knowledge acquisition
- e.g., use WWW (Wikipedia, cooking recipe sites, Youtube, etc.) to extract formalized knowledge

Knowledge Engineering & use of WWW

THE END… ☺