



Planning

Some material adapted from slides by Tim Finin, Jean-Claude Latombe, Lise Getoor, and Marie desJardins



This Week's Classes

- What is planning?
- Approaches to planning
 - GPS / STRIPS
 - Situation calculus formalism
 - Partial-order planning
 - Graph-based planning
 - Satisfiability planning



Planning problems

- Autonomous vehicle navigation
- Travel planning
- Military operation planning and scheduling
- Emergency response planning
- Program composition/synthesis
- Diagnosis and decision support
- Process control
- Information gathering



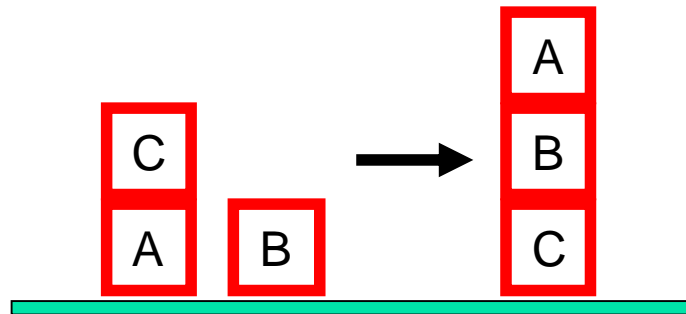
Before Jumping in....

- How do we plan?
- What are the problems?
- What are the objectives?
- When is a plan done?
- When is a plan redone?
- What about collaborative planning?

Classical planning

Problem definition: Generate one possible way to achieve a certain **goal** given an **initial situation** and a set of **operators** or **actions**.

Example: Blocks world problems



Simplifications: no-uncertainty, complete state information, open-loop control



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



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



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.



Formulating planning problems

- What is the state space?
- What are the operators?
- What is the branching factor?
- How do we separate the parts of the overall goal?
- How can we solve planning problems using search?



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)`

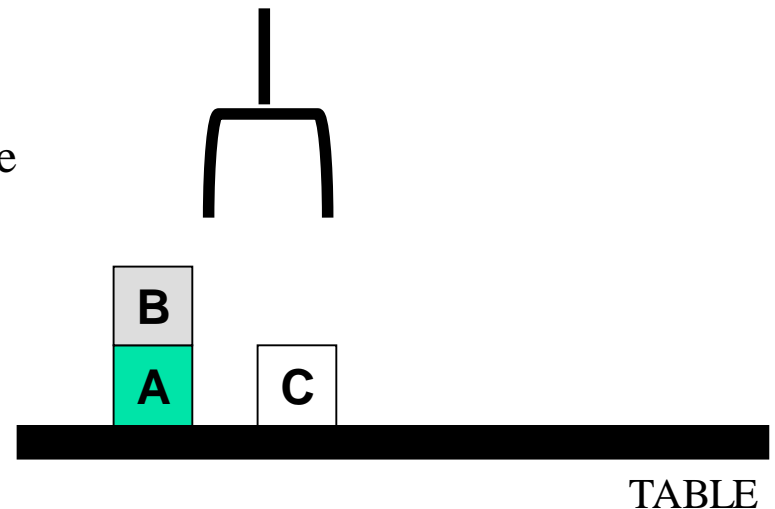
`ontable(c)`

`on(b,a)`

`handempty`

`clear(b)`

`clear(c)`





Major approaches

- GPS / STRIPS
- Situation calculus
- Partial-order planning
- Planning with constraints (SATplan, Graphplan)
- Hierarchical decomposition (HTN planning)
- Reactive planning



General Problem Solver

- The General Problem Solver (GPS) system was an early planner (Newell, Shaw, and Simon)
- GPS generated actions that reduced the difference between some state and a goal state
- GPS used Means-Ends Analysis
 - Compare what is given or known with what is desired and select a reasonable thing to do next
 - Use a table of differences to identify procedures to reduce types of differences
- GPS was a state space planner: it operated in the domain of state space problems specified by an initial state, some goal states, and a set of operations



Situation representation

Using logic to describe a situation:

- A set of propositions.
- A propositional logic KB.
- A set of ground clauses in FOL.
- Use variables in state representation.
- Use a FOL KB to describe a state
(explicit vs. implicit representations).



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



Situation calculus

- **Initial state:** a logical sentence about (situation) S_0

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

- **Goal state:**

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

- **Operators** are descriptions of how the world changes as a result of the agent's actions:

$$\begin{aligned} \forall (a, s) \text{Have}(\text{Milk}, \text{Result}(a, s)) \Leftrightarrow \\ ((a = \text{Buy}(\text{Milk}) \wedge \text{At}(\text{Grocery}, s)) \vee (\text{Have}(\text{Milk}, s) \wedge a \neq \text{Drop}(\text{Milk}))) \end{aligned}$$

- $\text{Result}(a, s)$ names the situation resulting from executing action a in situation s .
- Action sequences are also useful: $\text{Result}'(l, s)$ is the result of executing the list of actions (l) starting in s :

$$(\forall s) \text{Result}'([], s) = s$$

$$(\forall a, p, s) \text{Result}'([a|p]s) = \text{Result}'(p, \text{Result}(a, s))$$



Situation calculus II

- A solution is a plan that when applied to the initial state yields a situation satisfying the goal query:

$$\begin{aligned} & \text{At(Home, Result'(p, S}_0\text{))} \\ & \quad \wedge \text{Have(Milk, Result'(p, S}_0\text{))} \\ & \quad \wedge \text{Have(Bananas, Result'(p, S}_0\text{))} \\ & \quad \wedge \text{Have(Drill, Result'(p, S}_0\text{))} \end{aligned}$$

- Thus we would expect a plan (i.e., variable assignment through unification) such as:

$$\begin{aligned} p = & [\text{Go(Grocery), Buy(Milk), Buy(Bananas),} \\ & \text{Go(HardwareStore),} \\ & \text{Buy(Drill), Go(Home)}] \end{aligned}$$

Situation calculus: Blocks world

- Here's an example of a situation calculus rule for the blocks world:
 - $\text{Clear}(X, \text{Result}(A, S)) \leftrightarrow$
 $[\text{Clear}(X, S) \wedge$
 $(\neg(A = \text{Stack}(Y, X) \vee A = \text{Pickup}(X))$
 $\vee (A = \text{Stack}(Y, X) \wedge \neg(\text{holding}(Y, S)))$
 $\vee (A = \text{Pickup}(X) \wedge \neg(\text{handempty}(S) \wedge \text{ontable}(X, S) \wedge \text{clear}(X, S)))]$
 $\vee [A = \text{Stack}(X, Y) \wedge \text{holding}(X, S) \wedge \text{clear}(Y, S)]$
 $\vee [A = \text{Unstack}(Y, X) \wedge \text{on}(Y, X, S) \wedge \text{clear}(Y, S) \wedge \text{handempty}(S)]$
 $\vee [A = \text{Putdown}(X) \wedge \text{holding}(X, S)]$
- 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!



Basic representations for planning

- Classic approach first used in the STRIPS planner circa 1970
- States represented as a conjunction of ground literals
 - $\text{at(Home)} \wedge \neg \text{have(Milk)} \wedge \neg \text{have(bananas)} \dots$
- Goals are conjunctions of literals, but may have variables which are assumed to be existentially quantified
 - $\text{at(?x)} \wedge \text{have(Milk)} \wedge \text{have(bananas)} \dots$
- 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
- Seeking sequence of actions to attain goal.



Action representation

- Describing the preconditions and effects of an action.
- **Example:** Consider the action of moving from one location to another:

Op(Action: Go(there),

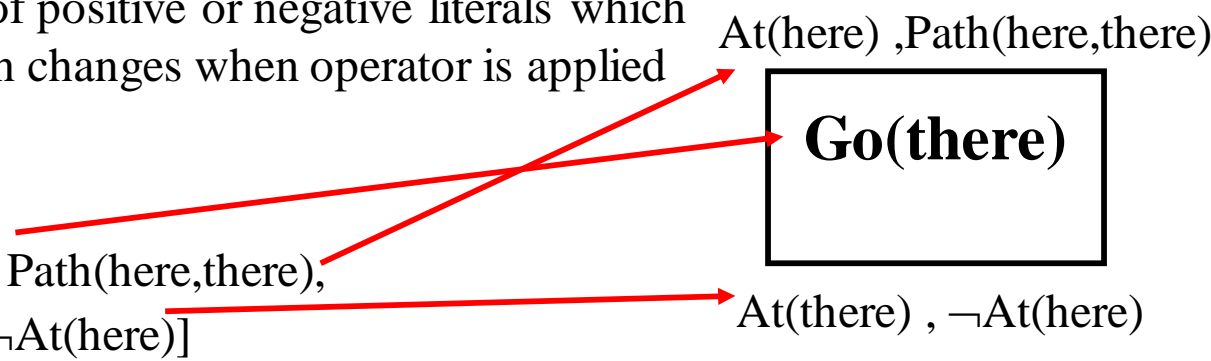
Preconds: At(here) \wedge Path(here,there),

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

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: $\text{At}(\text{here}) \wedge \text{Path}(\text{here}, \text{there})$,
Effect: $\text{At}(\text{there}) \wedge \neg \text{At}(\text{here})$]



$\text{At}(\text{here}), \text{Path}(\text{here}, \text{there})$

Go(there)

$\text{At}(\text{there}), \neg \text{At}(\text{here})$
- All variables are universally quantified
- Situation variables are implicit
 - Preconditions must be true in the state immediately before an operator is applied; effects are true immediately after



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 action 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])`



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(pickup(X),

[ontable(X), clear(X), handempty],

[holding(X)],

[ontable(X), clear(X), handempty],

[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(putdown(X),

[holding(X)],

[ontable(X), handempty,
clear(X)],

[holding(X)],

[X≠table]).



STRIPS planning

- STRIPS maintains 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.
- If current goal is not satisfied by present state, examine add lists of operators, and push operator and preconditions list on stack. (Subgoals)
- When a current goal is satisfied, POP it from stack.
- When an operator is on top of the stack, record the application of that operator in the plan sequence and use the operator's add and delete lists to update the current state.

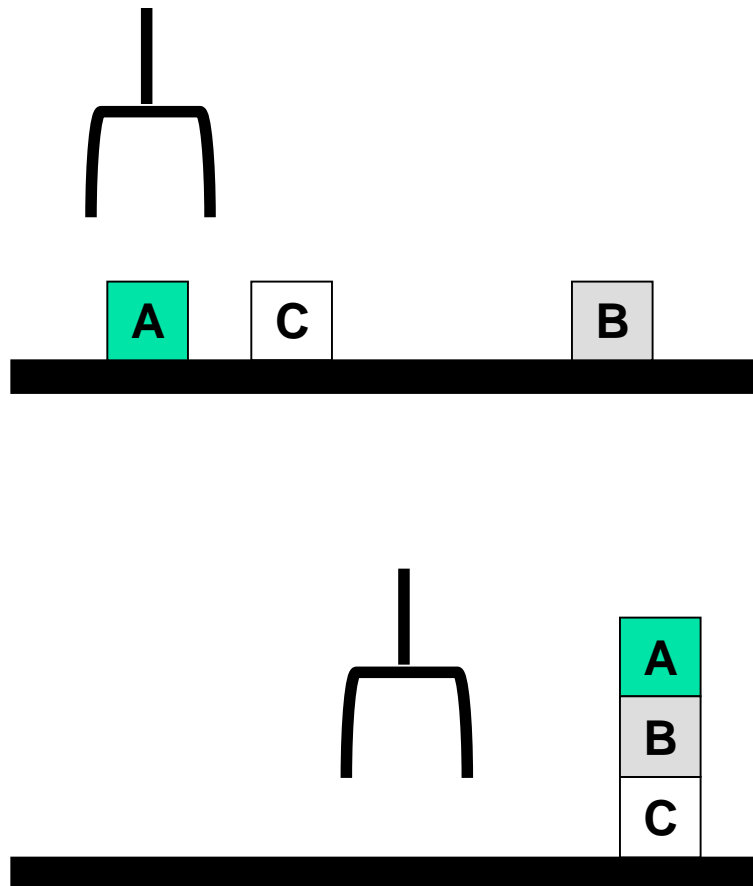
Typical BW planning problem

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
)
)

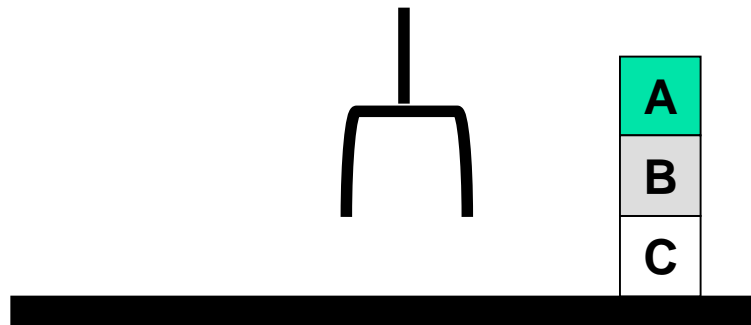
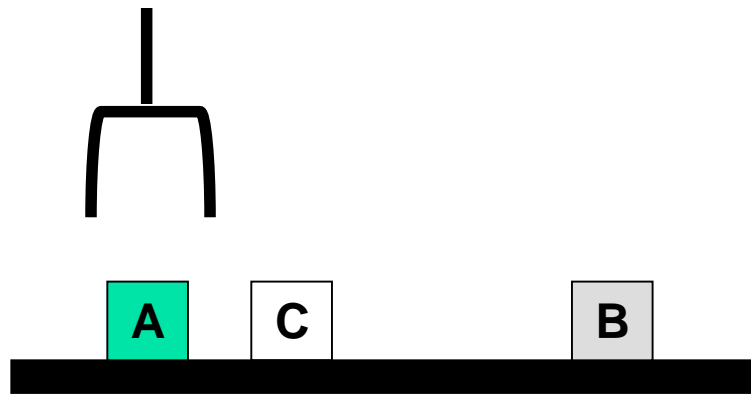
Another BW planning problem

Initial state:

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

Goal:

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

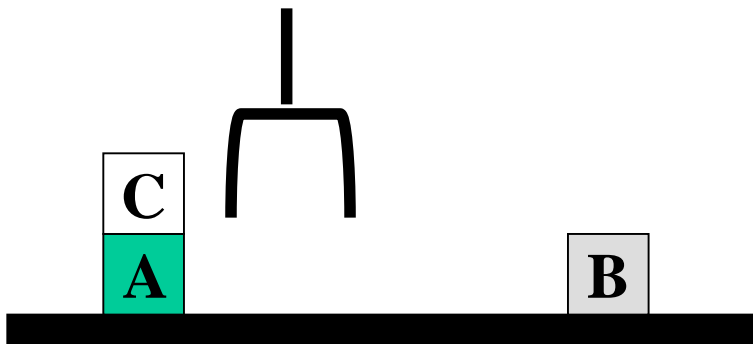


A plan:

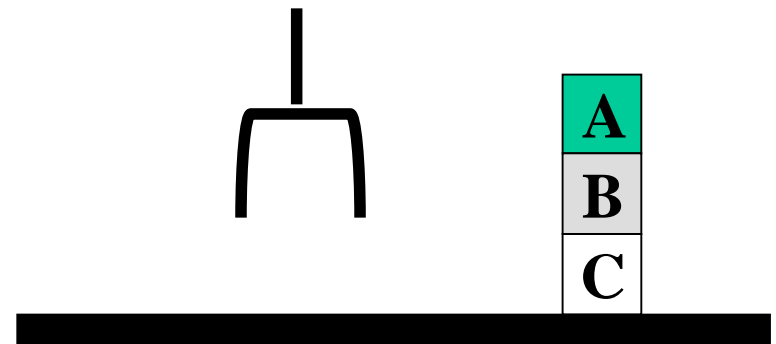
pickup(a)
stack(a,b)
unstack(a,b)
putdown(a)
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

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



Initial state



Goal state



Solution?

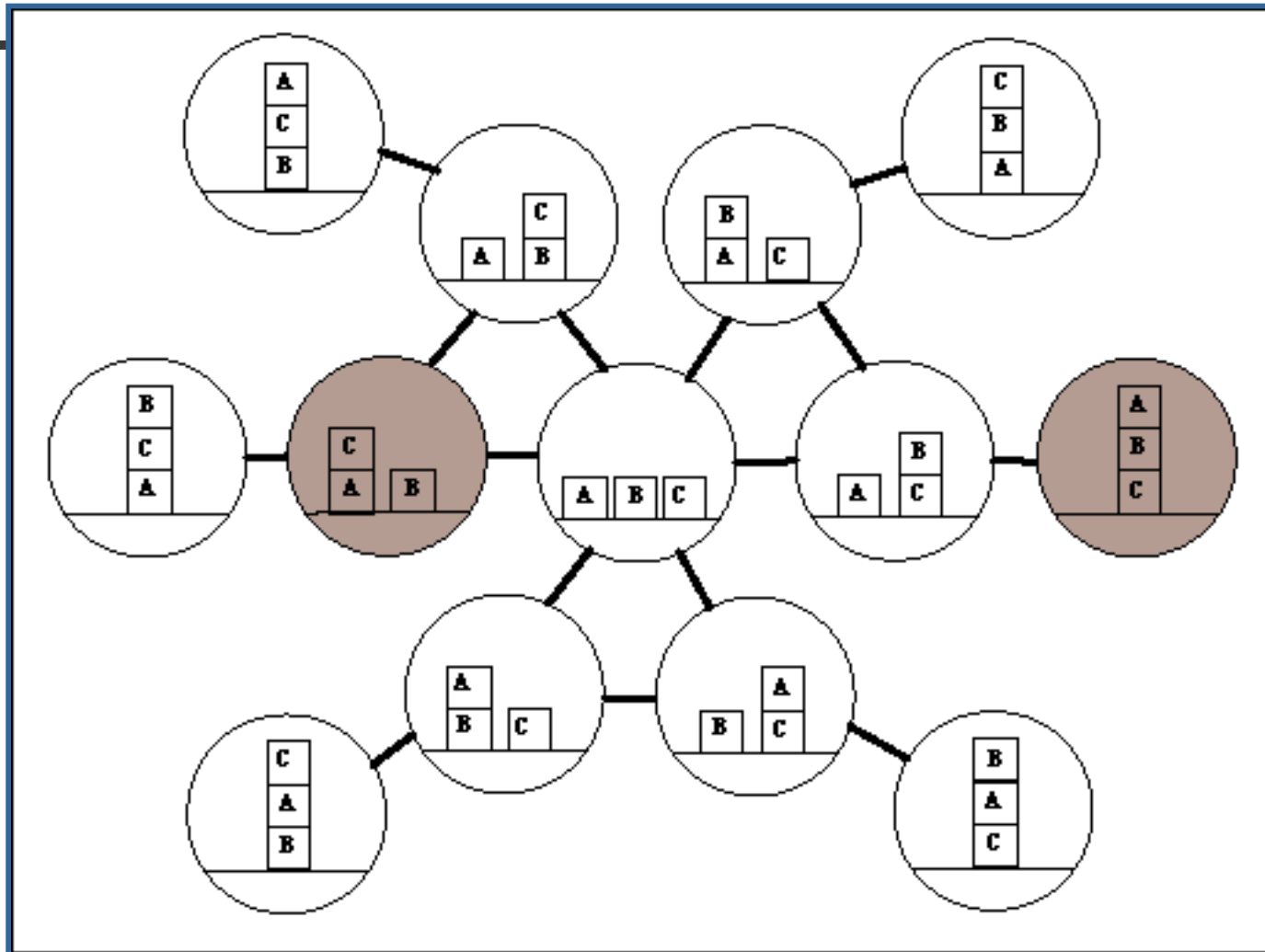
- Try to achieve each goal, then make sure they still hold.
- If not, retry achieving the goal.
- Make sure that the algorithm terminates when there is no possible plan.



The running around the block problem

- How can STRIPS handle goals where there is no net change in location (or situation)?
- Does that mean there would be no add- or delete- list? If so, there would be no reason to apply any operator.
- Goal can be “got-some-exercise” or “feel-tired” (need to add state variables).

Situation space for the blocks world





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: the planner is only considering things that are relevant to the goal



Planning heuristics

- Just as with search, we need an **admissible** heuristic that we can apply to planning states
 - Estimate of the distance (number of actions) to the goal
- Planning typically uses **relaxation** to create heuristics
 - Ignore all or selected preconditions
 - Ignore delete lists (movement towards goal is never undone)
 - Use state abstraction (group together “similar” states and treat them as though they are identical) – e.g., ignore fluents
 - Assume subgoal independence (use max cost; or if subgoals actually are independent, can sum the costs)
 - Use pattern databases to store exact solution costs of recurring subproblems



Plan-space planning

- An alternative is to **search through the space of *plans***, rather than situations (initially done in Sacerdoti's NOAH)
- Start from a **partial plan** that 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

32 [RightShoe, LeftShoe]

Plan space for the blocks world

