Planning

Planning

- The methods which focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interactions among the subparts as they are detected during the problem-solving process are often called as planning.
- Planning refers to the process of computing several steps of a problem-solving procedure before executing any of them.

Components of a planning system

- Choose the best rule to apply next based on the best available heuristic information.
- Apply the chosen rule to compute the new problem state that arises from its application.
- Detect when a solution has been found.
- Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.
- Detect when an almost correct solution has been found and employ special techniques to make it totally correct.

Choose the rules to apply

- The most widely used technique for selecting an appropriate rules to apply is first to isolate a set of differences between desired goal state and then to identify those rules that are relevant to reducing those differences.
- If several rules, a variety of other heuristic information can be exploited to choose among them

Applying Rules

- In simple systems, applying rules is easy. Each rule simply specified the problem state that would result from its application.
- In complex systems, we must be able to deal with rules that specify only a small part of the complete problem state.
- One way is to describe, for each action, each of the changes it makes to the state description.

Detecting a solution

- A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state.
- How will it know when this has been done?
- In simple problem-solving systems, this question is easily answered by a straightforward match of the state descriptions.
- One of the representative systems for planning systems is, predicate logic. Suppose that as a part of our goal, we have the predicate P(x). To see whether P(x) is satisfied in some state, we ask whether we can prove P(x) given the assertions that describe that state and the axioms that define the world model.

Detecting Dead Ends

- As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution.
- The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a dead end.
- If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.
- If search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or because little progress is being made.

Repairing an Almost Correct Solution

- The kinds of techniques we are discussing are often useful in solving nearly decomposable problems.
- One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the subproblems separately, and then check that when the subsolutions are combined, they do infact yield a solution to the original problem.

Goal Stack Planning

- In this method, the problem solver makes use of a single stack that contains both goals and operators that have been proposed to satisfy those goals.
- The problem solver also relies on a database that describes the current situation and a set of operators described as PRECONDITION, ADD, and DELETE lists.
- The goal stack planning method attacks problems involving conjoined goals by solving the goals one at a time, in order.
- A plan generated by this method contains a sequence of operators for attaining the first goal, followed by a complete sequence for the second goal etc.

Goal Stack Planning

- At each succeeding step of the problem solving process, the top goal on the stack will be pursued.
- When a sequence of operators that satisfies it is found, that sequence is applied to the state description, yielding new description.
- Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal.
- This process continues until the goal stack is empty.
- Then as one last check, the original goal is compared to the final state derived from the application of the chosen operators.
- If any components of the goal are not satisfied in that state, then those unsolved parts of the goal are reinserted onto the stack and the process is resumed.

Nonlinear Planning using Constraint Posting.

- Difficult problems cause goal interactions,
- The operators used to solve one subproblem may interfere with the solution to a previous subproblem.
- Most problems require an interwined plan in which multiple subproblems are worked on simultaneously.
- Such a plan is called nonlinear plan because it is not composed of a linear sequence of complete subplans.

Constraint Posting

- The idea of constraint posting is to build up a plan by incrementally hypothesizing operators, partial orderings between operators, and binding of variables within operators.
- At any given time in the problem-solving process, we may have a set of useful operators but perhaps no clear idea of how those operators should be ordered with respect to each other.
- A solution is a partially ordered, partially instantiated set of operators to generate an actual plan, we convert the partial order into any of a number of total orders.

Constraint Posting versus State Space search

State Space Search

- Moves in the space:
  - Modify world state via operator
- Model of time:
  - Depth of node in search space
- Plan stored in:
  - Series of state transitions

Constraint Posting Search

- Moves in the space:
  - Add operators
  - Oder Operators
  - Bind variables
  - Or Otherwise constrain plan
- Model of Time:
  - Partially ordered set of operators
- Plan stored in:
  - Single node

Algorithm: Nonlinear Planning

1. Initialize S to be the set of propositions in the goal state.
2. Remove some unachieved proposition P from S.
3. Achieve P by using step addition, promotion, declobbering, simple establishment or separation.
4. Review all the steps in the plan, including any new steps introduced by step addition, to see if any of their preconditions are unachieved. Add to S the new set of unachieved preconditions.
5. If S is empty, complete the plan by converting the partial order of steps into a total order, instantiate any variables as necessary.
6. Otherwise go to step 2.

Modal Truth Criterion

- A proposition P is necessarily true in a state S if and only if two conditions hold: there is a state T equal or necessarily previous to S in which P is necessarily asserted; and for every step C possibly before S and every proposition Q possibly codesignating with P which C denies, there is a step W necessarily between C and S which asserts R, a proposition such that R and P codesignate whenever P and Q codesignate.
- Modal truth Criterion tells us when a proposition is true.

Hierarchical Planning

- In order to solve hard problems, a problem solver may have to generate long plans.
- It is important to be able to eliminate some of the details of the problem until a solution that addresses the main issues is found.
- Then an attempt can be made to fill the appropriate details.
- Early attempts to do this involved the use of macro operators.
- But in this approach, no details were eliminated from actual descriptions of the operators.
- As an example, suppose you want to visit a friend in Europe but you have a limited amount of cash to spend. First preference will be find the airfares, since finding an affordable flight will be the most difficult part of the task. You should not worry about getting out of your driveway, planning a route to the airport etc, until you are sure you have a flight.

ABSTRIPS

- ABSTRIPS actually planned in a hierarchy of abstraction spaces, in each of which preconditions at a lower level of abstraction were ignored.
- ABSTRIPS approach is as follows:
  - First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
  - These values reflect the expected difficulty of satisfying the precondition.
  - To do this, do exactly what STRIPS did, but simply ignore the preconditions of lower than peak criticality.
  - Once this is done, use the constructed plan as the outline of a complete plan and consider preconditions at the next-lowest criticality level.
  - Because this approach explores entire plans at one level of detail before it looks at the lower-level details of any one of them, it has been called **length-first approach.**

Hierarchical Planning

- The assignment of appropriate criticality value is critical to the success of this hierarchical planning method.
- Those preconditions that no operator can satisfy are clearly the most critical.
- Example, solving a problem of moving robot, for applying an operator, PUSH-THROUGH DOOR, the precondition that there exist a door big enough for the robot to get through is of high criticality since there is nothing we can do about it if it is not true.

Reactive Systems

- The idea of reactive systems is to avoid planning altogether, and instead use the observable situation as a clue to which one can simply react.
- A reactive system must have access to a knowledge base of some sort that describes what actions should be taken under what circumstances.
- A reactive system is very different from the other kinds of planning systems we have discussed because it chooses actions one at a time.
- It does not anticipate and select an entire action sequence before it does the first thing.
- Example is a Thermostat. The job of the thermostat is to keep the temperature constant.

Thermostat

- Is an example for reactive systems.
- Its job is to keep the temperature constant inside a room.
- One might imagine a solution to this problem that requires significant amounts of planning, taking into account how the external temperature rises and falls during the day, how heat flows from room to room, and so forth.
- Real thermostat uses simple pair of situation-action rules:
  1. If the temperature in the room is k degrees above the desired temperature, then turn the airconditioner on.
  2. If the temperature in the room is k degrees below desired temperature, then turn the airconditioner off.

Reactive Systems

- Ractive systems are capable of surprisingly complex behaviours.
- The main advantage reactive systems have over traditional planners is that they operate robustly in domains that are difficult to model completely and accurately.
- Reactive systems dispense with modeling altogether and base their actions directly on their perception of the world.
- Another advantage of reactive systems is that they are extremely responsive, since they avoid the combinatorial explosion involved in deliberative planning.
- This makes them attractive for real time tasks such as driving and walking.

Other Planning Techniques

- Triangle tables
- Metaplanning
- Macro-operators
- Case based planning.

Blocks World

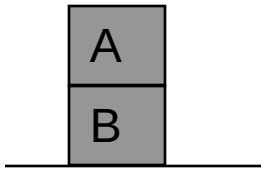
- Inorder to compare the variety of methods of planning, we should find it useful to look at all of them in a single domain that is complex enough that the need for each of the mechanisms is apparent yet simple enough that easy-to follow examples can be found.

Blocks world problem

- There is a flat surface on which blocks can be placed
- There are a number of square blocks, all the same size.
- They can be stacked one upon the other.
- There is robot arm that can manipulate the blocks

Actions of the robot arm

- UNSTACK(A,B)
- STACK(A,B)
- PICKUP(A)
- PUTDOWN(A)



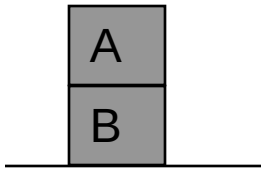
- Notice that the robot arm can hold only one block at a time.

Predicates

- Inorder to specify both the conditions under which an operation may be performed and the results of performing it, we need the following predicates:
- ON(A,B)
- ONTABLES(A)
- CLEAR(A)
- HOLDING(A)
- ARMEMPTY

Simple position

- State S0
- ON(A,B, S0) ^ ONTABLE(B, S0)^CLEAR(A,S0)



If we execute UNSTACK(A,B) in state S0, in the resulting state S1:

HOLDING(A,S1)^CLEAR(B,S1)

To enable the complete situation to be described, we provide a set of rules called **frame axioms**, that describe components of the state that are not affected by each operator.

- ONTABLE(x,s) -> ONTABLE(z, DO(UNSTACK(x,y),s))
- DO is a function that specifies for a given state and a given action, the new state that results from the execution of the action.

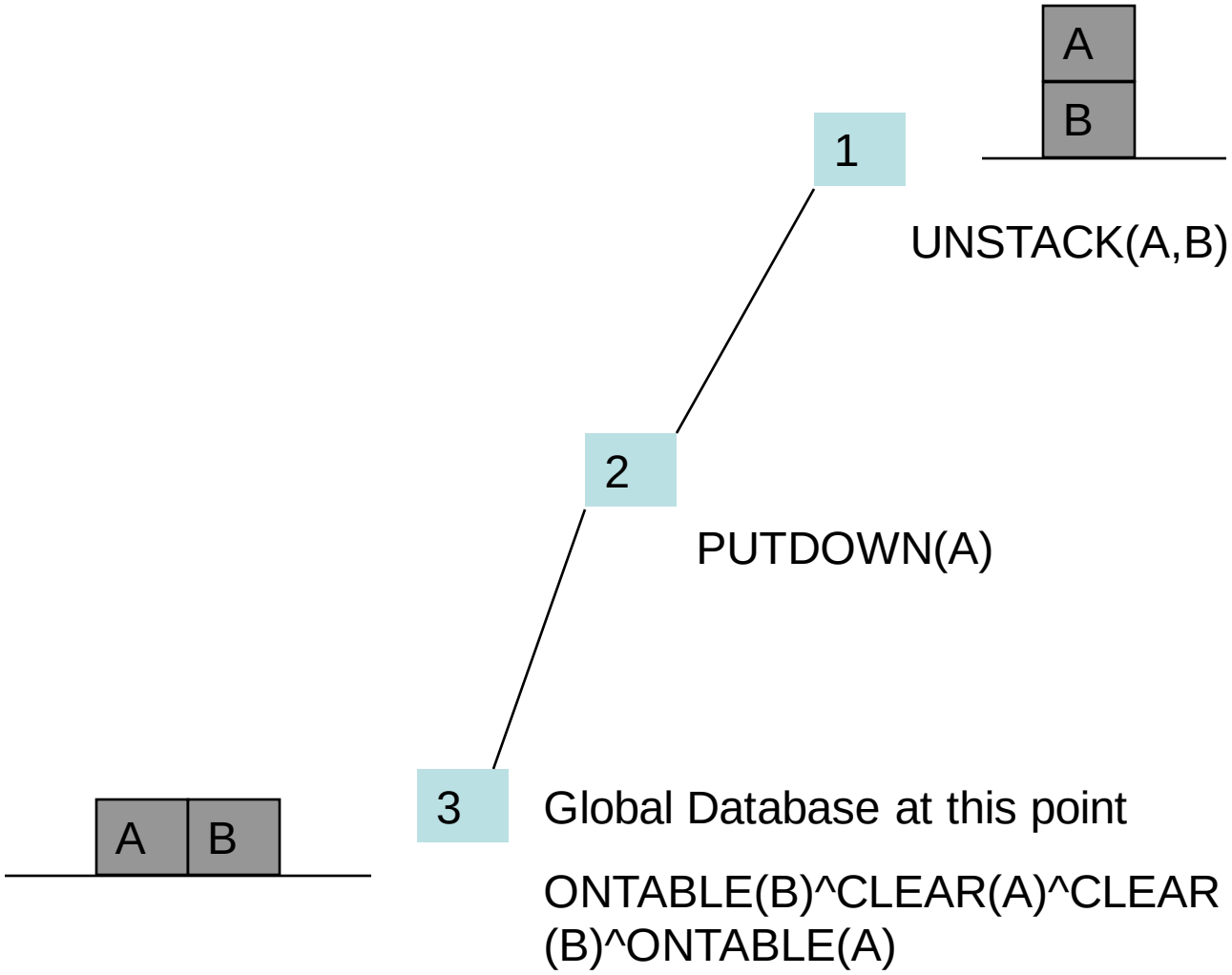
Robot –problem solving system (STRIPS)

- List of new predicates that the operator causes: ADD, DELETE
- PRECONDITIONS list contains those predicates that must be true for the operator to be applied.

STRIPS style operators for BLOCKs World

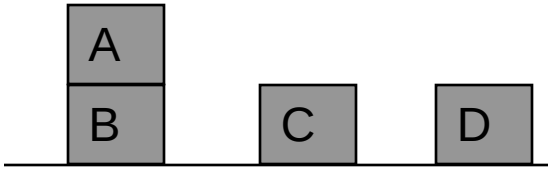
- STACK(x,y)
  - P: CLEAR(y)^HOLDING(x)
  - D: CLEAR(y)^HOLDING(x)
  - A: ARMEMPTY^ON(x,y)
- PICKUP(x)
  - P: CLEAR(x) ^ ONTABLE(x) ^ARMEMPTY
  - D: ONTABLE(x) ^ ARMEMPTY
  - A: HOLDING(x)

A simple Search Tree

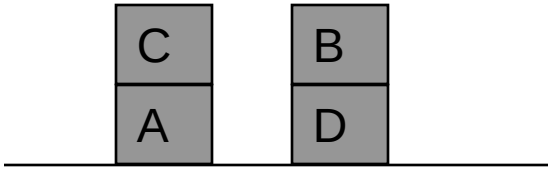


Goal Stack Planning

- To start with goal stack is simply:
- ON(C,A)^ON(B,D)^ONTABLE(A)^ONTABLE(D)
- This problem is separate into four subproblems, one for each component of the goal.
- Two of the subproblems ONTABLE(A) and ONTABLE(D) are already true in the initial state.
- Alternative 1: Goal Stack:
- ON(C,A)
- ON(B,D)
- ON(C,A)^ON(B,D)^OTAD
- Alternative 2: Goal stack:
- ON(B,D)
- ON(C,A)
- ON(C,A)^ON(B,D)^OTAD



Start:
ON(B,A)^ONTABLE(A) ^
ONTABLE(C)
^ONTABLE(D)
^ARMEMPTY



Goal: ON(C,A)^ON(B,D)^
ONTABLE(A)^ONTABLE(D)

Exploring Operators

- Pursuing alternative 1, we check for operators that could cause ON(C,A)
- Of the 4 operators, there is only one STACK. So it yields:
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- Preconditions for STACK(C,A) should be satisfied, we must establish them as subgoals:
- **CLEAR(A)**
- **HOLDING(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- Here we exploit the Heuristic that if HOLDING is one of the several goals to be achieved at once, it should be tackled last.

Goal stack Planning contd

- Next we see if CLEAR(A) is true. It is not. The only operator that could make it true is UNSTACK(B,A). This produces the goal stack:
- **ON(B,A)**
- **CLEAR(B)**
- **ON(B,A)^CLEAR(B)^ARMEMPTY**
- **UNSTACK(B,A)**
- **HOLDING(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- We see that we can pop predicates on the stack till we reach HOLDING(C) for which we need to find suitable operator
- The operators that might make HOLDING(C) true : PICKUP(C) and UNSTACK(C,x). Without looking ahead, since we cannot tell which of these operators is appropriate, we create two branches of the search tree corresponding to the following goal stacks:

ALT1:	**ALT2:**
ONTABLE(C)	**ON(C,x)**
CLEAR(C)	**CLEAR(C)**
ARMEMPTY	**ARMEMPTY**
ONTABLE(C) ^CLEAR(C)^ARMEMPTY	**ON(C,x)^CLEAR(C)^ ARMEMPTY**
PICKUP(C)	**UNSTACK(C,x)**
CLEAR(A)^HOLDING(C)	**CLEAR(A)^HOLDING (C)**
STACK(C,A)	**STACK(C,A)**
ON(B,D)	**ON(B,D)**
ON(C,A)^ON(B,D)^OTAD	**ON(C,A)^ON(B,D)^OTAD**

Choosing Alternative

- How should our program choose now between alternative 1 and alternative 2?
- We can tell that picking up C (alt 1) is better than unstacking it because it is not currently on anything. So to unstack it, we would first have to stack it.This would be waste of effort.
- But how could the program know that?
- If we pursue the alternative 1, the top element on the goal stack is ONTABLE(C) which is already satisfied, so we pop it off. CLEAR(C) is also satisfied and is popped off.
- The remaining precondition of PICKUP(C) is ARMEMPTY which is not satisfied since HOLDING(B) is true.So we apply the operator STACK(B,D). This makes the Goal stack:
- **CLEAR(D)**
- **HOLDING(B)**
- **CLEAR(D)^HOLDING(B)**
- **STACK(B,D)**
- **ONTABLE(C)^CLEAR(C)^ARMEMPTY**
- **PICKUP(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**

Complete plan

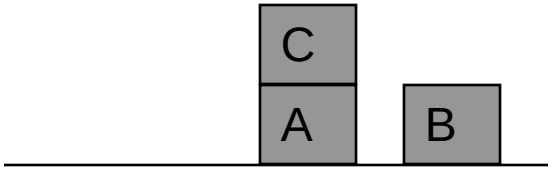
1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A,B)
5. UNSTACK(A,B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B,C)
9. PICKUP(A)
10. STAKC(A,B)

A slightly Harder Blocks problem

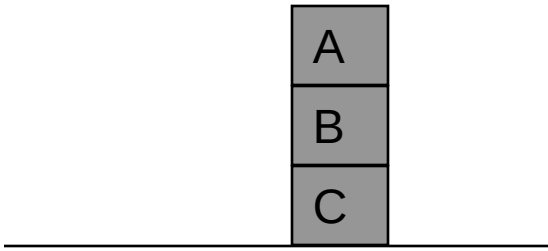
- Start
- Alt1:
- ON(A,B)
- ON(B,C)
- ON(A,B)^ON(B,C)
- Alt2:
- ON(B,C)
- ON(A,B)
- ON(A,B)^ON(B,C)

Complete Plan:

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A,B)
5. UNSTACK(A,B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B,C)
9. PICKUP(A)
10. STACK(A,B)



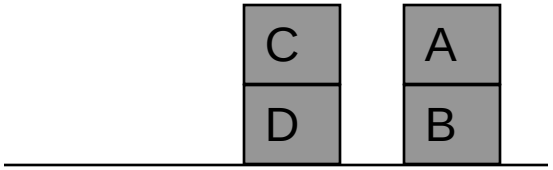
Start:
ON(C,A)^ONTABLE(A) ^
ONTABLE(B)
^ARMEMPTY



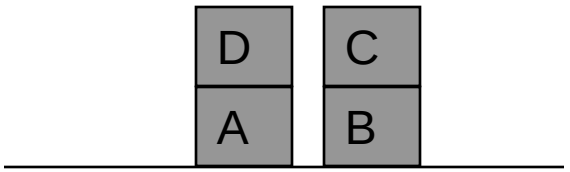
Goal: ON(A,B)^ON(B,C)

Problem

- Show how the STRIPS would solve this problem?
- Show how the TWEAK would solve this problem?



Start:
ON(C,D)^ON(A,B)^ONTABLE(D) ^ ONTABLE(B) ^ARMEMPTY

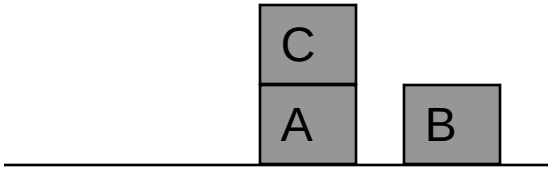


Goal:
ON(C,B)^ON(D,A)^ONTABLE(A)^ONTABLE(B)

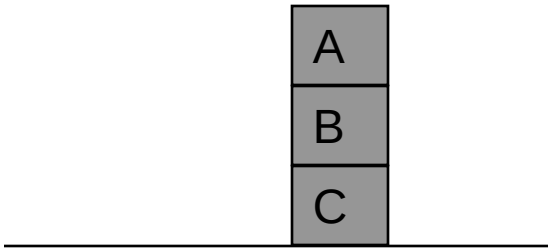
Sussman Anomaly

- it is an *anomaly* because a non-interleaved planner cannot solve the problem
- 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.
- This is an example for nonlinear plan. A good plan for the solution of this problem is the following:

1. Begin work on the goal ON(A,B) by clearing A, thus putting C on the table.
2. Achieve the goal ON(B,C) by stacking B on C.
3. Complete the goal ON(A,B) by stacking A on B.



Start:
ON(C,A)^ONTABLE(A) ^
ONTABLE(B)
^ARMEMPTY



Goal: ON(A,B)^ON(B,C)

Heuristics for Planning using Constraint Posting (TWEAK)

1. Step addition – creating new steps for a plan.
2. Promotion – Constraining one step to come before another in a final plan.
3. Declobbering – Placing one (possibly new) step S2 between two old steps S1 and S3 such that S2 reasserts some precondition of S3 that was neglected (or “clobbered”) by S1.
4. Simple establishment – Assigning a value to a variable, in order to ensure the preconditions of some step.
5. Separation – Preventing the assigment of certain values to a variable.

Two steps with respect to ON(A,B) and ON(B,C)

CLEAR(B) *HOLDING(A)	CLEAR(C) *HOLDING(B)
STACK(A, B)	STACK(B, C)
ARMEMPTY ON(A,B) ¬CLEAR(B) ¬HOLDING(A)	ARMEMPTY ON(B,C) ¬CLEAR(C) ¬HOLDING(B)

Each step is written with its preconditions above it and its postconditions below it.

Delete postconditions are marked with a negation symbol (¬)

Neither can be executed right away because some of their preconditions are not satisfied.

An unachieved precondition is marked with *.

Step Addition

- Introducing new steps to achieve goals or preconditions is called Step addition.
- It is one of the heuristics used in generating nonlinear plans.
- Step addition is a very basic method dating back to GPS, where Means-Ends Analysis was used to pick operators with postconditions corresponding to desired states.
- To achive the preconditions of the two steps, we can use step addition again:

*CLEAR(A) ONTABLE(A) *ARMEMPTY	*CLEAR(B) ONTABLE(B) *ARMEMPTY
PICKUP(A)	PICKUP(B)
¬ONTABLE(A) ¬ARMEMPTY HOLDING(A)	¬ONTABLE(B) ¬ARMEMPTY HOLDING(B)

Partial Ordering and Promotion

- Adding PICKUP steps is not enough to satisfy the *HOLDING preconditions of the STACK steps.
- If in the eventual plan, the PICKUP steps were to follow the STACK steps, then the *HOLDING preconditions would need to be satisfied by some other set of steps.
- In this case we want each PICKUP step should precede its corresponding STACK step
- PICKUP(A) <- STACK(A,B)
- PICKUP(B) <- STACK(B,C)
- We now have four partially ordered steps and four unachieved preconditions.
- To achieve precondition CLEAR(B), we use Heuristic known as PROMOTION.
- **Promotion** amounts to posting a constraint that one step must precede another in eventual plan.
- We can achieve CLEAR(B) by stating that the PICKUP(B) step must come before the STACK(A,B) step:
- PICKUP(B) <- STACK(A,B)

Declobbering

- A third heuristic called Declobbering can help achieve *ARMEMPTY precondition in the PICKUP(A) step.
- PICKUP(B) asserts ¬ARMEMPTY, but if we can insert another step between PICKUP(B) and PICKUP(A) to reassert ARMEMPTY, then the precondition will be achieved. The STACK(B,C) does the trick, so we post another constraint:
- PICKUP(B) <- STACK(B,C) <-PICKUP(A)
- The step PICKUP(B) is said to “clobber” PICKUP(A)’s precondition. STACK(B,C) is said to “Declobber” it.

Simple Establishment

- *ON(x, A)
- *CLEAR(x)
- *ARMEMPTY
- ----------
- UNSTACK(x,A)
- ----------
- ¬ARMEMPTY
- CLEAR(A)
- HOLDING(A)
- ¬ON(x,A)
- A variable x is introduced because the only precondition we are interested in is CLEAR(A). Whatever block is on top of A is irrelevent.
- Variables allow us to avoid committing to particular instantiations of operators.
- We have three unachieved preconditions. We can achieve ON(x,A) easily by constraining the value of x to be block C. This works because block C is on block A in the initial state.
- This is called Simple establishment and allows us to state that two different propositions must be ultimately instantiated to the same proposition.
- x = C in step UNSTACK(x, A)

Plan ordering and Variable Binding

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(B)
4. STACK(B,C)
5. PICKUP(A)
6. STACK(A,B)

We used four different Heuristics to synthesize it : Step addition, promotion, declobbering and simple establishment.

Some examples

- If our goal is to have a white fence around our yard and we currently have brown fence, we would select operators whose results involves change of colour of an object. If on the other hand, we have no fence, we must first consider operators that involve constructing a fence.
- We have a plan for baking an angel food cake. It involves separating some eggs. While carrying out the plan, we turn out to be slightly clumsy and one of the egg yolk falls into the dish of whites. We do not need to create a completely new plan. Instead, we simply redo the egg-separating step until we get it right and then continue with the rest of the plan.

Some examples

- Suppose that we want to move all the furnitures out of a room. This problem can be decomposed into a set of smaller problems, each involving moving one piece of furniture out of the room.But if there is a bookcase behind the couch, then we must move the couch before the bookcase.
- Suppose we have a fixed supply of paint; some white, some pink and some red. We want to paint a room so that it has light red walls and a white ceiling. We could produce light red paint by adding some white paint to red. But then we could not paint the ceiling white. So this approach should be abandoned in favor of mixing the pink and red paints together.

Example problem of cleaning a kitchen

- Cleaning the stove or refrigerator will get the floor dirty.
- To clean the oven, it is necessary to apply oven cleaner and then to remove the cleaner.
- Before the floor can be washed, it must be swept.
- Before the floor can be swept, the garbage must be taken out.
- Cleaning the refrigerator generates garbage and messes up the counters.
- Washing the counters or the floor gets the sink dirty.

- Show how the technique of planning using goal stack could be used to solve this problem.

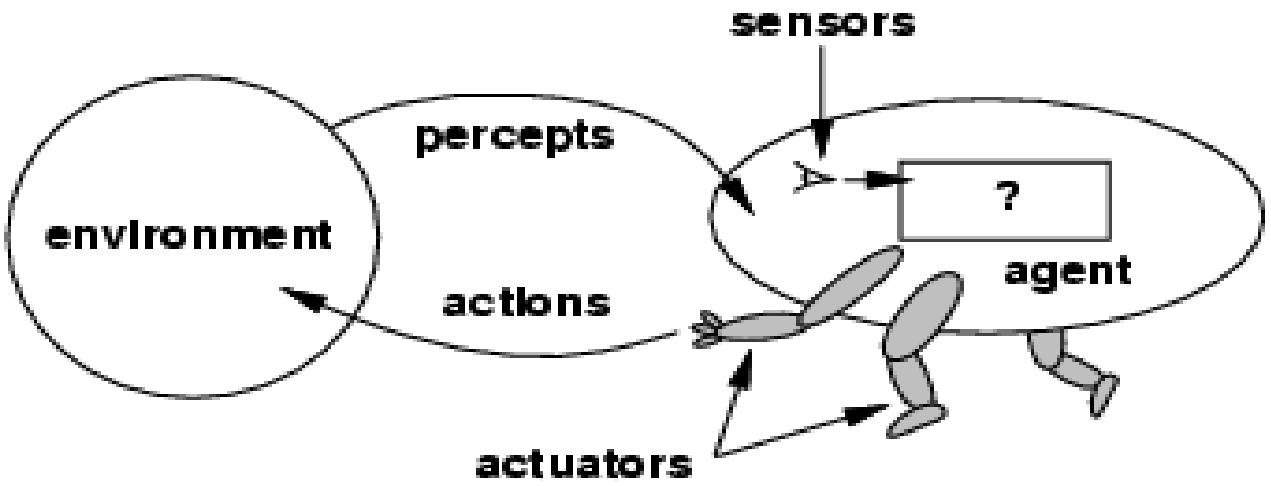
Intellignt Agent : Design and Construction

Agents

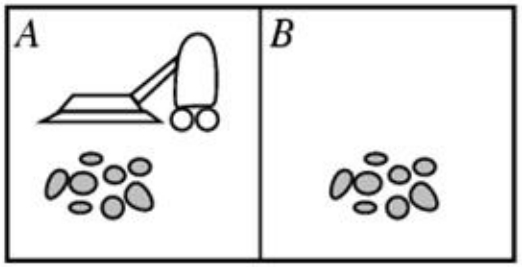
An agent is anything that can be viewed as

--- perceiving its environment through sensors and

---acting upon that environment through actuators or effectors.


sensors
percepts
environment
?
agent
actions
actuators

Vacuum-cleaner world



- Two locations: A and B
- Percepts: location and contents, e.g., [A, Dirty]
- Actions: Left, Right, Suck, NoOp

Percept sequence	Actions
[A, Clean]	Right
[A, Dirty]	Suck
[B ,Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean],[A, Dirty]	Suck
…	…
[A, Clean],[A .Clean],[A, Clean]	Right
[A, Clean],[A, Clean],[A, Clean]	Suck

One simple function is :

if the current square is dirty then suck, otherwise move to the other square

Specifying the task environment (PEAS)

- Performance Measure
- Environment
- Sensors
- Actuators

In designing an agent, the first step must always be to specify the task environment (PEAS) as fully as possible

PEAS for an automated taxi driver

- Performance measure: Safe, fast, legal, comfortable trip, maximize profits
- Environment: Roads, other traffic, pedestrians, customers
- Actuators: Steering wheel, accelerator, brake, signal, horn
- Sensors: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

PEAS for a medical diagnosis system

- Performance measure: Healthy patient, minimize costs, lawsuits
- Environment: Patient, hospital, staff

- Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
- Sensors: Keyboard (entry of symptoms, findings, patient's answers)

PEAS for Interactive English tutor

- Performance measure: Maximize student's score on test
- Environment: Set of students
- Actuators: Screen display (exercises, suggestions, corrections)
- Sensors: Keyboard

Environment types

The critical decision an agent faces is determining which action to perform to best satisfy its design objectives.

- Accessible vs. Inaccessible
- Deterministic vs. stochastic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Single agent vs. multiagent

Accessible vs. Inaccessible:- An environment is fully accessible if an agent's sensors give it access to the complete state of the environment at each point in time where as An environment might be inaccessible because of noisy and inaccurate sensors;

- Examples: vacuum cleaner with local dirt sensor, taxi driver

Deterministic vs. stochastic:- The environment is deterministic if the next state of the environment is completely determined by the current state and the action executed by the agent where as If the environment is partially observable then it could appear to be stochastic

- Examples: Vacuum world is deterministic while taxi driver is not

Episodic vs. sequential:-In episodic environments, the agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself. Examples: classification tasks.

- In sequential environments, the current decision could affect all future decisions. Examples: chess and taxi driver

Static vs. dynamic: The static environment is unchanged while an agent is deliberating where as Dynamic environments continuously ask the agent what it wants to do.

- Examples: taxi driving is dynamic, crossword puzzles are static.

Discrete vs. continuous: A limited number of distinct, clearly defined states, percepts and actions.

Examples: Chess has discrete set of percepts and actions. While Taxi driving has continuous states, and actions

Single agent vs. multiagent: An agent operating by itself in an environment is single agent

Examples: Crossword is a single agent while taxi driving is a multiagent environment

Environment types

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical Diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image Analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English Tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

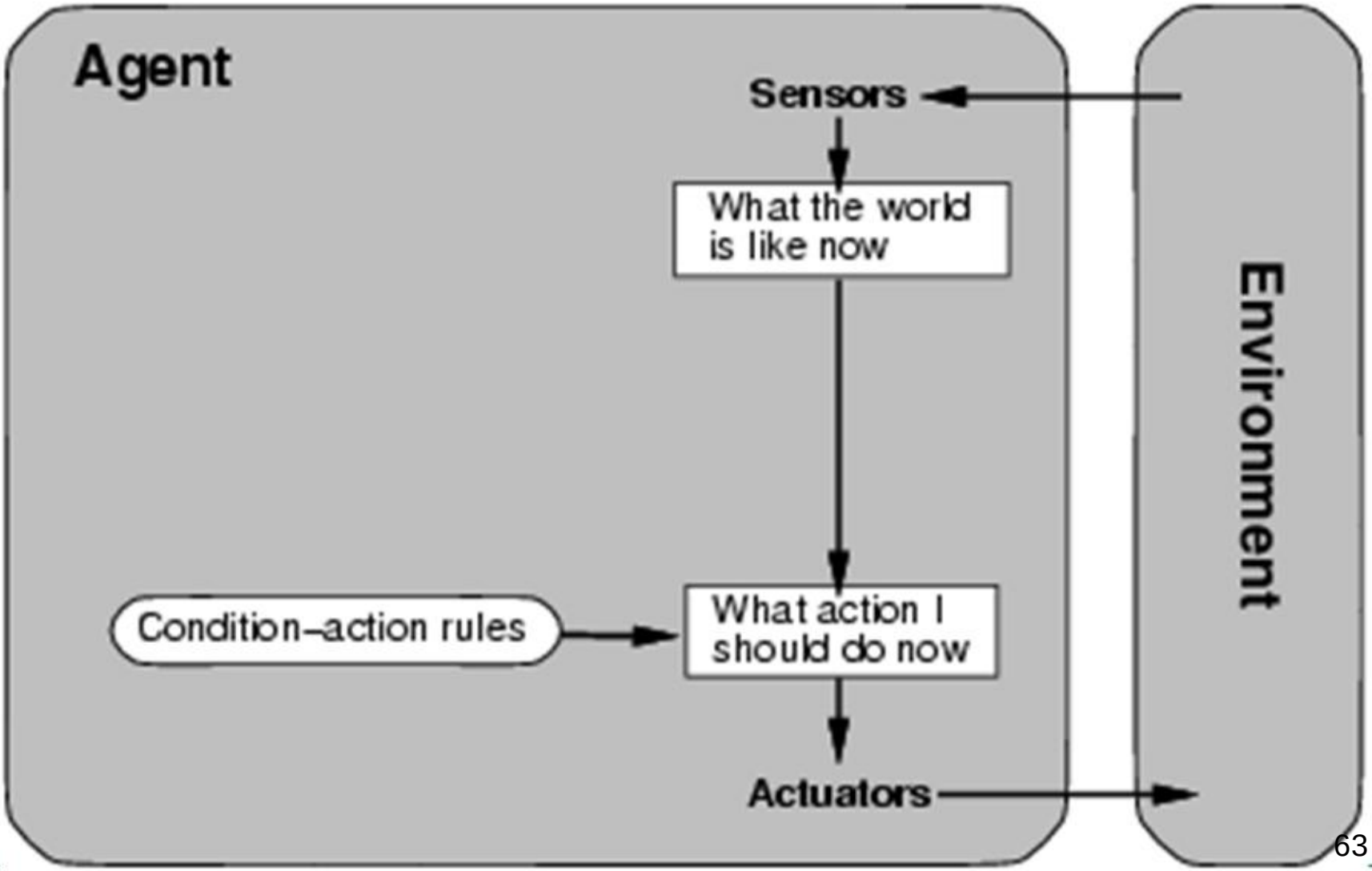
- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Agent types

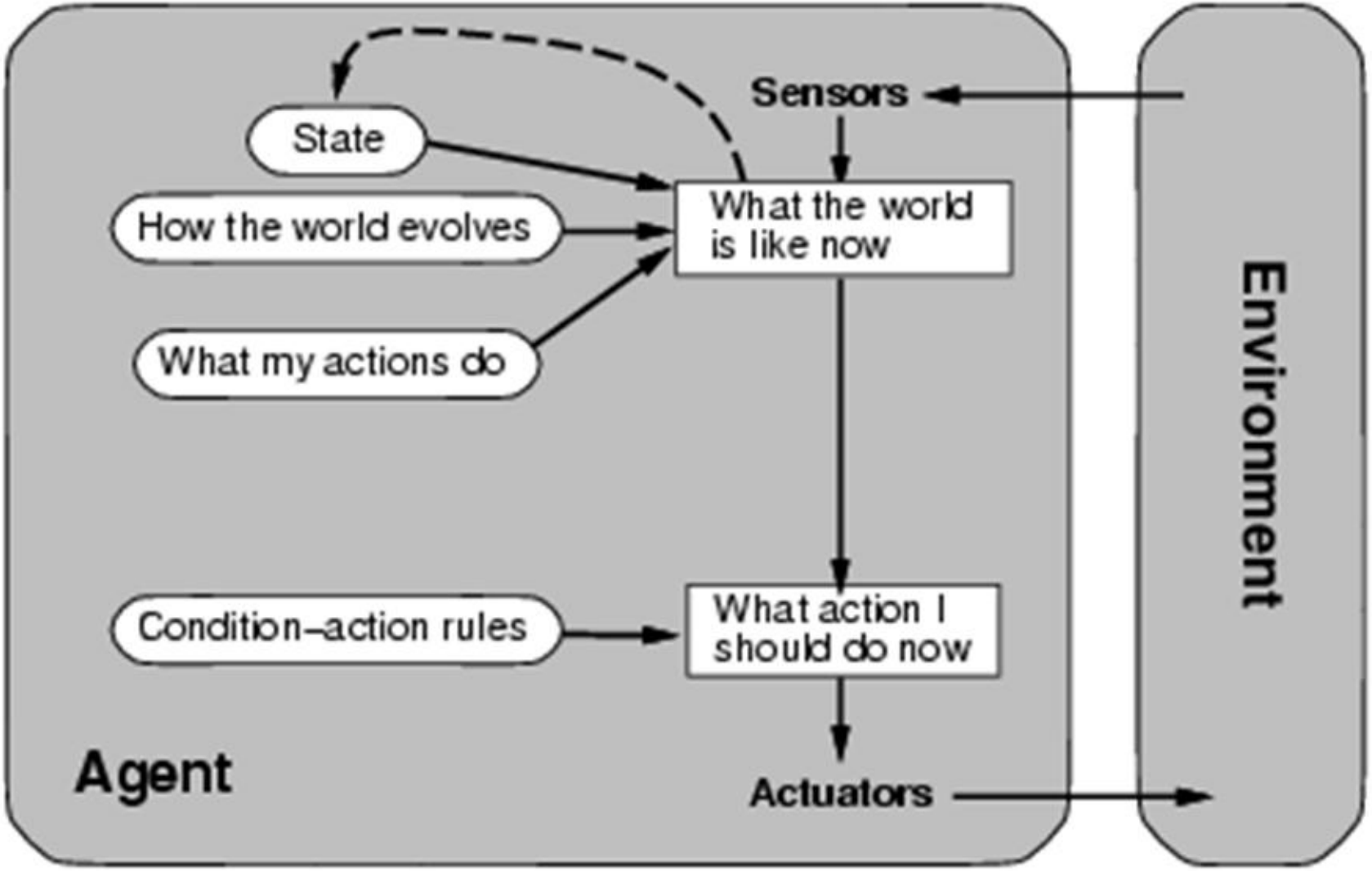
Four basic types in order to increasing generality:

- Simple reflex agents
- Model-based reflex agents
- Goal based agents
- Utility based agents

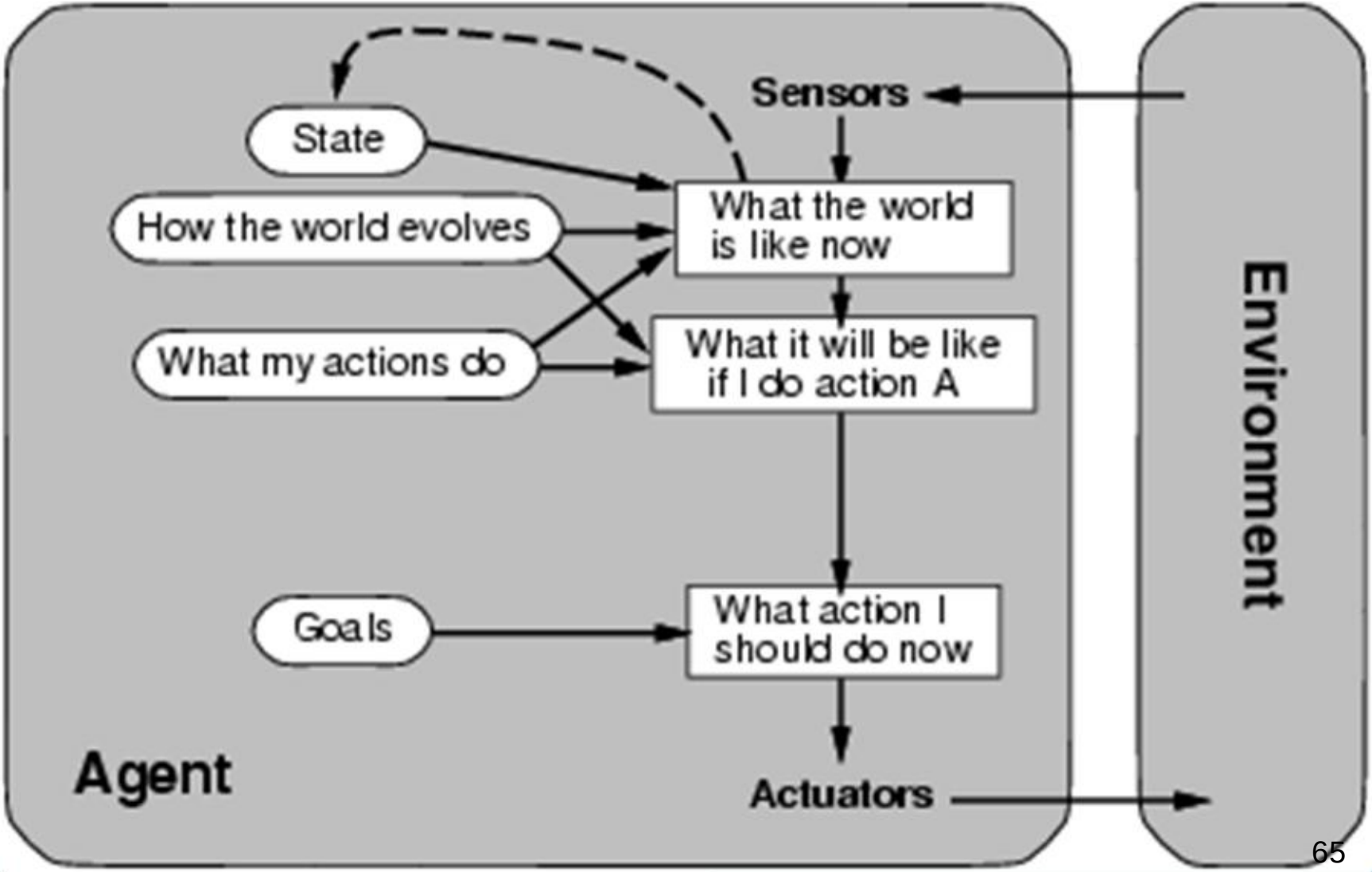
Simple reflex agents



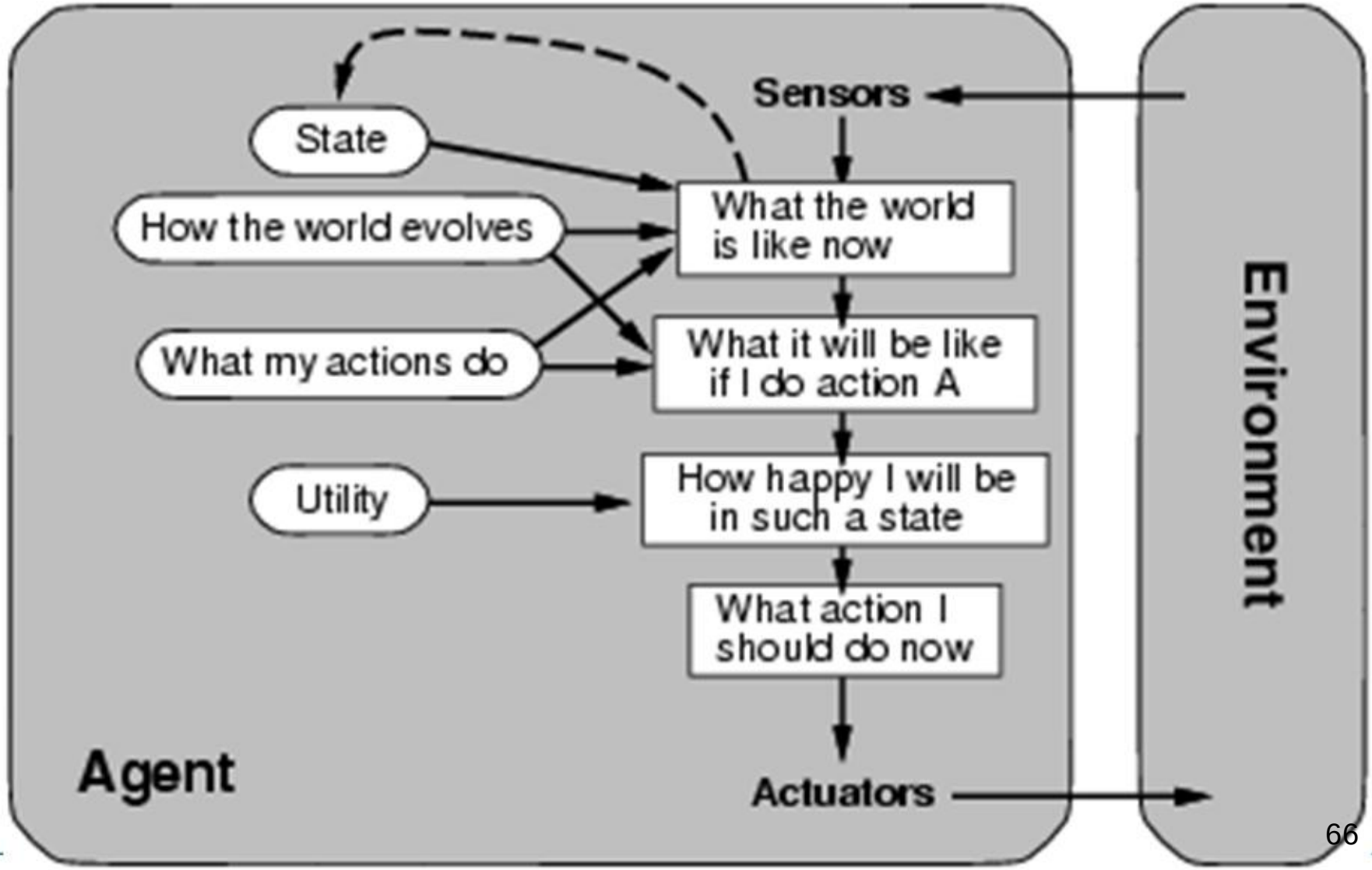
Model-based reflex agents



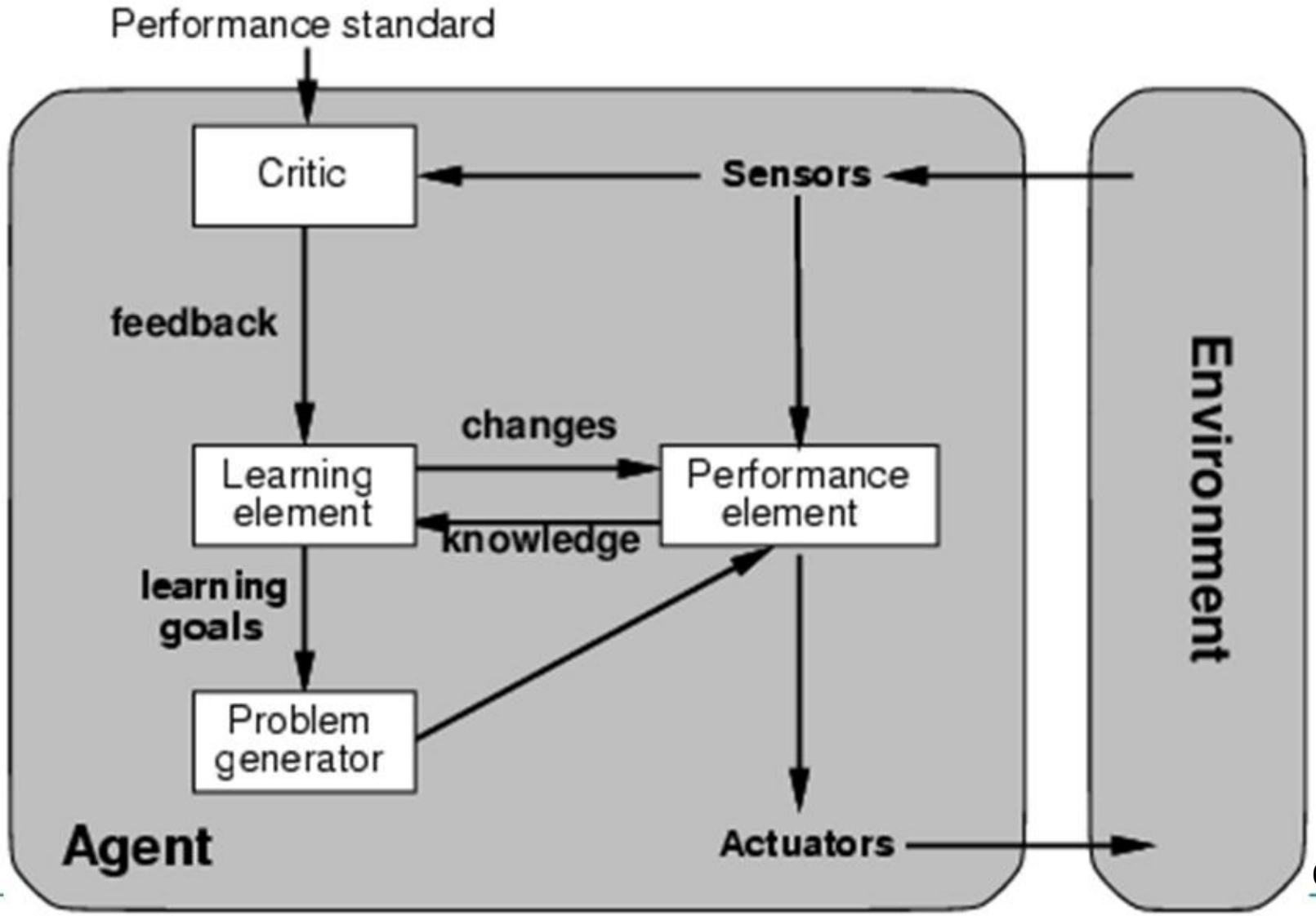
Goal-based agents



Utility-based agents



Learning agents



Agents and Objects

- The designers of an object oriented system work towards a common goal where as agents may be built for different and organizations, no such common goal can be assumed.
- “Objects invoke, agents request” or as one researcher said that

“Objects do it for free; agents do it for money”.

Agents and Expert systems

- Expert system could not be considered as agents.
- Expert systems typically do not exist in an environment they are disembodied.
- expert systems do not act on any environment but instead give feedback or advice to a third party.
- This does not mean that an expert system cannot be an agent.
- In fact, some real-time (typically process control) expert systems are agents.

What Kinds of Things Can Intelligent Agents Do?

- Search for information automatically
- Answer specific questions
- Inform you when an event has occurred.
- Provide custom news to you on a just-in-time format
- Provide intelligent tutoring
- Find you the best prices on nearly any item
- Provide automatic services, such as checking web pages for changes or broken links

Features of an Agent

- Responsive (explicit: programmed, implicit:learn)
- Predictable
- Interactive (accessible)
- Trustworthy
- Expertise
- Skill
- Quick
- Accurate

Know how, with no how

- One way to convey to an agent the task it should perform is to simply write a program that the agent should execute.
- The agent will do exactly as told and no more - if an unforeseen circumstance arises the agent will have no clue as to how it should react.
- Thus, what we really want is to tell our agent what to do without really telling it how to do it.

But how do they do it?

- How is this know-how incorporated into software?
- Shoham introduced a new programming paradigm based on societal views of computation that he called agent-oriented programming.
- He called the programming language AGENT0.
- In AGENT0, an agent is specified in terms of a set of capabilities (things the agent can do), a set of initial beliefs, a set of initial commitments (an agreement to perform a particular action at a particular time) and a set of commitment rules.

INTELLIGENT AGENTS

- Some researchers in 1995 define an intelligent agent as one that is capable of flexible autonomous action to meet its design objectives. Flexible means**:**
- **Reactivity:** intelligent agents perceive and respond in a timely fashion to changes that occur in their environment in order to satisfy their design objectives.
- **Pro-activeness:** reacting to an environment by mapping a stimulus into a set of responses is not enough.
- **Social ability:** intelligent agents are capable of interacting with other agents.

Other properties

- **Mobility**: the ability to move around an electronic environment
- **veracity:** an agent will not knowingly communicate false information.
- **Benevolence:** agents do not have conflicting goals and every agent will therefore always try to do what is asked of it.
- **Rationality:** an agent will act in order to achieve its goals insofar as its beliefs permit.
- **Learning/adaptation**: agents improve performance over time

Conclusion

- Agent-based systems technology is a vibrant and rapidly expanding field of academic research and business world applications.
- Agent technology is greatly hyped as a panacea for the current ills of system design and development, but the developer is cautioned to be aware of the pitfalls inherent in any new and untested technology.
- The potential is there but the full benefit is yet to be realized.
- Much work is yet to be done.

Chaining or reasoning

Rule-based Systems

A rule based system is also called a production system.
A production rule is an:

IF situation THEN action
IF premise THEN conclusion
IF antecedent THEN consequent

Cont…

Rule-based systems are the most popular type of expert systems.

Two inference methods are used in rule-based systems

- ✓Forward reasoning (Forward chaining, data driven reasoning)

  start with known data and progress to a conclusion.

- ✓Backward reasoning (Backward chaining, goal driven reasoning)

  start with a possible conclusion and try to prove its validity by searching for evidance.

Why are rule-based systems more popular?

Modular nature (easy to expand)

Explanation facilities easily implemented (by keeping track of the rules that fire)

Similarity to human cognitive process (work of Newell and Simon)

Forward Reasoning

Forward Chaining

Data Driven Reasoning

Two types

- Forward chaining:- starts with the data available and uses the inference rules to conclude more data until a desired goal is reached.
- An inference engine using forward chaining searches the inference rules until it finds one in which the if-clause is known to be true.
- It then concludes the then-clause and adds this information to its data.
- It would continue to do this until a goal is reached. Because the data available determines which inference rules are used.
- this method is also called *data driven*

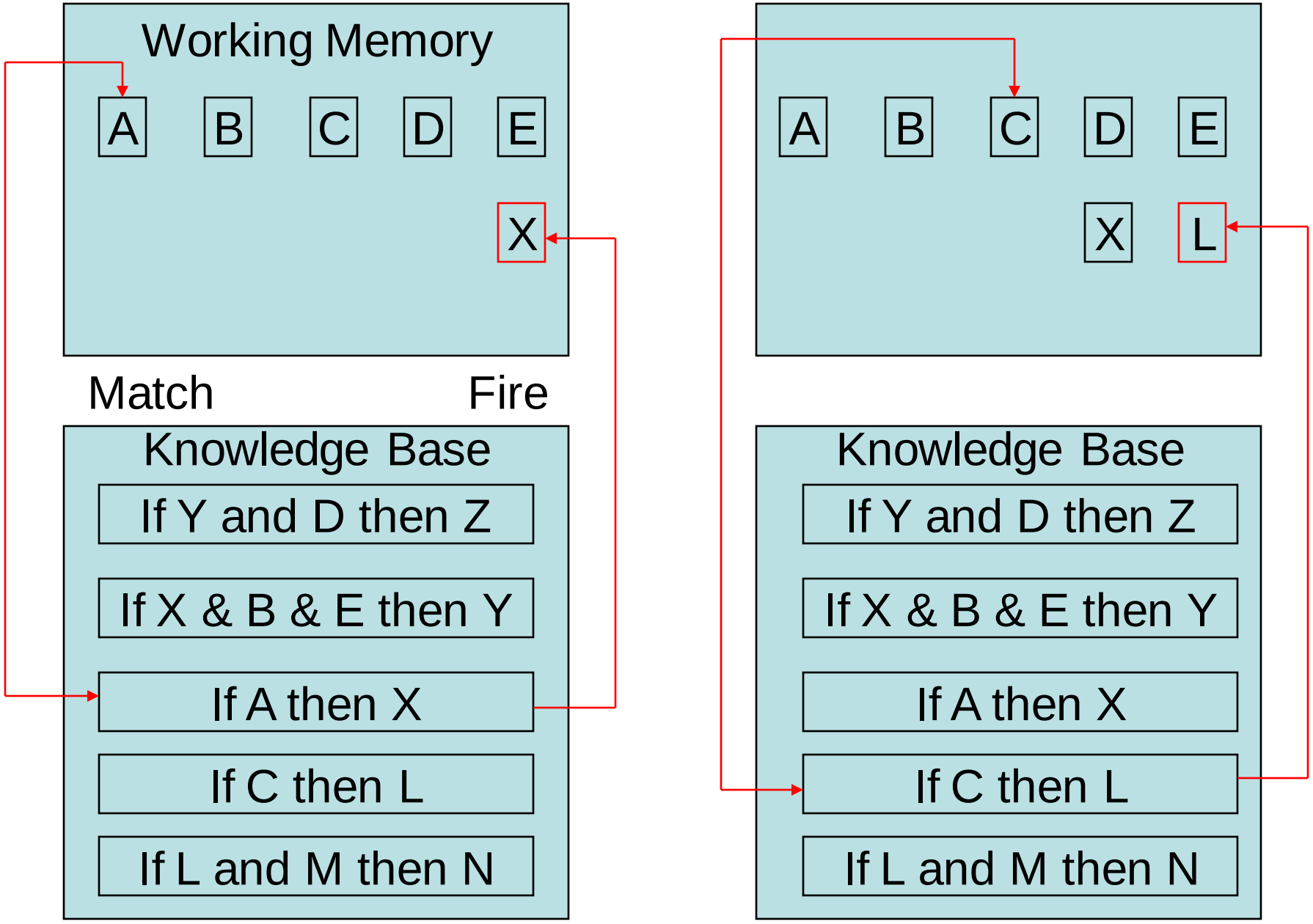
A Simple Example

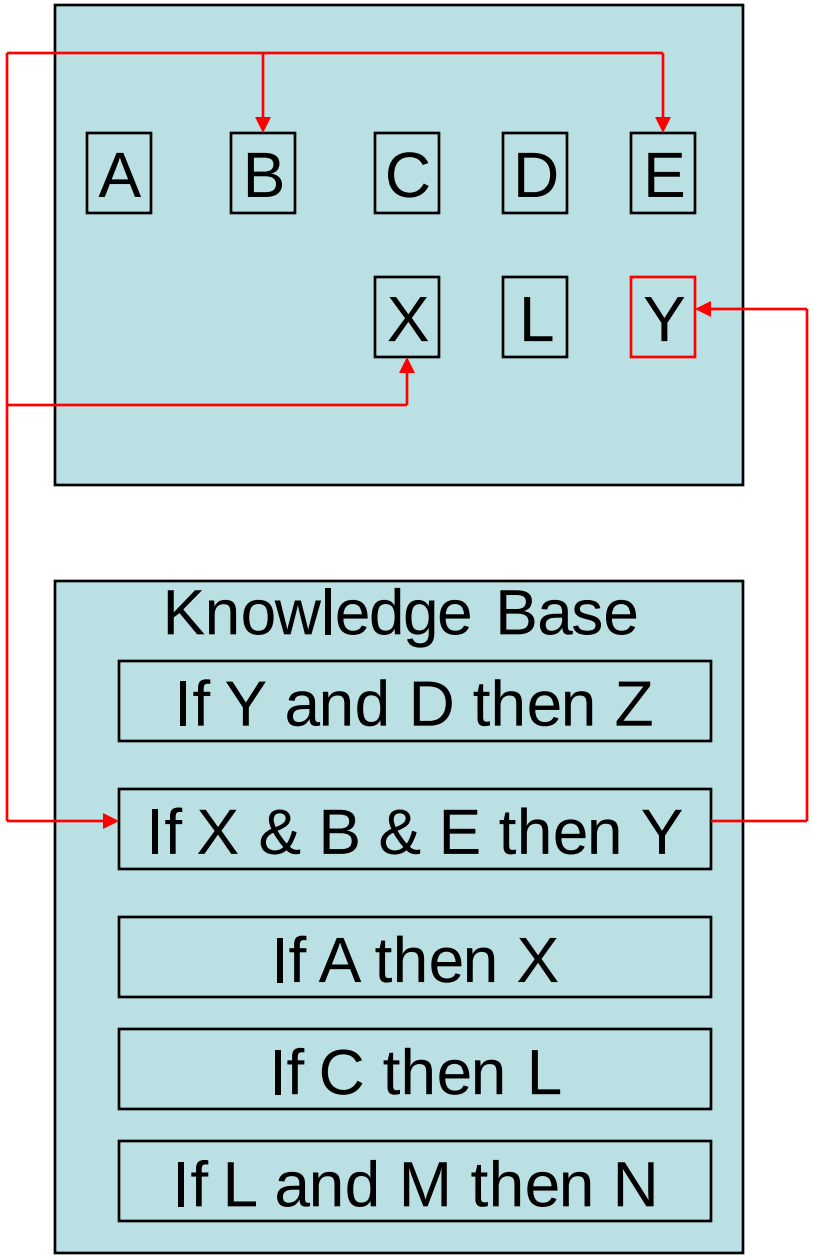
```
R1: IF hot AND smoky THEN fire
R2: IF alarm_beeps THEN smoky
R3: If fire THEN switch_on_sprinklers

F1: alarm_beeps     [Given]
F2: hot             [Given]
```

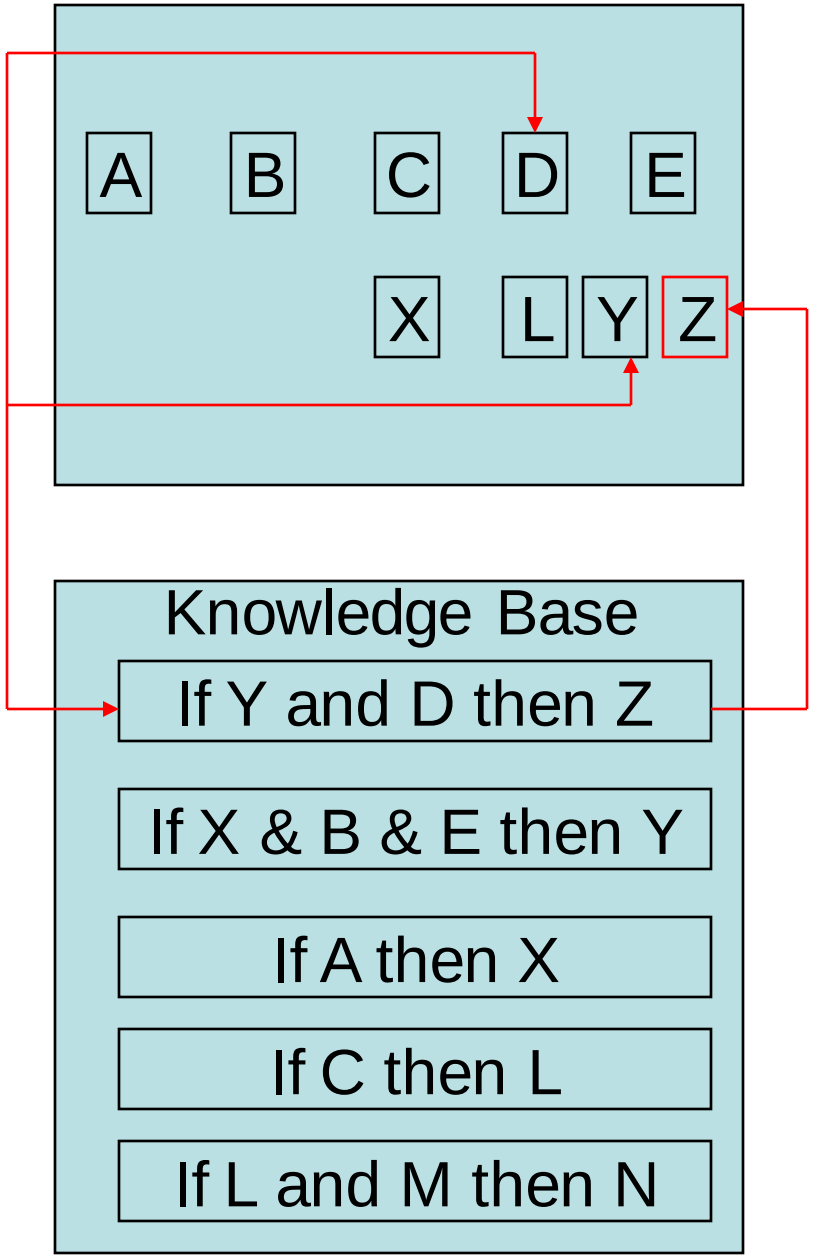
A Simple Example

```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers

F1: alarm_beeps     [Given]
F2: hot             [Given]

F3: smoky                       [from F1 by R2]
F4: fire                        [from F2, F3 by R1]
F5: switch_on_sprinklers        [from F4 by R3]
```

A typical Forward Chaining example


Working Memory
A
B
C
D
E
X
Match
Fire
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N
A
B
C
D
E
X
L
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N


A
B
C
D
E
X
L
Y
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N


A
B
C
D
E
X
L
Y
Z
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N

Forward Chaining Algorithm

Start from a set of facts (data available) and check to see if the premises of any rules are satisfied. If there is a match then the rule fires (is executed).
The steps followed in forward chaining are:

1. Matching: Compare rules with known facts and find rules that are satisfied.
2. Conflict Resolution: More than one rule may be satisfied. Conflict resolution is the process of selecting the one with highest priority for execution.
3. Execution: The rule selected is executed (fired). This may result in a new fact(s) to be added and the process continues forward.

Backward Reasoning

Backward Chaining

Goal Driven Reasoning

Backward chaining:→starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals.

- An inference engine using backward chaining would search the inference rules until it finds one which has a then-clause that matches a desired goal.
- If the if-clause of that inference rule is not known to be true, then it is added to the list of goals.

Backward Chaining

- Same rules/facts may be processed differently, using backward chaining interpreter
- Backward chaining means reasoning from goals back to facts.
  - The idea is that this focuses the search.
- Checking hypothesis
  - `Should I switch the sprinklers on?`

Backward Chaining Algorithm

- To prove goal G:
  - If G is in the initial facts, it is proven.
  - Otherwise, find a rule which can be used to conclude G, and try to prove each of that rule's conditions.

Example

Rules:

R1: IF hot AND smoky THEN fire

R2: IF alarm_beeps THEN smoky

R3: If fire THEN switch_on_sprinklers

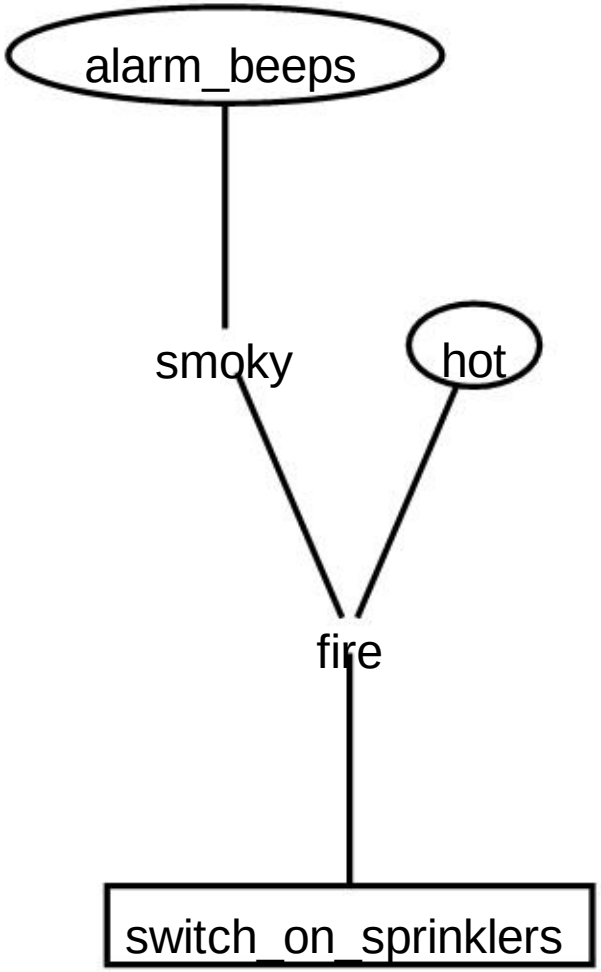
Facts:

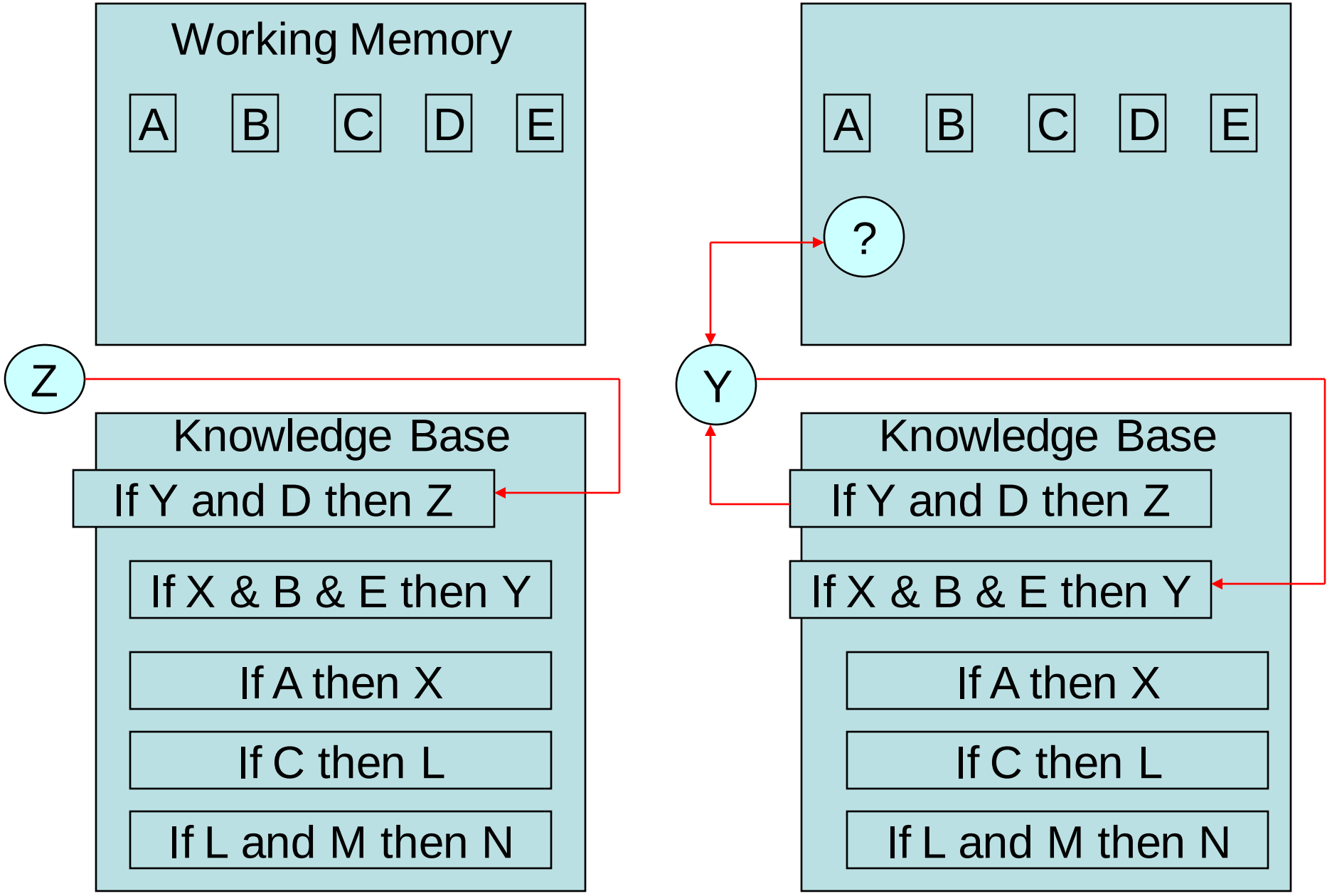
F1: hot

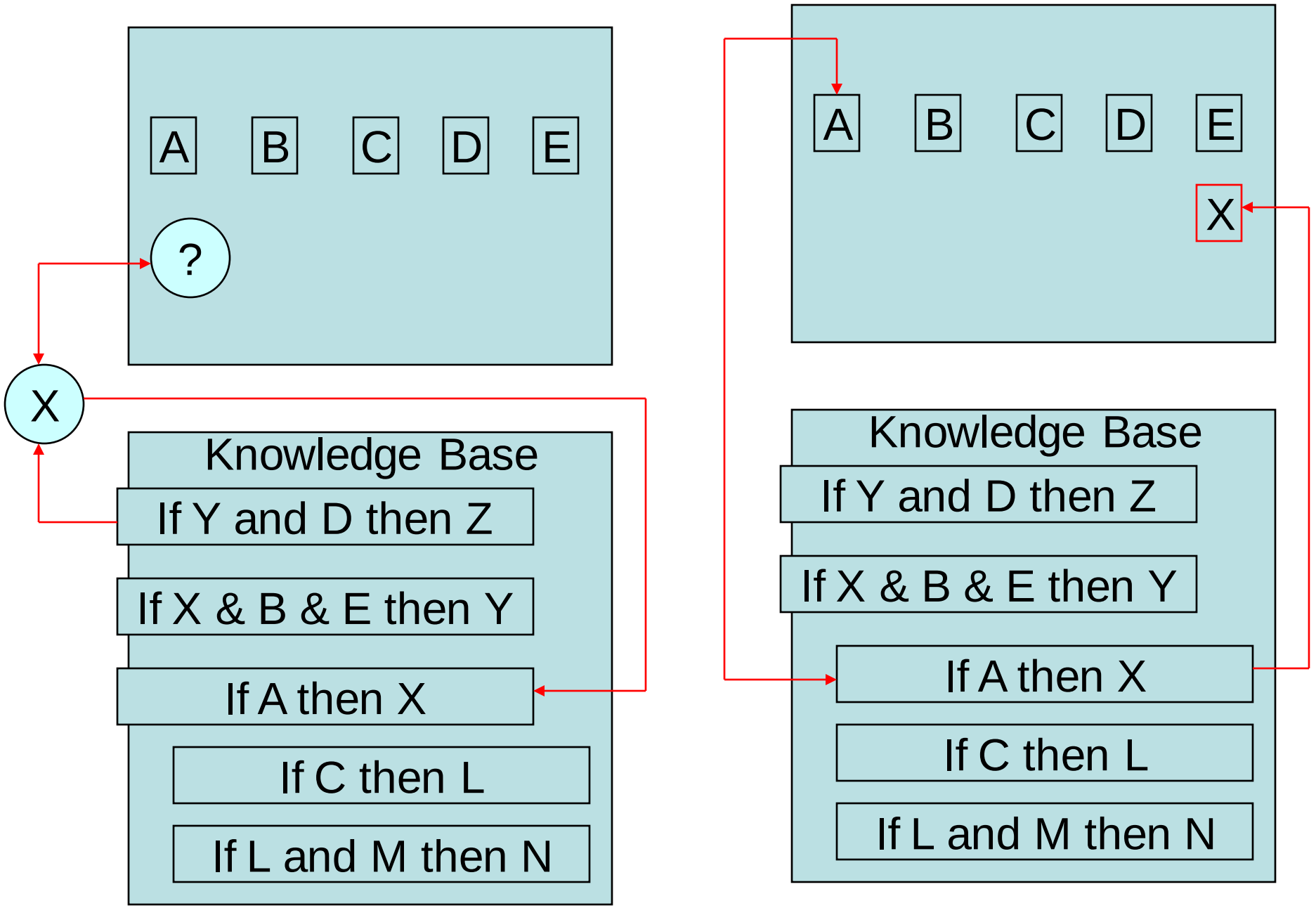
F2: alarm_beeps

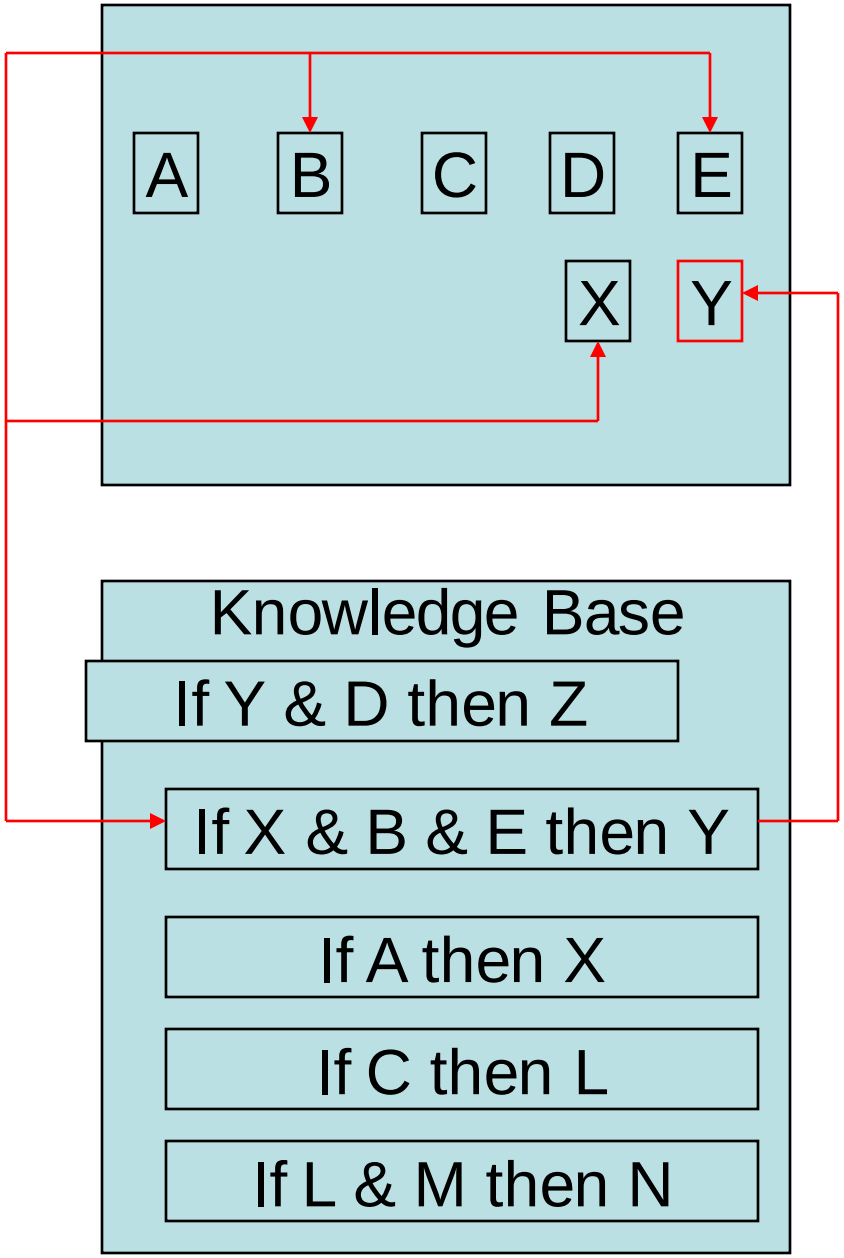
Goal:

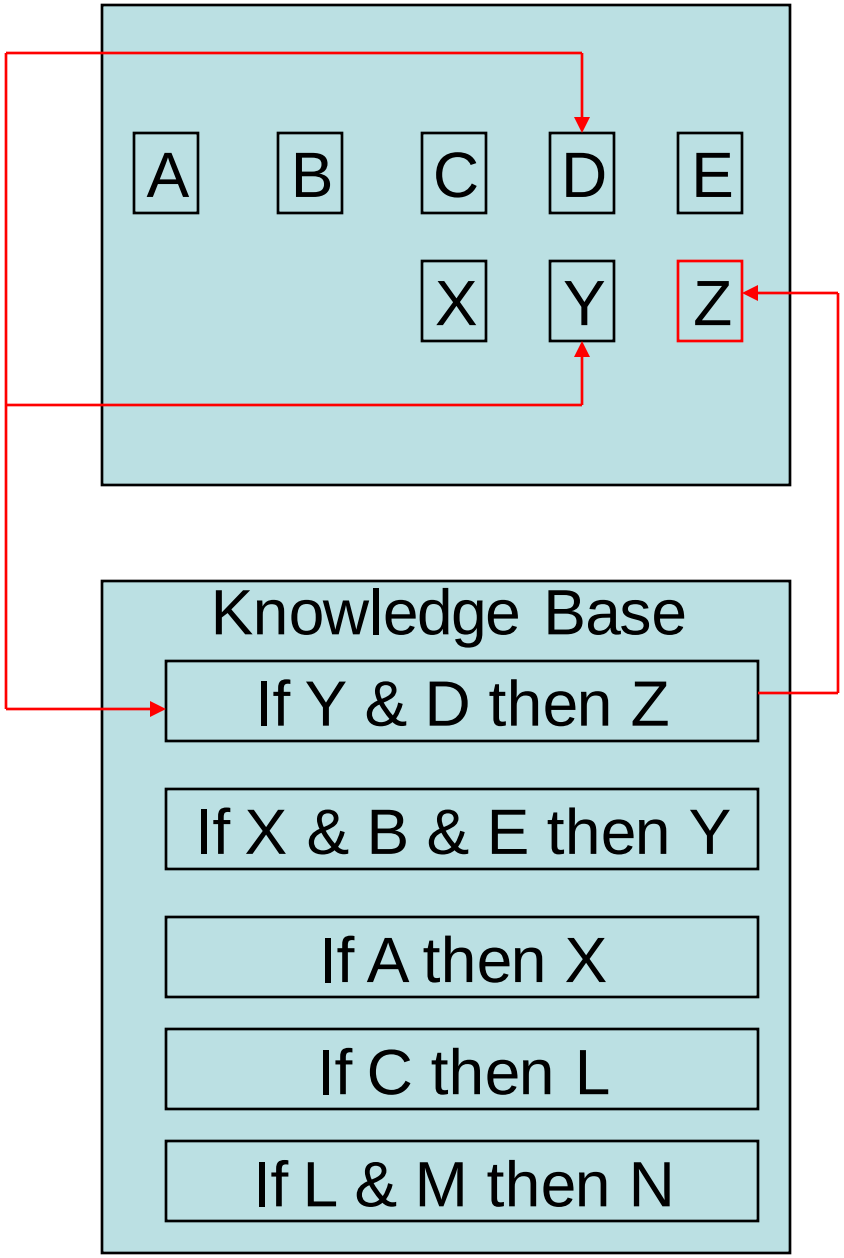
Should I switch sprinklers on?




Working Memory
A
B
C
D
E
Z
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N
A
B
C
D
E
?
Y
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N


A
B
C
D
E
?
X
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N
A
B
C
D
E
X
Knowledge Base
If Y and D then Z
If X & B & E then Y
If A then X
If C then L
If L and M then N


A
B
C
D
E
X
Y
Knowledge Base
If Y & D then Z
If X & B & E then Y
If A then X
If C then L
If L & M then N


A
B
C
D
E
X
Y
Z
Knowledge Base
If Y & D then Z
If X & B & E then Y
If A then X
If C then L
If L & M then N

Advantages of Rule Based Systems

- Modularity: Each rule is a separate unit. This makes adding, editing or removing of rules easily possible giving great flexibility to the system.

- Uniformity: The same format is used for representing all of the knowledge.

- Naturalness: In many domains rules are used to express the knowledge.

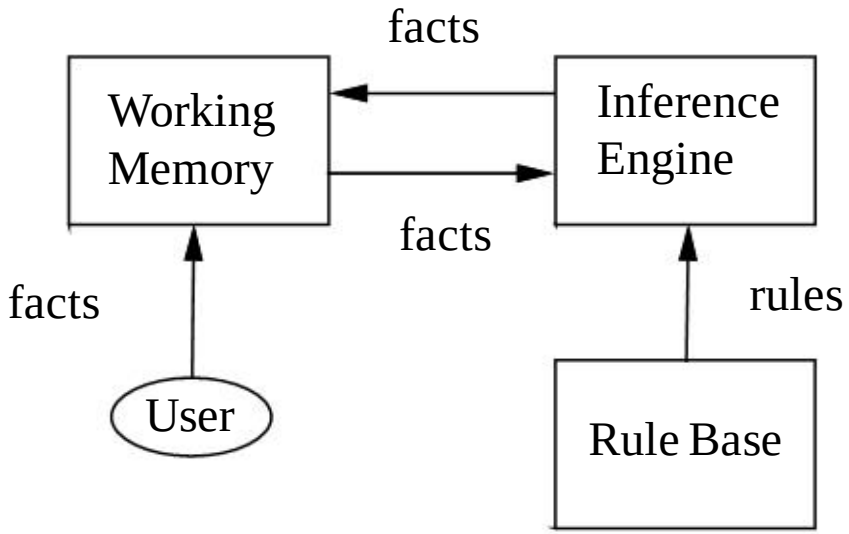
Disadvantages of Rule Based Systems

- Infinite Chaining
- Addition of new contradictory knowledge
- Modification of existing Knowledge
- Inefficiency
- Large number of rules needed to cover some domains (e.g. air traffic control)

Forward Chaining

In a forward chaining system: Facts are

- held in a working memory
- Condition-action rules represent actions to take when specified facts occur in working memory.
- Typically the actions involve adding or deleting facts from working memory.



Forward Chaining Algorithm (I)

Repeat

Collect the rule whose condition matches a fact in WM.

Do actions indicated by the rule

(add facts to WM or delete facts from WM)

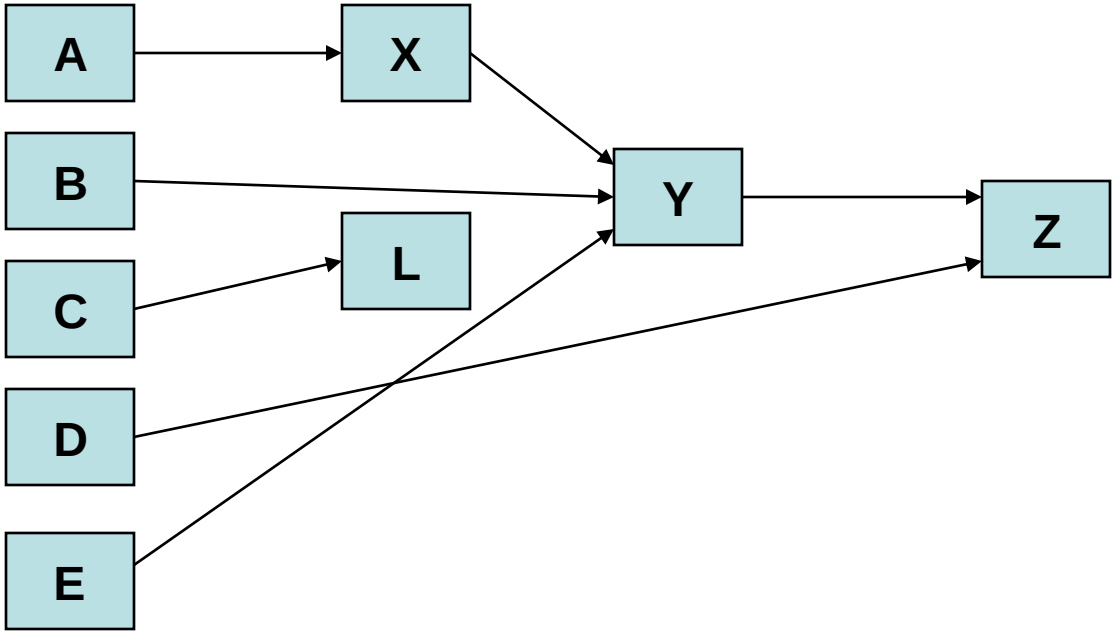
Until problem is solved or no condition match

Forward vs Backward Chaining

- Depends on problem, and on properties of rule set.
- If you have clear hypotheses, backward chaining is likely to be better.
  - Goal driven
  - Diagnostic problems or classification problems
    - Medical expert systems
- Forward chaining may be better if you have less clear hypothesis and want to see what can be concluded from current situation.
  - Data driven
  - Synthesis systems
    - Design / configuration

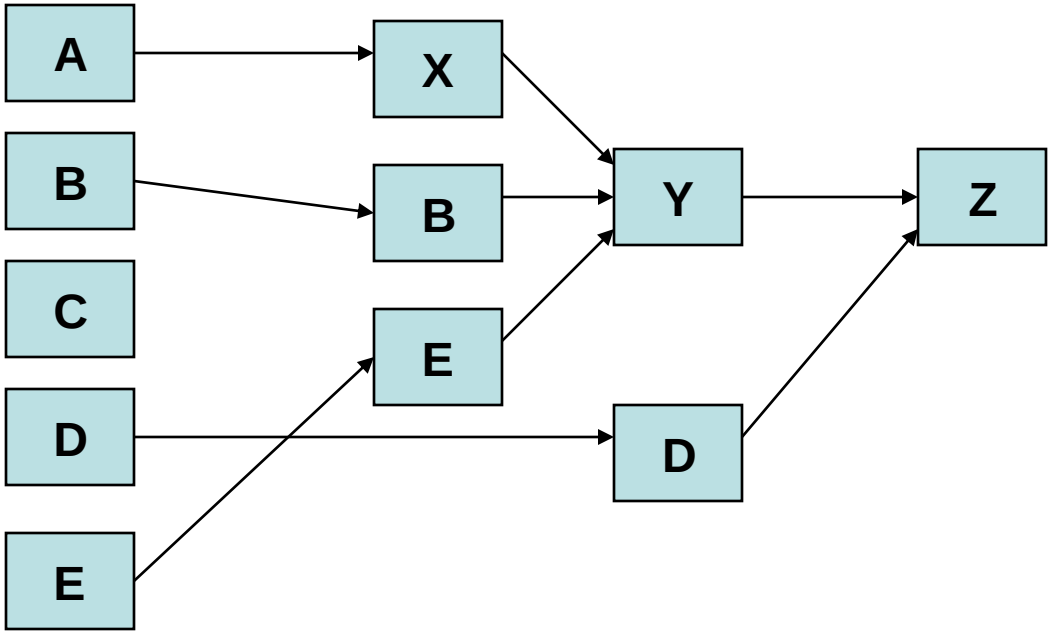
Forward-chaining Example (A,B,E, and D are given)

- If Y and D then Z
- If X and B and E then Y
- If A then X
- If C then L
- If L and M then N



Backward Chaining Example

- If Y and D then Z
- If X and B and E then Y
- If A then X
- If C then L
- If L and M then N



Natural Language Processing

What's Natural Language Processing?

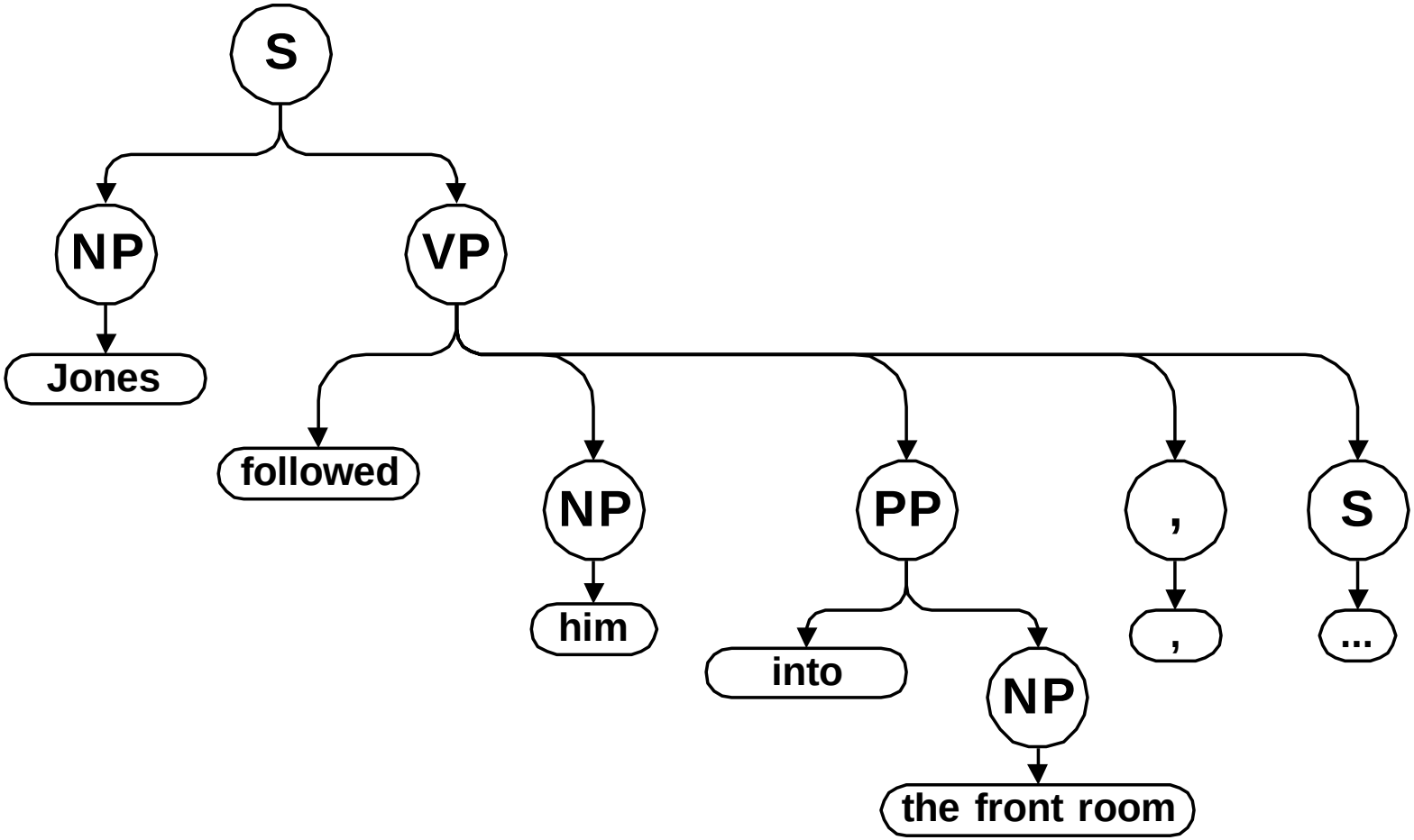
- Depends on your point of view
- *Psychology:* Understand human language processing
  - How do we learn language?
  - How do we understand language?
  - How do we produce language?
  - How is language tied to thought?
- *Engineering:* Build systems to process language
  - Build dialogue-based call centers
  - Build information retrieval engines
  - Build question-answering systems
  - Design general algorithms for a range of applications

Brief History of NLP

- 1950s: Machine Translation
  - Abandoned due to lack of computing resources
- 1960s: Despair
  - Problem considered impossible philosophically (Quine)
  - Problem considered impossible linguistically (Chomsky)
- 1970s: Dawn of Artificial Intelligence
  - "Full" dialog systems (eg. SHRDLU)
  - Symbolic processing in LISP
  - Early theories of semantics; first big systems (eg. LUN
  - First information retrieval systems

W.V.O. Quine

“Standard” Parse Tree Notation



NLP…

- Phrase Structure Rules
  - S → NP VP S → V NP PP
  - S → NP VP PP NP → N DET PP
  - NP → Det N or ART N
  - VP → V Adj VP → V NP
  - VP→ V PP
  - PP → P NP
  - VP → V NP PP VP → AUX V NP DET → ART ADJ
- Lexical Entries
  - N → book, cow, course, … V → printed, want …
  - ART → the, a, …
  - AUX → was, were …
  - PREP → by, on, with …
  - ADJ → short, long, fast …

Noam Chomsky

Problem reduction methods

- I want to be a famous musician
  - Learn to sing
  - Learn to play the guitar
  - Learn to play the bass
  - Learn to play drums
- If I want to play the guitar what do I do?
  - Buy a guitar
  - Take lessons
  - Practice

Problem reduction methods