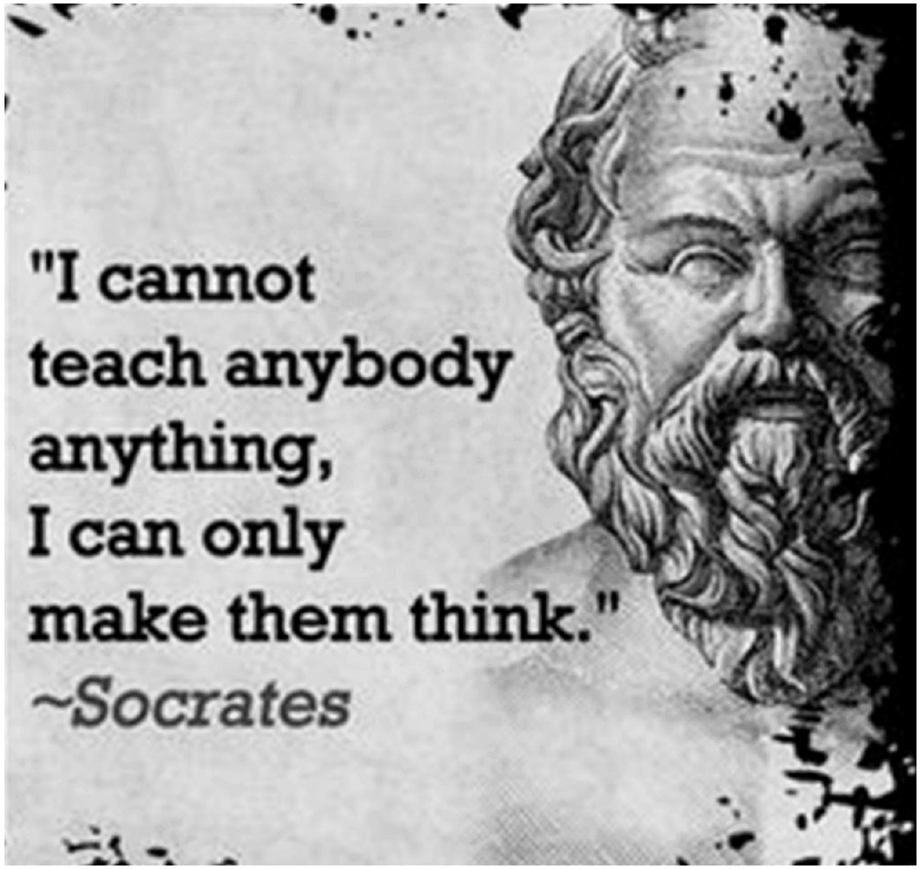
Artificial intelligence CC-721

Propositional Logic


"I cannot teach anybody anything, I can only make them think."
~Socrates

Knowledge-Based Systems

- A knowledge-based system is a system consisting of the following:
  - 1. A data set called the knowledge that contains knowledge about the domain of interest.
  - 2. An inference engine that processes the knowledge to solve a problem.
- Often knowledge-based systems are expert systems, which are systems that make the judgment or decisions of an expert.
- A system that performs medical diagnosis and possibly recommends treatment options or further testing is a medical expert system.

Example

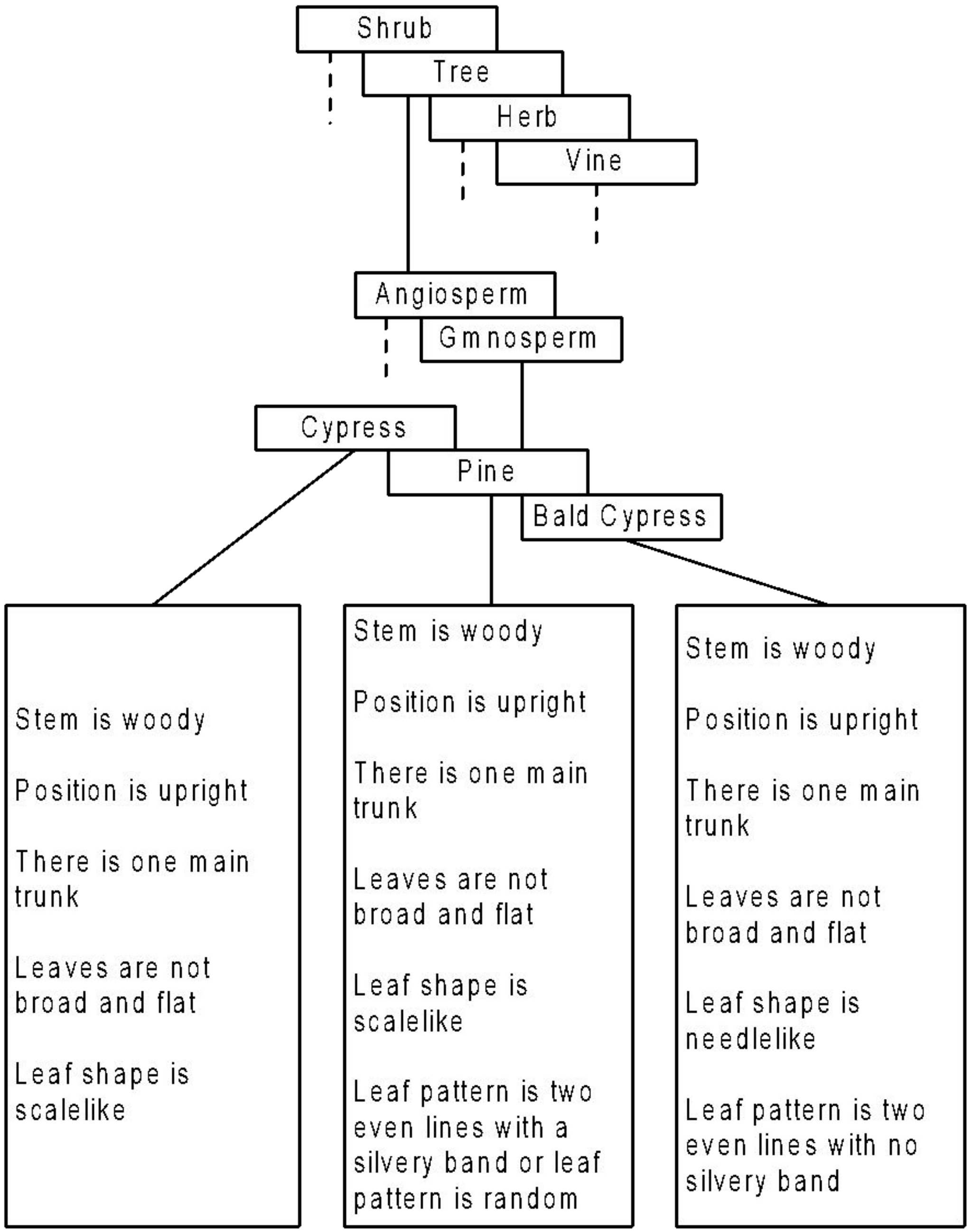
- Suppose a botanist is trying to identify a plant based on information supplied to her over the Internet by her colleague.

- Because she cannot see the plant, she must ask the colleague questions in order to obtain the facts need to identify the plant.

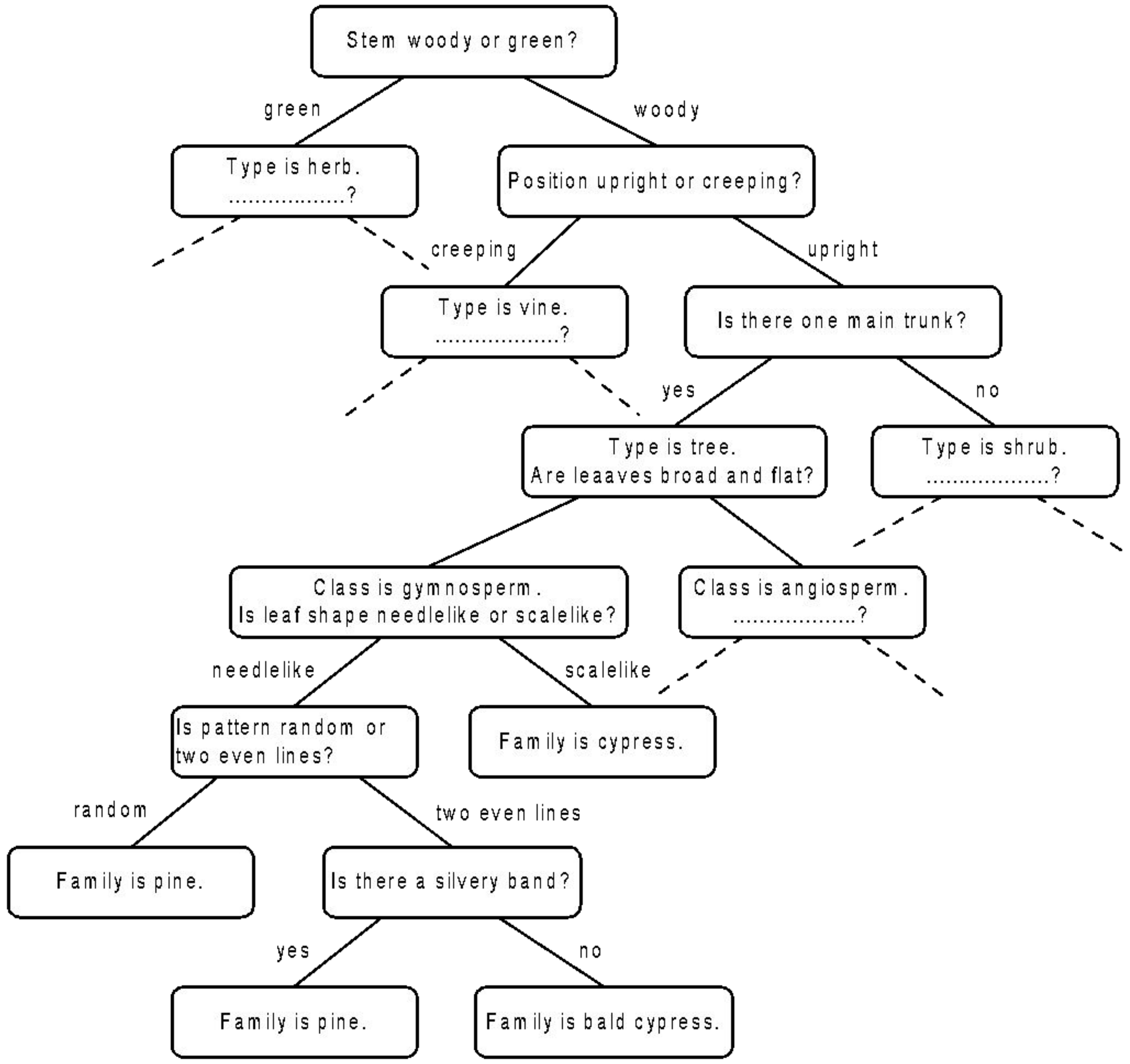
- Plants are classified according to type, class, and family.

- Within a type there are many classes, and within a class there are many families.

A portion of a Tree for Plant Classification



A decision tree for plant classification


Stem woody or green?
green
woody
Type is herb. ....?
Position upright or creeping?
creeping
upright
Type is vine. ....?
Is there one main trunk?
yes
no
Type is tree. Are leaaves broad and flat?
Type is shrub. ....?
Class is gymnosperm. Is leaf shape needlelike or scalelike?
Class is angiosperm. ....?
needlelike
scalelike
Is pattern random or two even lines?
Family is cypress.
random
two even lines
Family is pine.
Is there a silvery band?
yes
no
Family is pine.
Family is bald cypress.

Problems with Decision Trees

- There are two problems with the decision tree approach.
- First, experts systems are often developed by eliciting information or knowledge from an expert.
- It may be difficult for a botanist, for example, to identify the entire decision tree from the top down.
- Second, if the entire tree was developed, if a mistake were made and information needed to be added or deleted, it would be necessary to break edges in the tree and perhaps add new ones.
- So, the tree would need to be reconstructed each time it was necessary to change the knowledge in the system.

- It seems that experts may have their knowledge locally organized among closely connected attributes.

- For example, a botanist might know that all plants that have woody stems, stand upright, and have one main trunk are trees.

We can express this knowledge as follows:

IF stem is woody
AND position is upright
AND there is one main trunk
THEN type is tree.

The preceding is an item of knowledge expressed as an **IF-THEN rule**.

Often it is easier for a knowledge engineer to elicit localized rules such as this from an expert.

Our botanist might also know that all trees that have broad and flat leaves are gymnosperms.

The rule representing this knowledge is

```
IF    type is tree
AND    leaves are broad and flat
THEN  class is gymnosperm.
```

Portion of a Knowledge Base for Plant Classification

1.	IF	class is gymnosperm
	AND	leaf shape is scalelike
	THEN	family is cypress.
2.	IF	class is gymnosperm
	AND	leaf shape is needlelike
	AND	pattern is random
	THEN	family is pine.
3.	IF	class is gymnosperm
	AND	leaf shape is needlelike
	AND	pattern is two even lines
	AND	silvery band is yes
	THEN	family is pine.
4.	IF	class is gymnosperm
	AND	leaf shape is needlelike
	AND	pattern is two even lines
	AND	silvery band is no
	THEN	family is bald cypress.
5.	IF	type is tree
	AND	broad and flat is yes
6.	IF	type is tree
	AND	broad and flat is no
	THEN	class is gymnosperm.
7.	IF	stem is green
	THEN	type is herb.
8.	IF	stem is woody
	AND	position is creeping
	THEN	type is vine.
9.	IF	stem is woody
	AND	position is upright
	AND	one main trunk is yes
	THEN	type is tree.
10.	IF	stem is woody
	AND	position is upright
	AND	one main trunk is no
	THEN	type is shrub.

- Those 10 rules contain all the knowledge in the decision tree we presented.
- However, the knowledge base is simply a collection of separate items of knowledge.
- It does not entail any particular way for using this knowledge.
- A mechanism that exploits the knowledge to solve a problem is called an **inference engine**.

Backward Chaining Inference Engine

- The backward chaining algorithm processes the rule base in such a way that questions are asked of the user in the exact same way as a decision tree.

- The algorithm essentially builds the decision tree on the fly.

How Backchain Works

1. The user enters her *goal*, which in his case would be *family.*
2. Backchain cycles through the rule base, looking for a rule whose conclusion is *family*.
3. Rule 1 is the first such rule.
4. If Backchain can learn that the premises in Rule 1 are true, it can conclude the *family* is *cypress*.
5. Backchain tries to determine if the first premise in Rule 1 is true by recursively going back to Step 1 and making *class* its new *goal*.

1. The current *goal* is *class.*
2. Backchain cycles through the rule base, looking for a rule whose conclusion is *class*.
3. Rule 5 is the first such rule.
4. If Backchain can learn that the premises in Rule 5 are true, it can conclude the *class* is *angiosperm*.
5. Backchain tries to determine if the first premise in Rule 5 is true by recursively going back to Step 1 and making *type* its new *goal*.

1. The current *goal* is *type.*
2. Backchain cycles through the rule base, looking for a rule whose conclusion is *type*.
3. Rule 7 is the first such rule.
4. If Backchain can learn that the premises in Rule 7 are true, it can conclude the *type* is *herb*.
5. Backchain tries to determine if the first premise in Rule 7 is true by recursively going back to Step 1 and making *stem* its new *goal*.

1. The current *goal* is *stem.*
2. Backchain cycles through the rule base, looking for a rule whose conclusion is *stem*.
3. There is no such rule.
4. So, Backchain asks the user the value of *stem*.

Note that this is the first question asked of the user by the decision tree.

Backchain continues in this manner, resulting in the decision being built on the fly.

Algorithm 2.3 Backward_Chaining

```
Input: The user's goal Goal and information requested of the user.
Output: The value of the user's goal if it can be determined; otherwise "unknown."

Function Backchain(Goal);
var Assertion, Rule, Next_Goal, Premise, True_Assertion,
    All_Premises_True;

if there is an assertion in Assertion_List with Goal as its attribute
    Assertion = that assertion;
else
    Assertion = Null;
if Assertion = Null
    Rule = first rule;
    while Assertion = Null and there are more rules
        if Goal = attribute in the conclusion of Rule
            All_Premises_True = False;
            Premise = first premise in the antecedent of Rule;
            repeat
                Next_Goal = attribute in Premise;
                True_Assertion = Backchain(Next_Goal);
                if Premise = True_Assertion
                    Premise = next premise in the antecedent of Rule;
                    if Premise = Null
                        All_Premises_True = True;
                endif
            until All_Premises_True or Premise ≠ True_Assertion;
            if All_Premises_True
                Assertion = conclusion of Rule;
        endif
        Rule = next rule;
    endwhile
    if Assertion = Null
        prompt user for value of Goal;
        read Value;
        if Value ≠ Null
            Assertion = Goal "is" Value;
        else
            Assertion = Goal "is unknown";
    endif
    add Assertion to Assertion_List;
endif
if Goal = User_Goal
    if Assertion = Goal "is unknown"
        write "You cannot answer enough questions to determine " Goal;
    else
        write Assertion;
endif
return Assertion;
```

The top level call to Backchain is as follows:

```
$Empty(Assertion\_List)$;          // Make the $Assertion\_List$ empty.
write Enter your goal.;
read $User\_Goal$ ;
$Assertion = Backchain(User\_Goal)$;
```

Forward Chaining Inference Engine

- Suppose we have a list of true assertions:
  - The stem is woody.
  - The position is upright.
  - There is one main trunk.
  - The leaves are not broad and flat.
- Our goal is to conclude all we can from them using the knowledge base.

The Forward Chaining Algorithm

Algorithm: Forward_Chaining
Input: A set *Assertion_List* of true assertions.
Output: The set *Assertion_List* with all assertions that can be deduced by applying the rules to the input added.

```
Procedure Forward_Chain (var Assertion_List);
var Rule;
Rule = first rule;
while there are more rules
    If all premises in Rule are in Assertion_List
    and conclusion in Rule is not in Assertion_List
        add conclusion of Rule to Assertion_List;
        Rule = first rule;
    else
        Rule = next rule;
endwhile
```

- Forward chaining can be made more efficient by sorting the rules before doing forward chaining.
- The sorting scheme is as follows. If rule A's conclusion is a premise in Rule B's antecedent, then we place rule A before rule B.
- With the rules sorted in this manner, there is no need to return to the first rule when a conclusion is added to the true assertion list.

Using Forward and Backward Chaining in a Diagnostic System

- **Diagnosis** is the process of determining or analyzing the cause or nature of a problem.
- A **diagnostic system** is one that performs diagnosis.
- The classical example of diagnosis is medical diagnosis in which we are trying to determine the disease that is causing some manifestations.
- Another example is an auto mechanic trying to diagnose the problem with an automobile.

We illustrate a rule-based diagnostic system using automobile problem diagnosis. We have these rules:

1.	IF	the car does not start
	AND	the engine does not turn over
	AND	and the lights do not come on
	THEN	the problem is battery.
2.	IF	the car does not start
	AND	the engine does turn over
	AND	and the engine is getting enough gas
	THEN	the problem is spark plugs.
3.	IF	the car does not start
	AND	the engine does not turn over
	AND	the lights do come on
	THEN	the problem is the starter motor.
4.	IF	there is gas in the fuel tank
	AND	there is gas in the carburetor
	THEN	the engine is getting enough gas.

Using Forward Chaining

- Suppose Melissa observes the following facts about her car:
  - The car does not start.
  - The engine does turn over.
  - There is gas in the fuel tank.
  - There is gas in the carburetor.
- Using forward chaining, rule 4 will trigger first because both its premises are true.
- We conclude that the engine is getting gas.
- Now all three premises in rule 2 are true. So it will trigger.
- We conclude that the problem is spark plugs.

Using Backward Chaining

- Suppose Melissa only knows the following:
  - The car does not start.
  - The engine does turn over.
- We can use backward chaining starting with each rule that concludes a problem.
- Rule 1 is tried first, but it does not trigger because a premise is false.
- Rule 2 is tried next. Its first two premises are true, and there is a rule (rule 4) that concludes its third premise.
- We backchain to rule 4 and check its premises. Neither of them is true and there is no rule for either of them.
- So we now prompt the user for their values. Melissa now knows to check these matters because she is prompted. Suppose she observes the following:
  - There is gas in the fuel tank.
  - there is gas in the carburetor.
- Rule 4 now triggers and we conclude “The engine is getting enough gas.”
- We now return to rule 2 and conclude that “The problem is spark plugs.”

- An actual production system may use both forward chaining and backward chaining in turn.
- That is, it could first use forward chaining to conclude all that it can based on the user's initial knowledge.
- If that is not enough to make a diagnosis, it could then use goal-driven backward chaining.

Using Forward Chaining in a Configuration System

- A **configuration system** is one that arranges parts into some whole.
- A system that arranges grocery items in grocery bags is a configuration system.

Rules for bagging groceries:

1.	IF	step is Bag_large_items
	AND	there is a large item to be bagged
	AND	there is a large bottle to be bagged
	AND	there is a bag with <6 items
	THEN	put the large bottle in the bag.
2.	IF	step is Bag_large_items
	AND	there is a large item to be bagged
	AND	there is a bag with <6 items
	THEN	put the large item in the bag.
3.	IF	step is Bag_large_items
	AND	there is a large item to be bagged
	THEN	start a fresh bag.
4.	IF	step is Bag_large_items
	THEN	step is Bag_medium_items.
5.	IF	step is Bag_medium_items
	AND	there is a medium item to be bagged
	AND	there is a bag with <10 medium items
	AND	that bag contains 0 large items
	AND	the medium item is frozen
	AND	the medium item is not in an insulated bag
	THEN	put the medium item in an insulated bag.
6.	IF	step is Bag_medium_items
	AND	there is a medium item to be bagged
	AND	there is a bag with <10 medium items
	AND	that bag contains 0 large items
	THEN	put the medium item in the bag.
7.	IF	step is Bag_medium_items
	AND	there is a medium item to be bagged
	THEN	start a fresh bag.
8.	IF	step is Bag_medium_items
	THEN	step is Bag_small_items.
9.	IF	step is Bag_small_items
	AND	there is a small item to be bagged
	AND	there is a bag that is not full
	THEN	put the small item in the bag.
10.	IF	step is Bag_small_items
	AND	there is a small item to be bagged
	THEN	start a fresh bag.
11.	IF	step is Bag_small_items
	THEN	halt.

We apply forward chaining to the rules to bag a set of items:

Item	Container	Size	Frozen	Bagged?
Soda	Bottle	Large	No	No
Bread	Bag	Medium	No	No
Ice cream	Carton	Medium	Yes	No
Detergent	Box	Large	No	No
Eggs	Carton	Small	No	No
Popsicles	Insulated Bag	Medium	Yes	No

Large items are bagged first, medium items next, and small items last.

We assure that large items are bagged first by initially setting

Step = Bag_large_items;

When bagging the items we do **conflict resolution**.

Specificity Ordering. *If the set of premises of one rule are a superset of the set of premises of another rule, then the first rule is triggered on the assumption that it is more specialized to the current situation.*

In this way large bottles get bagged ahead of other large items.

We can implement specificity ordering by sorting the rules so that the rule with more premises appears first.

Context Limiting. *Separate the rules into disjoint subsets.*
Only the rules in one subset are active at any particular time.
The context is changed to a new context by a rule in the current context.

When all large items are bagged,
we change the context from Bag_large_items to Bag_medium_items.

When all medium items are bagged,
we change the context from Bag_medium_items to Bag_small_items.