PRODUCTION RULES

Reference: Bratko ed. 3, chapter 15, page 347-

Aim:
To describe the systems that represent knowledge in the form of rules. Rule-based systems normally use a *working memory* that initially contains the input data for a particular run, and an *inference engine* to find applicable rules and apply them.
Keywords: backward chaining, condition-action rule, conflict resolution, expert system, fire, forward chaining, inference engine, match-resolve-act cycle, ripple-down rules, rule-based system, working memory
Plan:
• condition-action rules can represent knowledge • backward and forward chaining • rules, facts, working memory, inference engine • match-resolve-act cycle: conflict resolution strategies • BAGGER example system

Introduction

"Production" in the title of these notes (or "production rule") is a synonym for "rule", i.e. for a condition-action rule (see below). The term seems to have originated with the term used for rewriting rules in the Chomsky hierarchy of grammar types, where for example context-free grammar rules are sometimes referred to as context-free productions.

Rules

These are also called **condition-action rules**.
These components of a rule-based system have the form:

```
if <condition> then <conclusion>
```

or

```
if <condition> then <action>
```

Example:
if patient has high levels of the enzyme ferritin in their blood
and patient has the Cys282→Tyr mutation in HFE gene
then conclude patient has haemochromatosis*

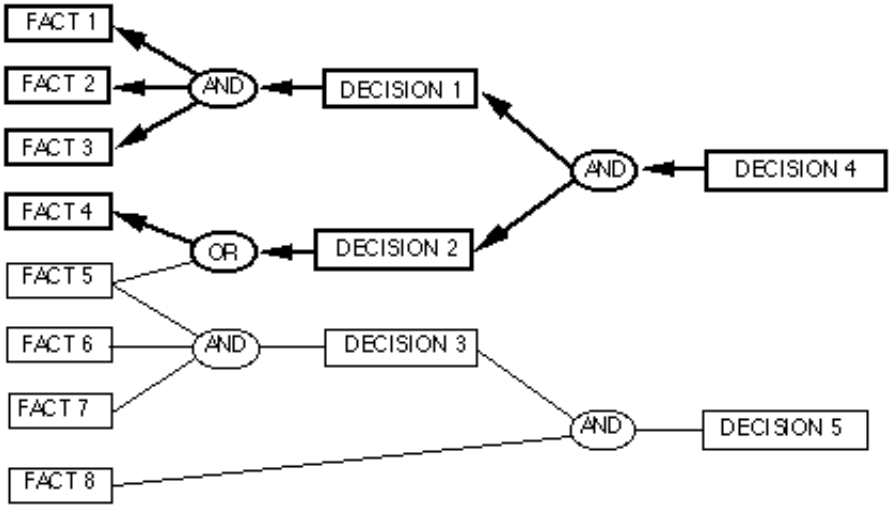
* medical validity of this rule is not asserted here

Rules can be evaluated by:

- backward chaining
- forward chaining

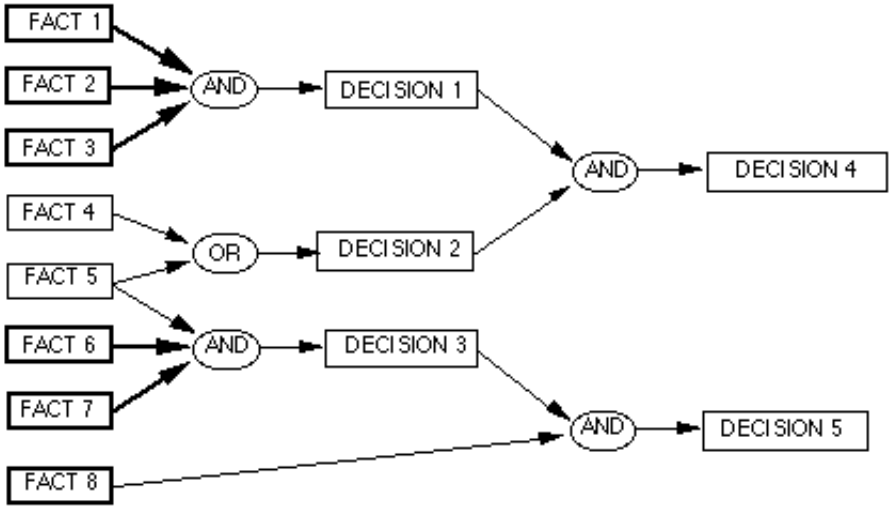
Backward Chaining

- To determine if a decision should be made, work backwards looking for justifications for the decision.
- Eventually, a decision must be justified by facts.



Forward Chaining

- Given some facts, work forward through inference net.
- Discovers what conclusions can be derived from data.



Forward Chaining 2

Until a problem is solved or no rule's 'if' part is satisfied by the current situation:

1. Collect rules whose 'if' parts are satisfied.

2. If more than one rule's 'if' part is satisfied, use a conflict resolution strategy to eliminate all but one.
3. Do what the rule's 'then' part says to do.

Production Rules

A production rule system consists of

- a set of rules
- working memory that stores temporary data
- a forward chaining inference engine

Match-Resolve-Act Cycle

The match-resolve-act cycle is what the inference engine does.

loop

match conditions of rules with contents of working memory
if no rule matches **then** stop
resolve conflicts
act (i.e. perform conclusion part of rule)

end loop

BAGGER

- Bagger is a simple rule-based system that describes how to pack items at a supermarket check-out.
- While explaining Bagger, we shall describe a number of potential strategies for conflict resolution.
- Bagger's working memory has an associated table of attributes of the objects (stock items) at the supermarket.
- There are 4 steps in Bagger, and Bagger uses a Working Memory item called "Step" to keep track of where it is up to.
- Each rule checks the value of "Step" as part of its **if** part, and will be applicable only to one of the four steps.
- This makes it easier to be sure that the rules will not interact in unexpected ways (a pitfall in creating rule-based systems).

Steps in Bagger

1. **Check order**: Check what the customer has selected; look to see if something is missing, suggest

additions.
2. **Pack large items**:Put the large items in the bag; put big bottles first.
3. **Pack medium items**:Put in the medium sized items; put frozen food in plastic bags.
4. **Pack small items**:Put in the small items wherever there is room.

Working Memory

```
Step:      Check order
Bag1:      <empty>
Unpacked:  Bread
           Glop
           Granola (2)
           Ice cream
           Chips
```

Attributes of Objects

ITEM	CONTAINER TYPE	SIZE	FROZEN?
Bread	Plastic bag	Medium	No
Glop	Jar	Small	No
Granola	Cardboard box	Large	No
Ice cream	Cardboard carton	Medium	Yes
Pepsi	Bottle	Large	No
Chips	Plastic bag	Medium	No

Rules for Step 1

```
B1:
if      the step is check-order
and     there is a bag of chips
and     there is no soft-drink bottle
then    add one bottle of soft drink to the order

B2:
if      the step is check-order
then    discontinue the check-order step
and     start the pack-large-items step
```

Which of these rules should be chosen when in the check order step?

Conflict Resolution

Specificity Ordering

If a rule's condition part is a superset of another, use the first rule since it is more specialised for the current task.

Rule Ordering
: Choose the first rule in the text, ordered top-to-bottom.

Data Ordering
: Arrange the data in a priority list. Choose the rule that applies to data that have the highest priority.

Size Ordering
: Choose the rule that has the largest number of conditions.

Conflict Resolution continued

Recency Ordering
: The most recently used rule has highest priority *or*
the least recently used rule has highest priority *or*
the most recently used datum has highest priority *or*
the least recently used datum has highest priority.
More details

Context Limiting
: Reduce the likelihood of conflict by separating the rules into groups, only some of which are active at any one time. Have a procedure that activates and deactivates groups.

Rules for Step 2

```
B3:
if      the step is pack-large-items
and     there is a large item to be packed
and     there is a large bottle to be packed
and     there is a bag with < 6 large items
then    put the bottle into the bag

B4:
if      the step is pack-large-items
and     there is a large item to be packed
and     there is a bag with < 6 large items
then    put the large item into the bag

B5:
if      the step is pack-large-items
and     there is a large item to be packed
then    get a new bag
```

Working Memory So Far

```
Step:           pack-medium-items
Bag1:           Pepsi
                Granola (2)
```

```
Unpacked:       Bread
                Glop
                Ice cream
                Chips
```

Rules for Step 3

```
B7:
if      the step is pack-medium-items
and     there is a medium item to be packed
and     there is an empty bag or a bag with medium items
and     the bag is not yet full
and     the medium item is frozen
and     the medium item is not in a freezer bag
then    put the medium item in a freezer bag

B8:
if      the step is pack-medium-items
and     there is a medium item to be packed
and     there is an empty bag or a bag with medium items
and     the bag is not yet full
then    put the medium item in the bag

B9:
if      the step is pack-medium-items
and     there is a medium item to be packed
then    get a new bag

B10:
if      the step is pack-medium-items
then    discontinue the pack-medium-items step
and     start the pack-small-items step
```

Working Memory So Far

```
Step:           pack-small-items
Bag1:           Pepsi
                Granola (2)
Bag2:           Bread
                Ice cream (in freezer bag)
                Chips
Unpacked:       Glop
```

Rules for Step 4

```
B11:
```

```
if      the step is pack-small-items
and     there is a small item to be packed
and     the bag is not yet full
and     the bag does not contain bottles
then    put the small item in the bag

B12:
if      the step is pack-small-items
and     there is a small item to be packed
and     the bag is not yet full
then    put the small item in the bag

B13:
if      the step is pack-small-items
and     there is a small item to be packed
then    get a new bag

B14:
if      the step is pack-small-items
then    discontinue the pack-small-items step
and     stop
```

Implementing Rules in Prolog

Click here to see Prolog code for a simple production rule system.

To use this code, copy it to your own directory, e.g. by

```
% cd
% cp ~cs9414/public_html/Examples/rules-swi.pro ~
```

then start Prolog and do the following dialogue:

Dialogue with rules-swi.pro

```
% prolog rules-swi.pro
?- wm(X).
```

[Prolog will tell you which facts it knows (a, b, and c). Don't forget to type ";" after each solution is produced by Prolog.]

```
?- run.
Yes
?- wm(X).
```

[The answer tells you that Prolog still knows that a, b, and c are true, and also which other "facts" it now

knows. Don't forget the ";"s.]

```
?- already_fired(X, Y).
```

[This time the answer tells you that two rules have fired, and gives their names and their conditions. A rule with the name `null` is also mentioned - this is a workaround in the code to avoid Prolog complaining that `already_fired` is undefined, in cases where no rules have yet been fired.]

You can also play with the code - e.g. by writing your own rules and facts, and running the system with them.

Scaling Up

Two problems that became apparent in attempts at commercial use of rule-based systems were:

1. stopping the rules from interacting with each other in unexpected ways as the number of rules grew large;
2. maintenance: adding extra rules to correct undesired behaviour (or deal with unusual cases) without messing up the behaviour of the rest of the system.

A partial solution to the first problem is to use **partitioned production systems** where at any given time only a subset of the rules are active, so someone building a system can concentrate on just those rules, and hopefully understand their interactions. The Bagger system is effectively a partitioned production system, with the value of `step` determining which rules are active at any given time.

The best known approach to the maintenance problem (and it also deals with the interaction problem) is to use **Ripple Down Rules (RDRs)**. If you plan to use rule-based systems in a large-scale application, you should spend some time reading up on RDRs.

You can find material on RDRs at http://www.cse.unsw.edu.au/~cs9416/06s1/lectures/rdr/RDR_links.html and Paul Compton's Home Page/ - the UNSW course COMP9416 Rule-based Systems covers these in more detail, but may not be offered every year.

Summary: Rule-Based Systems

Rule-based systems consist of a set of rules, a working memory and an inference engine. The rules encode domain knowledge as simple condition-action pairs. The working memory initially represents the input to the system, but the actions that occur when rules are fired can cause the state of working memory to change. The inference engine must have a conflict resolution strategy to handle cases where more than one rule is eligible to fire.
