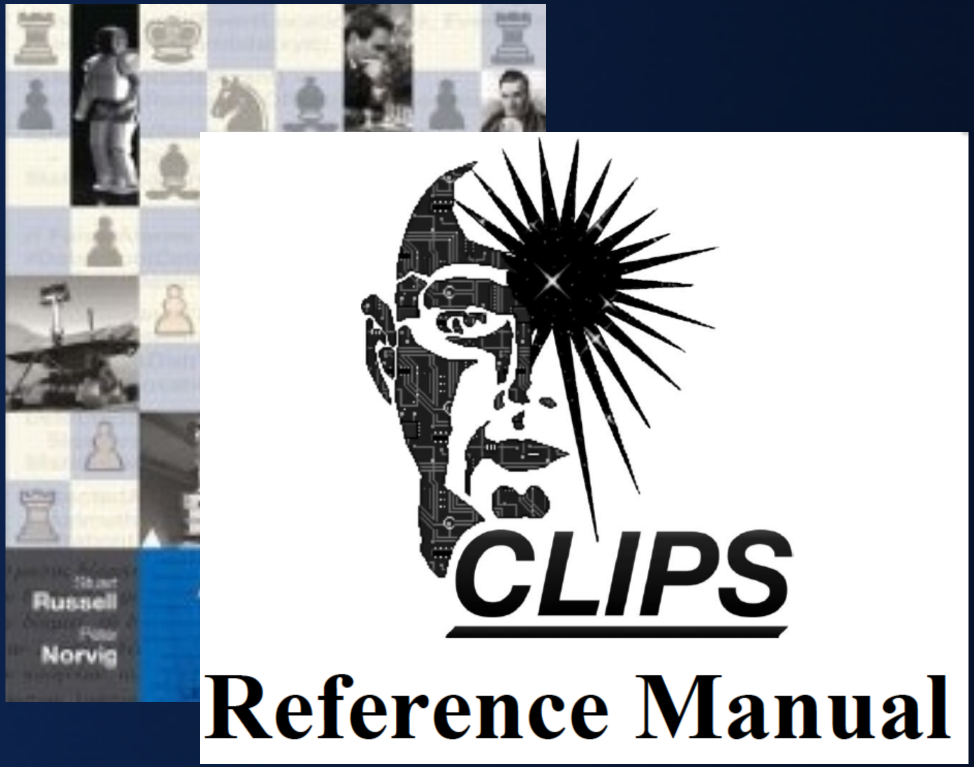


AI Fundamentals: rule-based systems

Maria Simi

Production systems and CLIPS

LESSON 3 – PRODUCTION SYSTEMS – AN INTRODUCTION TO CLIPS – THE RETE MATCH ALGORITHM.

Production systems

Rule-based systems are one of the first paradigms for representing knowledge in A.I.. Most expert systems are rule-based.

Differently from logic programming, rules are used forward to produce new facts, hence the names **productions** and **production systems**.

They are a general computational model based guided by **patterns**: the rule to be applied next is determined by **pattern matching**, a simplified form of unification.

CLIPS

CLIPS is a successor of OPS-5 (Forgy), developed by NASA and now freely available from SourceForge (http://www.clipsrules.net/).

CLIPS is described as **forward-chaining rule-based programming language** written in C (the name stands for "C Language Integrated Production System").

It also provides procedural and object-oriented programming facilities.

CLIPS is probably the most widely used **expert system tool**.

Production system components

A typical rule-based system has four basic components:

1. A list of **facts** (no variables), the temporary **Working Memory (WM)**.

   (*predicate arg*$_1$ *arg*$_2$ … *arg*$_k$) *ordered facts*

   (*predicate* (*slot*$_1$ *val*$_1$) … (*slot*$_k$ *val*$_k$)) *structured facts*

2. A list of rules or **rule base**, a specific type of knowledge base.

   (**defrule** *rule_name* [“*comment*”]

   (*pattern*$_1$) (*pattern*$_2$) … (*pattern*$_k$) => (*action*$_1$) (*action*$_2$) … (*action*$_m$))

3. An interpreter, called **inference engine**, which infers new facts or takes action based on the interaction of the WM and the rule base.

4. A **conflict resolution strategy**.

The inference engine

The interpreter executes a *match-resolve-act* cycle:

1. **Match**: the left-hand sides (**LHS**) of all rules are matched against the contents of the working memory. This produces a **conflict set:** instantiations of all the rules that are satisfied/applicable.
2. **Conflict-Resolution**: one of the rules instantiations in the conflict set is chosen for execution. If no applicable rule, the interpreter halts.
3. **Act**: the actions in the right-end side (**RHS**) of the rule selected in the conflict-resolution phase are executed. These actions may change the contents of working memory. The cycle repeats.

Relation templates for facts in CLIPS

Facts may be structured. In this case the structure of facts must be defined before use. The general format of a deftemplate is:

(**deftemplate** <*relation-name*> [<*optional-comment*>]
    <*slot-definition*>*)

The syntax description <slot-definition> is defined as:

(**slot** <*slot-name*>) | (**multislot** <*slot-name*>)

```
(deftemplate person "example"
   (multislot name)
   (slot age)
   (slot eye-color)
   (slot hair-color))
```

Note: ordered facts do not need a template nor names for slots.

Facts

```
(assert
   (person
      (name John Fox) (age 23)
      (eye-color blue)(hair-color black))
   (person
      (name Jane Fox) (age 22)
      (hair-color brown) (eye-color blue)))
<Fact-1>

(assert
      (food-groups meat dairy bread
           fruits-and-vegetables))
(assert (number-list 7 9 3 4 20))
```

Functions on facts:

1. (assert <*facts*>+)
2. (retract <*fact-index*>+)
3. Modify:
   (modify <*fact-index*>
   <*slot-modifier*>+)
4. (facts)
5. (reset) *asserted facts are deleted*
6. (watch <*watch-item*>)
7. (unwatch <*watch-item*>)

Rules

The general format of a rule is:

(**defrule** <*rule name*> [<*comment*>]
  <*patterns*>* => <*actions*>*)

A rule may have multiple patterns (**condition elements**) in the LHS and actions in the RHS.

Each pattern consists of one or more constraints intended to **match** the fields of a *deftemplate* fact or ordered fact (they must have the same form of facts in WM). The LHS may also contain **tests** on the matched variables.

The actions may include , **assert**, **retract**, **modify**, **printout** … any user defined action

Example:

```
(deftemplate emergency (slot type))
(deftemplate response (slot action))
```

```
(defrule fire-emergency “An example rule"
    (emergency (type fire)) =>
    (assert (response
              (action activate-sprinkler-system))))
```

Matching facts and rules

To identify applicable rules (the conflict set) the interpreter tries to match any pattern in the LHS of a rule with facts in the working memory. This process is called **pattern matching.**

Pattern matching is a simplified form of unification: variables may appear in the pattern but do not appear in facts. The result is a list of **bindings** for the variables which is then used to instantiate the actions in the RHS.

Example. The patterns may contain variables (**?**<*id*>) or wild-cards (**?**):

(person (name John ?a) (age 23) (hair-color ?) (eye-color ?e))

matches with

(person (name John Fox) (age 23) (eye-color blue)(hair-color black))

producing the list of bindings {?a/Fox, ?e/blue}

CLIPS agenda

Matching rules (**activations**) are put in an agenda from where one will be selected according to the following policy:

1. Newly activated rules are added to the agenda and the agenda is reordered according to the **salience** of the rules.
2. Among the rules of **equal salience,** the current **conflict resolution strategy** is used to determine the rule to be **fired** (analogy with neurons).

The command (**agenda**) will display the activated rules in the agenda with their salience. Default salience is 0.

Salience may be controlled with the command (**declare** (**salience** *n*)) by assigning numbers in the range (-10000, +10000).

The firing of the rules in the agenda is activated by the (**run** [<*n*>]) command, where the optional *n* is the number of iterations. No *n* means until agenda is empty.

Conflict resolution

After a neuron fires, it undergoes **refraction** and cannot be fired again for a certain period of time. The same for rules: the rules are fired only once for the same WM elements and rule.

The **depth** strategy is the standard default strategy of CLIPS.

Others predefined strategies are available in CLIPS:

1. **breadth**: newly activated rules are placed **below** all rules of the same salience
2. **simplicity**: among rules of the same salience, less specific rules are preferred.
3. **complexity**: among rules of the same salience, most specific rules are preferred
4. **lex, mea:** derived from OPS-5, both deal with some notion of recency (recently activated rules are preferred)
5. **random**: among rules of the same salience, choose at random. Useful for debugging.

The strategy may be changed with (**set-strategy** <strategy>).

You can define your own.

Monitoring an industrial plant

```
(deftemplate emergency (slot type))

(deftemplate response (slot action))

(defrule fire-emergency
    (emergency (type fire)) =>
    (printout t "Activate the sprinkler system” crlf) )
(defrule flood-emergency
    (emergency (type flood)) =>
    (printout t "Shut down electrical equipment” crlf) )
(assert (emergency (type fire)))
(facts)
(agenda)
(run)
```

Rules with variables: *marriage agency*

```
(deftemplate person
   (slot name)
   (slot sex)
   (slot single)
   (slot status)
   (slot look))

(deffacts customers
    (person (name mary) (sex female)
                 (single t) (status 2) (look best))
    (person (name john) (sex male)
                 (single t) (status 4) (look normal))
    (person (name paul) (sex male)
                 (single t) (status 1) (look ugly))
    (person (name george) (sex male)
                 (single t) (status 1) (look best))
```

```
(defrule marry-money
       (declare (salience 100))
       ?m <- (person (name ?hername)
       (sex female) (single t)
       (status ?herstatus)(look best))
       ?f <- (person (name ?hisname)
       (sex male) (single t) (status ?hisstatus)
       (look normal))
       (test (<= ?herstatus 2))
       (test (> ?hisstatus 2))
        =>
       (modify ?m (single nil))
       (modify ?f (single nil))
       (printout t ?hername " and " ?hisname
                     “married for money" crlf))
```

Complexity issues

The interpreter is required to perform a many-to-many pattern matching. If we have:

- *r* rules to try (in OR)
- *n* preconditions in each rules to be satisfied (in AND)
- *w* facts in the WM (in OR)

$$\begin{tikzcd} \text{(OR . . .} \arrow[d] & \text{(AND . . .} \arrow[d] & \text{(OR . . .)))} \arrow[d] \\ \textit{on r rules} & \textit{on n preconditions} & \textit{on w facts} \end{tikzcd}$$

Everything repeated for c cycles:

$r \times n \times w \times c$ pattern matching operations!

The RETE match algorithm [Forgy 1982]

The **Rete Match Algorithm** is an algorithm for computing the conflict set (the set of instantiations). RETE adopts two basic strategies:

1. **Store information between cycles**: store, with each pattern, a list of the WM elements that it matches. The lists are updated when WM changes, producing a new conflict set.
2. The LHS of the rules are compiled into a **discrimination network** (hence the name RETE):
   - **Pattern compilation:** nodes of the networks are build for each pattern and for each constraint among patterns; equal patterns result in shared nodes.
   - **Execution**: the network is used to compute the current activations
   - **Update**: when the WM is changed, the network is **locally** udated.

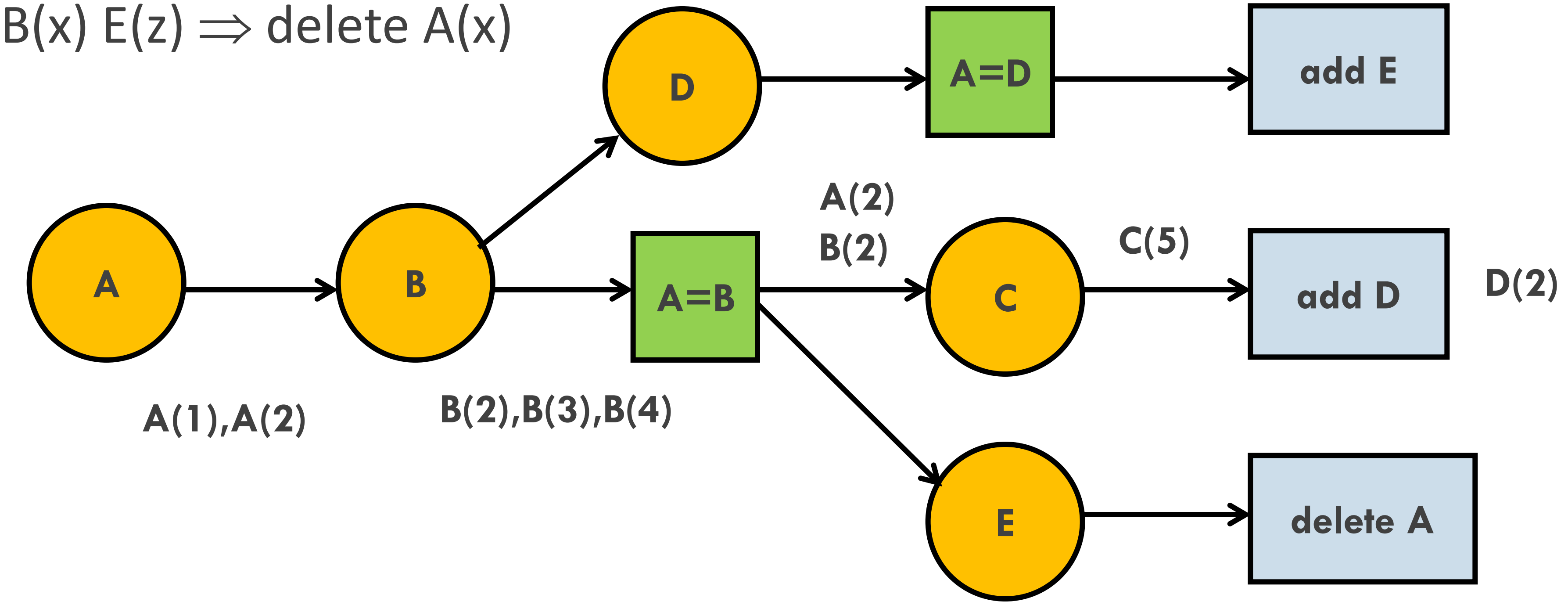
RETE: an example from AIMA (2° ed.)

WM = {A(1), A(2), B(2), B(3), B(4), C(5)}

A(x) B(y) D(x) ⇒ add E(x)

A(x) B(x) C(y) ⇒ add D(x)

A(x) B(x) E(z) ⇒ delete A(x)



Advantages and disadvantages of rule based systems

- Advantages
  - Writing rules is very natural for experts or end-users.
  - Modular architecture: the rule KB is separate for the inference engine.
  - Modularity of rules and incremental acquisition of knowledge.
  - Explanations: justification of conclusions is possible.
  - General programming paradigm: extensible with user defined functions and OOP for object definition and ontologies.
- Disadvantages
  - It may be difficult to control the firing of rules: unexpected behavior.
  - Expressing knowledge as rules may be a bottleneck.

References

- ✓ CLIPS User guide.
- ✓ CLIPS reference manual.
- ✓ Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach* (2nd edition). Pearson Education [Chapter 9]

Your turn

- ✓ You are encouraged to experiment with CLIPS: write your own small expert system in CLIPS
- ✓ Describe the OO extension of CLIPS
- ✓ More on conflict resolution strategies: examples