Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 / MoSIG M1 Second Semester 2011/2012

Lesson 8 7 March 2012

Control of Reasoning and Decision Trees

Using Context to Structure Rules

It is possible to organize an expert system as a finite state machine, where each state corresponds to a rule "context" (also called a phase). Phases (or contexts) can be organized into cycles, networks or trees. The transition between phases can be coded reactively (as rules) or declaratively (as facts). Declarative control is slightly slower but provides the advantages of being easy to inspect, easy to change, and easier to debug. Examples include:

Diagnostic Systems (e.g. MYCIN) - tree of phases (or contexts)
Systems for self-monitoring and self repair – cycles

Within each phase, a set of rules encode the systems knowledge.

In general it is good practice to group rules into contexts (or phases). Each context (or phase) concerns some sort of calculation coded as a body of rules that may fire in any order.

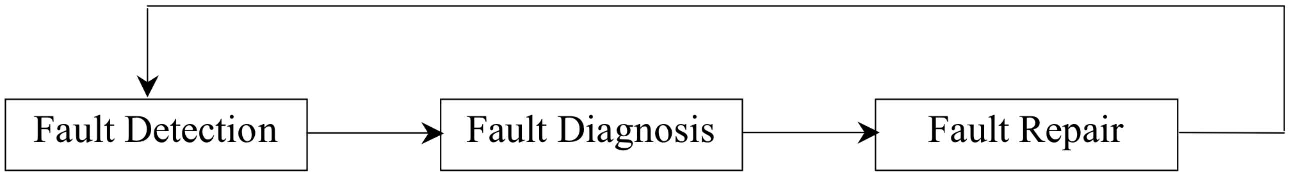
Phases and Control Elements

Phases are indicated by the existence of a token: an element in the facts list that indicates the current phase.

(phase <Name of the phase>)

A classic example is a self-monitoring and self-repair system used for satellites and space applications. Such systems typically operate in cycle with 3 phases:

Fault-Detection: A set of rules that test the integrity of subsystems
Fault-Diagnosis: A set of rules that determine the origin of an error.
Fault-Repair: A set of rules that reconfigure the system to repair a fault.



One can imagine many ways to code control.
For example,

1) It is possible to code the fault condition in each rule. However this is expensive and can allow multiple interacting faults to interfere with each other.

2) It is possible to use salience to define a hierarchy for the rules.

example :	Fault Detection : Salience 3
	Fault Diagnosis : Salience 2
	Fault Repair : Salience 1.

This is a very bad idea that leads to complex un-maintainable code.
DO NOT DO THIS!

3) CORRECT METHOD: Use a “phase” element to mark each phase and then use a rule to manage the transitions . This is the preferred solution.

The transition rules encode the control of the system. These can be reactive or declarative. For example:

```
(defrule detection-to-diagnosis
   (declare (salience 10))
   ?phase <- (phase detection)
   (fault ?f detected)
=>
   (retract ?phase)
   (assert (phase diagnosis))
   (printout t "Fault " ?f " detected!" crlf)
)

(defrule diagnosis-to-repair
   (declare (salience 10))
   ?phase <- (phase diagnosis)
   (fault ?f detected)
   (fault ?f diagnosis ?d)
=>
   (retract ?phase)
   (assert (phase repair))
   (printout t " Fault " ?f " diagnosis " ?d crlf)
)
```

```
(defrule repair-to-detection
   (declare (salience 10))
   ?phase <- (phase repair)
   (fault ?f diagnosis ?d repair ?r)
=>
   (retract ?phase)
   (repair ?f ?d ?r)
   (assert (phase detection))
   (printout t "Fault repaired" crlf)
)
```

Within each phase, are a collection of rules for domain knowledge that diagnosis and suggest a repair for the fault.

Declarative Control Structures

An alternative to coding the phase transitions in explicit rules, is to encode the phase transitions in a declarative data structure and use a single generic transition rule.

```
(deffacts control-list
    (phase detection)
    (next-phase detection diagnosis)
    (next-phase diagnosis repair)
    (next-phase repair detection)
)

(defrule phase transition rule.
    (declare (salience -10))
    ?P <- (phase ?phase)
    (next-phase ?phase ?next)
=>
    (retract ?P)
    (assert (phase ?next))
)
```

This is an example of a DECLARATIVE representation of control knowledge. Declarative structures make it possible to treat knowledge representations as data for calculation. A declarative representation can be used as data to reason about knowledge.

Decision Trees

A decision trees can be used for diagnosis and classification problems.
They require that :

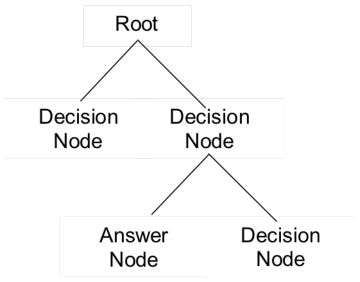
The set of answers be finite and known in advance
The space of problems can be reduced to a series of yes/no tests

A decision tree is composed of decision nodes and leaves.
The decision nodes compose the tree. The answers are found in the leaves.

Example: Learn to guess the animal

The following example is a system that learns to guess an animal by asking questions.

The decision tree is composed of decision-nodes and answer-nodes.



```
(deffacts tree
(node  root decision "Is it warm blooded?" n1 n2)
(node  n1 decision "Does it purr ?" n3 n4)
(node  n2 answer snake)
(node  n3 answer cat)
(node  n4 answer dog)
)

;; initialization rule

(defrule init
   (initial-fact)
=>
 (assert (current-node root))
)

(defrule init
=>
 (assert (current-node root))
)
```

;; rule to request a decision node

```
(defrule make-decision
   ?N <- (current-node   ?name)
   (node  ?name decision ?q ?yes ?no)
=>
   (retract ?N)
   (format t "%s (yes or no) " ?q)
   (bind ?answer (read))
   (if (eq ?answer yes)
      then (assert (current-node ?yes) )
      else (assert (current-node ?no))
   )
)
```

;; rule to give answers

```
(defrule give-answer
   ?N <- (current-node   ?name)
   (node  ?name answer ?r)
=>
   (printout t "I guess that it is a " ?r crlf)
   (printout t "Am I right? (yes or no) ")
   (bind ?rep (read))
   (if (eq ?rep yes)
      then (assert (phase play-again))
         (retract ?N)
      else (assert (phase correct-answer))
   )
)
```

;; rule to play again

```
(defrule play-again
   ?phase <- (phase play-again)
=>
   (retract ?phase)
   (printout t "play again? (yes or no) ")
   (bind ?rep (read))
   (if (eq ?rep yes)
      then (assert (current-node   root))
      else (save-facts "animal.dat")
            (halt)
   )
)
```

```
;; rule to learn the correct answer

(defrule learn-correct-answer
   ?P <- (phase correct-answer)
   ?N <- (current-node   ?name)
   ?D <- (node  ?name answer ?r)
=>
   (retract ?P ?N ?D)  ;;    Ask the correct answer
   (printout t "What animal was it? ")
   (bind ?new (read))
   (printout t  "What question should I ask to tell a "
      ?new " from a " ?r "? ")
   (bind ?question (readline))
   (bind ?newnode1 (gensym*))
   (bind ?newnode2 (gensym*))
   (assert (node  ?newnode1 answer ?new))
   (assert (node  ?newnode2 answer ?r))
   (assert
   (node ?name decision ?question ?newnode1 ?newnode2))
   (assert (phase play-again))
)


;; Rule to open the file animal.dat

(defrule init
   (initial-fact)
=>
 (assert (file (open "animal.dat" data "r")))
)
```

```
;; rule to close the file animal.dat

(defrule no-file
   ?f <- (file FALSE)
=>
   (retract ?f)
   (assert (current-node root))
)

;; rule to read the file.

(defrule init-file
   ?f <- (file TRUE)
=>
 (bind ?in (readline data))
 (printout t ?in crlf)
 (if (eq ?in EOF) then (assert (eof))
  else
    (assert-string ?in)
    (retract ?f)
    (assert (file TRUE))
   )
)

(defrule eof
   (declare (salience 30))
   ?f <- (file TRUE)
   ?eof <- (eof)
=>
   (retract ?f ?eof)
   (close data)
   (assert (current-node root))
)
```