Università degli Studi di Bari
Dipartimento di Informatica

LACAM
Machine Learning

CLIPS: Knowledge Representation and Problem Solving

Antonio Vergari

April 29, 2015

Knowledge Representation

A simple diagnostic classifier

Getting back to the previous example, the knowledge we wanted to embed could be expressed in such a few rules:

I. **If** the skin and eyes are yellowish, **then** there is *icterus* (partial diagnosis)

II. **If** the eyes are yellowish but not the skin, **then** there is *scleral icterus* (partial diagnosis)

III. **If** scleral icterus is present, there is no fever, and the patient's stressed or without food, **then** *Gilbert's syndrome* can be diagnosed

IV. **If** icterus is present, there is fever, the patient is young and tired, dyspepsia has been diagnosed and the liver is enlarged, **then** *Acute viral hepatitis* can be diagnosed

V. **If** there is icterus, fever and the patient is not young but has recurrent pains and cholecyst pains, **then** it is *cholecystitis*

VI. **If** there is icterus, there is no fever and the patient is not young and abuses alchool, the liver is enlarged and the spleen is enlarged as well, **then** it is *alchoolic cirrhosis*

A very simple fact representation

The minimal information needed to represent a symptom as a fact is its name and its presence. In this way we are limiting possible extensions (like user profiling), plus each precondition is treated as a symptom.

```
(deftemplate symptom
    (slot name (type SYMBOL))
    (slot observed (default FALSE)))
```

Thus a diagnostic rule could take this linear form:

```
(defrule alchoolic—cirrhosis
    (symptom (name icterus) (observed TRUE))
    (symptom (name fever) (observed FALSE))
    (symptom (name young) (observed FALSE))
    (symptom (name alcohol—abuse) (observed TRUE))
    (symptom (name enlarged—liver) (observed TRUE))
    (symptom (name enlarged—spleen) (observed TRUE))
    =>
    (assert (diagnosis "Alchoolic cirrhosis")))
```

In the case of partial diagnosis we will assert symptom facts and not diagnosis (this is a limitation as well).

Rule Hierarchy

To exend this simple formalism one can conceptualize in more depth what some symptoms really mean and how to observe them.

For instance, what does it mean for a patient to be young? Or to have the fever?

If the patient's age is less than 25 years, **then** he's young, **otherwise if** it is still less than 50, **then** it is an adult, **else** an elder.
If the patient's temperature is higher than 37.1 Celsius degrees, **then** he has a fever.

We are clearly defining different stages in the diagnosis process, refining available information and collecting missing information. At the same time the interaction (via questions) shall be more focused.

Randomizing order

Our naive conversation behaviour could be more banal if we could start the conversation from any possible rule.

We can achieve this by imposing a random conflict resolution strategy (all rules to ask questions shall have the same priority):

```
(clear)
(set-strategy random)
(load icterus.clp)
(reset)
(run)
```

However, please note that a real diagnostic interview can start from every point, *but then it is then guided by the partial diagnosed diseases and symptoms*. In order to do this we have to introduce meta-rules...

If there is a symptom that can be asked that is characterizing a disease we could diagnose, **then** ask about that symptom

Revising Asserted Knowledge

If we assume each asserted symptom to be erroneously observed, we could provide a routine that lets you change the truth value for them all, calling inference again on the new WM.

```
Would you like to revise the diagnosis? (yes/y/no/n): yes
Would you like to change some observed symptom?
(1) recurrent—pain: TRUE
(2) tired: TRUE
(3) enlarged—liver: TRUE
(4) yellowish—skin: TRUE
...
(11) enlarged—spleen: TRUE
(12) alcohol—abuse: TRUE
(13) cholecyst—pain: TRUE
(14) fever: TRUE
Enter the symptom number or 'e' to return or 'h' to stop:
```

Suppose you are setting yellowish-skin to FALSE, will the scleral icterus rule fire? *what if scleral icterus has already been asserted before*?

Revising Asserted Knowledge

A simple, manual approach:

```
(deffunction get-all-facts-by-names ($?template-names)
    (bind ?facts (create$))
    (progn$ (?f (get-fact-list)) ;; this is a foreach
        (if (member$ (fact-relation ?f) $?template-names)
            then (bind ?facts (create$ ?facts ?f)))) ?facts)

(deffunction change-symptom-by-index (?index)
    (bind ?f (nth ?index (get-all-facts-by-names symptom)))
    (modify ?f (observed (not (fact-slot-value ?f observed)))))

(deffunction ask-to-change-symptom (?question)
    (printout t "Would you like to change some observed symptom?" crlf)
    (print-all-symptoms-status)
    (bind ?n (length$ (get-all-facts-by-names symptom)))
    (bind ?response (ask-question ?question (create$ (gen-int-list ?n) e h)))
    (switch ?response  (case h then (printout t "Halt" crlf) (halt))
                       (case e then (return))
                       (default (change-symptom-by-index ?response)
    (ask-to-change-symptom ?question))))
```

Searching

Search State Representation

We can represent the inference problem as the search in the space of possible diagnosed states.
To implement a problem solver in CLIPS we have to model, for each problem[1]:

states	that form our search space (e.g. problem configurations)
transitions	that allow to jump from one state to its neighbors, the next states (e.g. rules in a game)
constraints	eventually imposed on states (e.g. no repeated or illegal states) and to check for a state to be a solution
search strategy	that controls which transition to apply and which state to move on at each time
explored tree	representing the states explored so far

[1] How could we model a *general problem solver*?

A tentative implementation

The simplest implementation we can try relies on the direct exploitation of the constructs that CLIPS offers, using in a non-transparent way its inference cycle

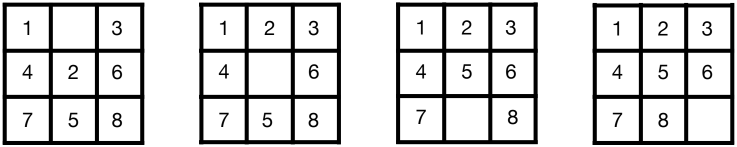
To be more precise we would have:

states	as facts in the current configuration of the WM.
transitions	encoded directly as CLIPS rules with the same priority
constraints	as constrained pattern in the LHS of transitions (e.g. the **not** logical operator). Some rules can be explicitly used for solution states
search strategy	directly relying on CLIPS conflict resolution strategy
explored tree	building it by some other facts as we proceed

Even though it lacks flexibility and constraints us in many ways, we will explore a use case by implementing a very simple game before moving forward.

8-Puzzle

The 8-puzzle game is one of the earliest games exploited in AI. Numbered tiles, from 1 to 8, are placed on a 3×3 grid.
Starting from an initial grid configuration the objective is to reach a final state in which all tiles are ordered numerically with the last one left empty.
The allowed moves consist in exchange the position of the the empty tile with one of its (at most) four neighbors.



At this point of the course you should be pretty familiar with this game.

8-Puzzle: Facts and Rules

Concerning states we can represent a grid configuration with a single fact. The information to be stored concerns *the numbered positions* and *the actual value* in them. For instance, we can use a special number like -1 to represent the empty tile value.

Even if inefficiently we can take advantage of the WM by simply asserting and never retracting (remind assertion and refraction properties).

We can have each rule modeling a single possible empty cell movement (24 rules total), something like:

If the grid has X_1 in the 1st position, -1 in the 2nd, X_3 in the third, X_4 in the fourth, X_5 in the fifth, ..., **then** the new grid will be: X_1 in the 1st position, X_5 in the 2nd, X_3 in the third, X_4 in the fourth, -1 in the fifth,...

We could reduce the number of rules by exploiting functional abstraction.

8-Puzzle: Constraints and Strategy

Avoiding loops is done by the WM use we are doing and by enforcing to match only rules leading to unseen configurations.
Refraction helps, but we need to avoid even to trigger rules that would assert and already asserted state.
One possible solution is to put a constraint as a pattern in the LHSs:

If the grid has X_1 in the 1st position, -1 in the 2nd, X_3 in the third, X_4 in the fourth, X_5 in the fifth, ..., **AND** there is *not* X_1 in the 1st position, X_5 in the 2nd, X_3 in the third, X_4 in the fourth, -1 in the fifth..., **then** the new grid will be: X_1 in the 1st position, X_5 in the 2nd, X_3 in the third, X_4 in the fourth, -1 in the fifth,...

The search strategy can be demanded to CLIPS by enforcing all rules to have the same salience and by employing the breadth or depth conflict resolution strategy.

```
(set—strategy depth)
```

What is, for this particular problem, the best strategy to employ? why?

8-Puzzle: Constraints and Search Tree

As a last constraint we need rules to check for a state to be the final one (solution). This is simply done like in the previous fashion.

To build a search tree we can store its edge information as facts containing information about the two configurations of the world, the parent and child grids.

After a solution has been found we have to crawl back from that state to the root. To do this we need a simple rule that searches for a state matching for the parent attribute in our edge facts

Exercise

Implement such a problem solver for the 8-puzzle.

Hint 1:
While programming apply a debugging strategy, inspect the WM after each activated rule execution. Moreover, keep track of strategy resolution by printing to stout the state that activates a rule and the one that the rule asserts.

Hint 2:
To set the strategy and maybe constants use another file to by loaded with the batch function.

Hint 3:
For the tree representation you can use something like this for edges, containing simply the fact addresses (pointers):

```
(deftemplate move
    (slot parent (type FACT—ADDRESS) (default ?NONE))
    (slot next (type FACT—ADDRESS) (default ?NONE)))
```

Pros&Cons

What ***we gain for free*** is CLIPS inference routines and constructs, i.e. we do not have to implement a breadth first search or the neighbor space generation by ourselves (we just describe how they are generated via transition rules).

However, *we cannot explore the search space in a more sophisticated way*.
If we use CLIPS conflict resolution strategy for free, we cannot assign different priorities to rules. Neither it is possible to implement other search techniques like best-first search, A^*, and so on. We could implement the Manatthan distance heuristic score as a function, but we would have no way to embed its evaluation on the pairs of states.

A higher level representation for transition rules and states is needed.

Cannibals and Missionaries