Tutorial 9
CLIPS Programming

COMPSCI 367

AGENDA

- Last week’s problem discussion
- Some more CLIPS
- Assignment Queries

Home Task Discussion

Below are heuristics for choosing which wine to have with a meal:

- *"If the main course is red meat then serve red wine."*
- *"If the main course is poultry and it is turkey then serve red wine."*
- *"If the main course is poultry and it is not turkey then serve white wine."*
- *"If the main course is fish then serve white wine."*

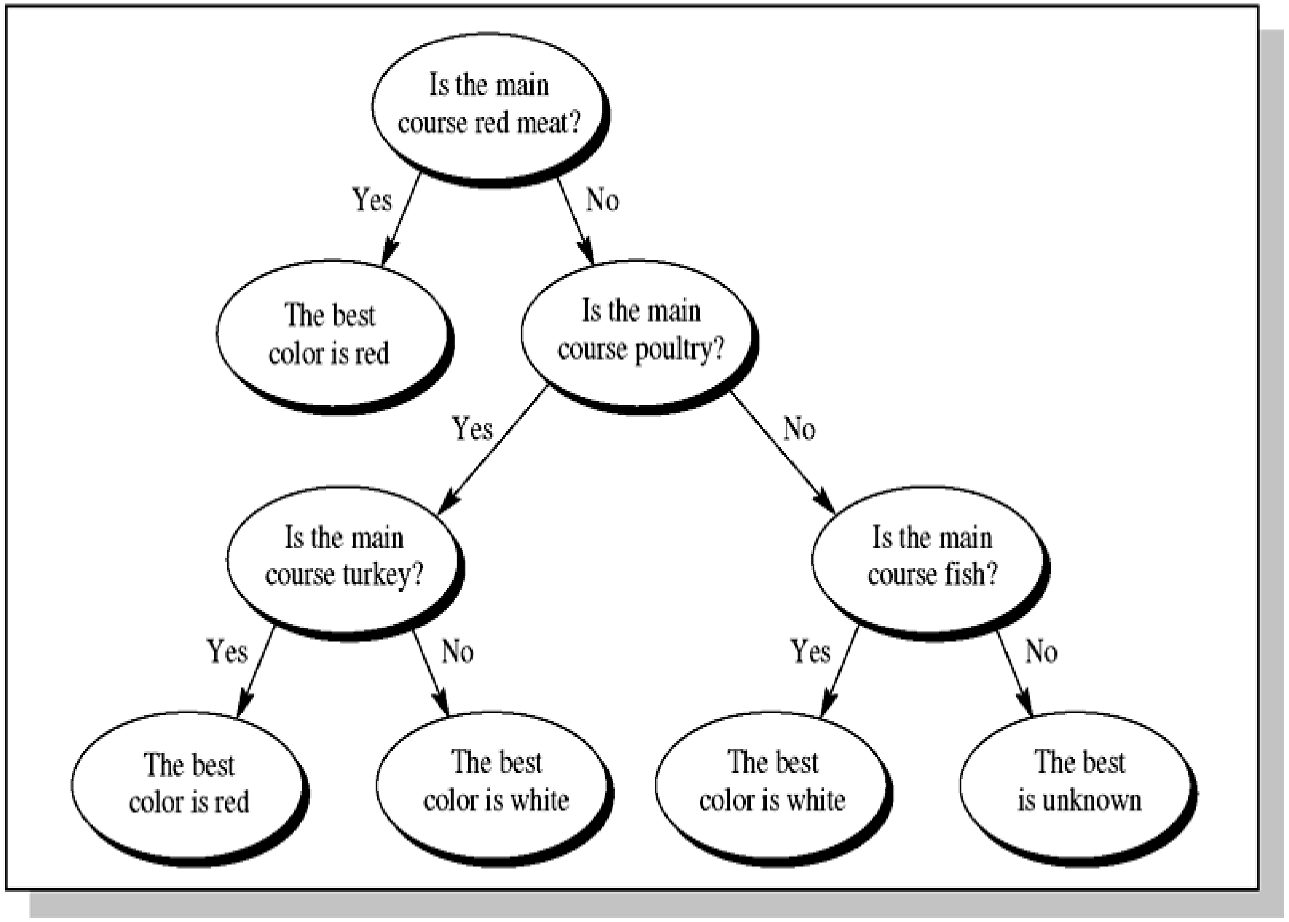
Tasks

a) Construct a binary decision tree (yes/no branching) to represent these heuristics.

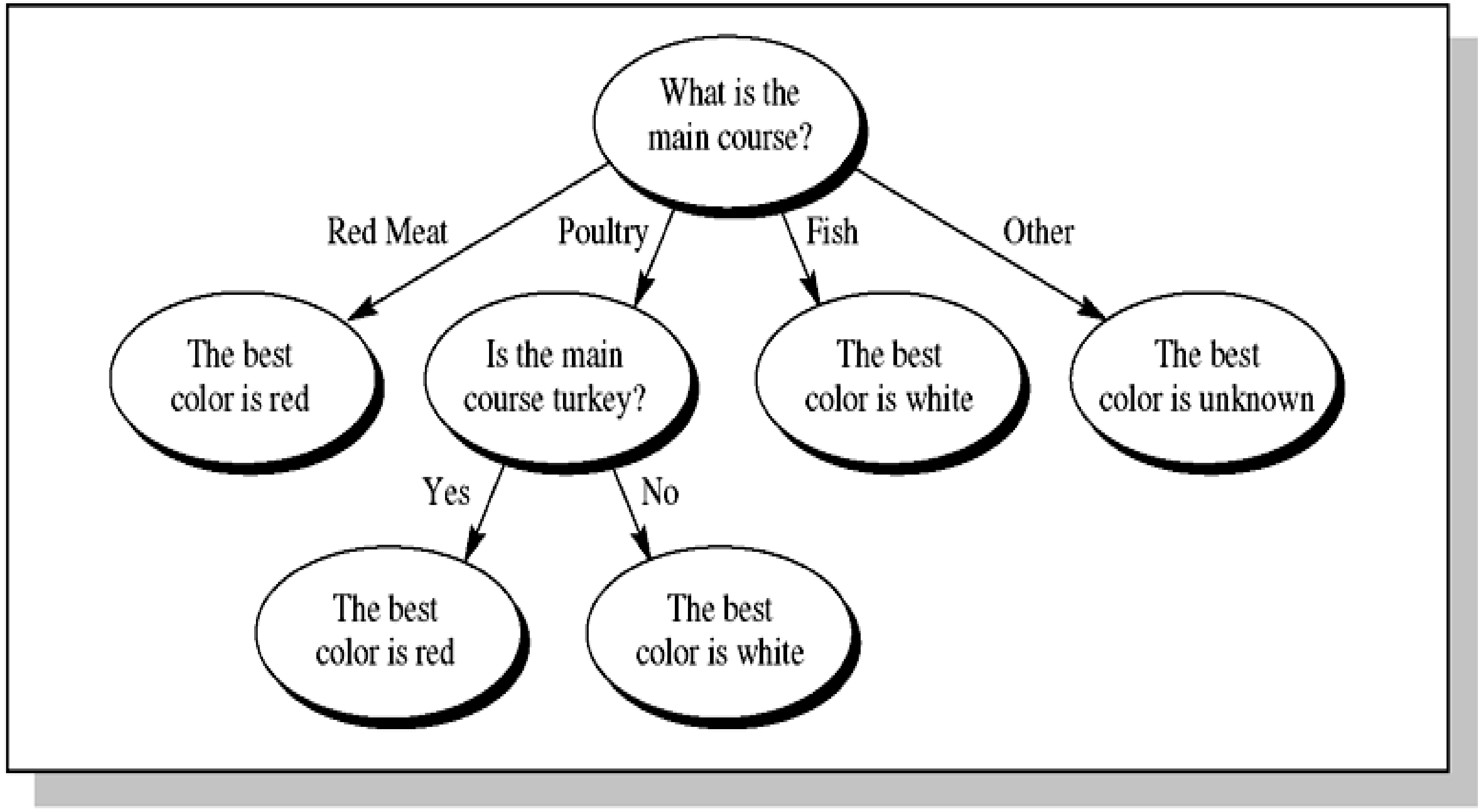
b) Using the wine-choosing heuristics above, construct a decision tree with multiplebranches.

Hint: make the root decision "What is the main course?".

Binary Decision Tree



Decision Tree with Multiple Branches



Can Facts be changed?

NO

- “modify” CLIPS keyword retracts the fact and then asserts a new fact with the changes

FACTS ARE STATIC/IMMUTABLE

Example

The function modify allows you to change the value of one or more slots on a fact. Suppose it's Andrew's birthday.

If we define the rule

```
(defrule birthday ?birthday <- (birthday ?name) ?data-fact
 <- (personal-data (name ?name) (age ?age))
      => (modify ?data-fact (age (+ ?age 1)))
      (retract ?birthday) )
```

Asserting the fact (birthday Andrew) will modify Andrew's age while leaving all his other personal data intact. Incidentally, the reason we retract the birthday fact is for the purposes of truth maintenance.

It's only Andrew's birthday once a year, and if we left the fact lying around it would soon become false. Further, every time any other part of Andrew's personal data (weight, for example) was changed the birthday rule would be fired again, causing rapid ageing!

Getting data from the user

- (read) function

  Wherever (read) is encountered, the program waits for the user to type something in, then substitutes that response.

  To demonstrate this, type

  (assert (user-input (read))) at the CLIPS prompt. After you press return, you'll need to type something else (anything) before the command will complete.

  When you look at the facts, you'll see a new fact with your input as the second item.

(read) example

(defrule what-is-child (animal ?name) (not (child-of ?name ?)) => (printout t "What do you call the child of a " ?name "?") (assert (child-of ?name (read))))

CAR Example:

(defrule are-lights-working (not (lights-working ?)) => (printout t "Are the car's lights working (yes or no)?")

(assert (lights-working (read))))

Modelling Knowledge

- we have facts: “You are a vegan”
- we have rules: “Hungry vegans can eat tofu and nut loaf”
- can use rules on current facts to help solve problems, e.g.
    a) selection (“I’m hungry – what should I eat?”)
    b) diagnosis (“The car won’t start – what is the problem?”)
    c) classification (“What animal is this?”)

Decision Trees

1. Decision trees provide a useful paradigm for solving certain types of classification problems.
2. Decision trees derive solutions by reducing the set of possible solutions with a series of decisions or questions that prune their search space.
3. Problems suitable for decision trees are those that provide the answer to a problem from a predetermined set of possible answers.

Decision Trees

4. Decision trees consist of nodes and branches, connecting parent nodes to child from the top to bottom. The top node (root) has no parent. Every other node has only one parent. Nodes with no children are leaves.
5. Leaf nodes represent all possible solutions that can be derived from the tree – answer nodes. All other nodes are decision nodes.

Decision Trees with Multiple Branches

- A binary decision tree may prove inefficient – not allowing for a set of responses or a series of cases.
- A modified decision tree allows for multiple branches – giving a series of possible decisions.

Rule-Based Decision Tree Program

- The first step implementing the learning process in a decision tree in CLIPS is to decide how knowledge should be represented.
- Since the tree should learn, the tree should be represented as facts instead of rules – facts are easily added / deleted from a tree.
- A set of CLIPS rules can be used to traverse the decision tree by implementing the *Solve_Tree_and_Learn* algorithm using rule-based approach.

Retract facts

- In order to retract a fact, you need to know its fact-index. You can retract facts from within rules by binding them to variables, like this:

  ```
  (defrule remove-mammals ?fact <- (mammal ?) =>
                  (printout t "retracting " ?fact crlf) (retract ?fact))
  ```

- In the pattern part of this rule, the variable ?fact is given the fact-index of each fact matching the pattern (mammal ?) in turn. That's what the leftwards arrow (<-) symbol means. When you run it, this is what happens (the fact numbers may be different):

-
  ```
  CLIPS>(run)
  retracting <Fact-13>
  retracting <Fact-14>
  retracting <Fact-15>
  retracting <Fact-16>
  CLIPS>
  ```
- All the mammal facts have been retracted.

Input/Output

- print information
  (printout <logical-device> <print-items>*)
- logical device frequently is the standard output device t (terminal)
  terminal input
  (read [<logical-device>]), (readline [<logical-device>])
  - read an atom or string from a logical device

- open / close file
  (open <file-name> <file-ID> [<mode>]), (close [<file-ID>])
  open /close file with <file-id> as internal name
- load / save constructs from / to file
  (load <file-name>), (save <file-name>)
  backslash \ is a special character and must be ``quoted'' (preceded by a backslash \)
  e.g. (load "B:\\clips\\example.clp")

Implementation in CLIPS

- In general, first thing you need to decide is how to represent knowledge – there are different options available, e.g. could represent served wine heuristics
- as **defrules or even just facts with deftemplates**

- Implementation must reflect underlying data structure.
- For a decision tree this means:

  a) implement decision nodes (root and non-leaf nodes)

  b) implement answer nodes (leaf nodes)
- Finally you need to consider:

  a) handling i/o

  b)handling application startup / shutdown

Assignment Example

- The decision tree opposite represents a small section of the diagnosis of a car’s failure to start.
- Each rounded box is a recommended remedy.
- Each rectangular box is piece of evidence, which might be represented by a fact such as **(lights-working no)** or **(petrol yes)**.
- Each connecting path to a remedy represents a rule, for example ‘IF starter is turning AND there is no petrol THEN buy some petrol’.

Examples

- mixes i/o and data structure
- decision nodes = CLIPS rules.

```
*****************************
rule20
 rule to test if a blue tinge on terminals
*****************************
(defrule blue-tinge
(declare (salience -10))
(burner problem y)
(burner type ?) =>
(printout t crlf crlf "Check the ends of the terminals for any sign of a blue" crlf
"tinge. Does one exist? y or n ")
(assert (blue tinge =(read))))
```

Examples

- **Answer nodes=CLIP rules**

```
;****************************
;rule27:
; handles if there is a crimped wire
;****************************
(defrule crimped-wire
?x <- (crimped wire y)
=>
(printout t crlf crlf "First be sure the stove is unplugged. Next strip the" crlf
"insulation away from the crimped area and twist the loose“ Crlf "end together.
Solder the wire and cover the splice with" crlf "a ceramic nut." crlf "*** CAUTION -
do not use a plastic nut as the high temperature" crlf " will cause it to melt" crlf
"Afterward, plug the stove in and recheck the element. If  there" crlf "is still a
problem rerun this program." crlf)
(retract ?x)
(assert (stop)))
```