

## More on Rules

Dan Rochowiak  
[drochowi@cs.uah.edu](mailto:drochowi@cs.uah.edu)

---

---

---

---

---

---

---

---

## Prefer Forward Chaining If...

- The rules are such that a typical hypothesis of a starting fact leads to many questions (there is a high degree of fan-in)
- The number of ways to reach a conclusion is large, but the number of conclusions one is likely to reach is small
- All of the facts are in hand, and the goal of the system is to draw all of the information out of them

---

---

---

---

---

---

---

---

## Prefer Backward Chaining If...

- The rules are such that typical facts can lead to many conclusions (a high degree of fan-out)
- The facts can lead to many conclusions but the number of ways to reach a particular desired or important conclusion is small
- The facts have not been gathered and the intent of the system is to find whether one of many possible conclusions is correct

---

---

---

---

---

---

---

---

## Backward Chaining

1. Get hypothesis
2. Find a rule whose consequent (assertion action) matches hypothesis
3. Create a binding set
4. Using the binding set
5. Try to match antecedent with existing assertion
6. If there is such an antecedent, then  
make it the hypothesis  
backward chain with binding set

---

---

---

---

---

---

---

---

## Backward Chaining

7. Repeat 4, 5, and 6 for each antecedent accumulating bindings until
8. There is no match with assertions or rule antecedents and the bindings
9. Back up to most recent unexplored alternative
10. If there are no more antecedents to be matched then  
hypothesis is supported
11. There are no more alternatives to be explored

---

---

---

---

---

---

---

---

## Simple Examples

- Taxes
- <http://wwwdev.uah.edu/dmr/cs530/taxes.txt>
- Car diagnosis
- <http://wwwdev.uah.edu/dmr/cs530/car.txt>

---

---

---

---

---

---

---

---

## A Standard Problem

- Animal classification
- Use forward chaining
- Proceed from top level category until animal is classified
- Used structured data to store an ontology
- Use few rules.
- Raw material:  
<http://wwwdev.uah.edu/dmr/cs530/animals1.txt>

---

---

---

---

---

---

---

---

## Ontology Issues

- The world for this program is composed of categories, related questions and answers, and a thing to be classified.
- The categories have more specific determinations but the sequence of categories is in a fixed hierarchical order

---

---

---

---

---

---

---

---

## Ontology Issues

- ```
(deftemplate category
  (slot name (default none))
  (slot level (default none))
  (slot difference (default none))
  (slot value (default none))
  (slot super (default none)))
```
- In other words a category is defined in terms of a name, level, difference, value and its relation to a super category.

---

---

---

---

---

---

---

---

## Ontology Issues

- So a category has a name, level, difference and value and is a kind of its super category.
- The categories are assumed to represent the classification of animals. So there are animals as well. At this point they are not explicitly represented.

---

---

---

---

---

---

---

---

## Ontology Issues

- Attempt to build a similar ontology in Protégé
- Now how to get the constructs out
  - First approach: export to clips format
    - Edit the file as needed
  - Second approach: Use Protégé api
    - Build a java application to run Jesse and use the Protégé api to get and set Jess constructs

---

---

---

---

---

---

---

---

## Ontology Issues

- Since there is an ordered structure that the requirement is for forward chaining an order must be imposed.

```
(deftemplate sequence
(multislot items))
```

```
(deftemplate currentLevel
(slot level (default top))
(slot super (default top)))
```

---

---

---

---

---

---

---

---

## Ontology Issues

- Since there is a user interaction, the user must be modeled. The user is modeled by questions and answers associated with the categories.

```
(deftemplate question
  (slot id (default none))
  (slot text (default "a question text"))
  (slot type (default none))
  (multislot value ))

(deftemplate answer
  (slot id (default none))
  (slot value (default none))
  (slot type (default none)))
```

---

---

---

---

---

---

---

---

## A General Pattern

- The general pattern here is rather simple.
  - A schematic pattern,
  - A set of filling instructions for the schematic elements in the pattern,
  - A classification describing the inferential characteristics of the schematic pattern

---

---

---

---

---

---

---

---

## General Pattern

- A schematic pattern - A pattern is classified if its categories are identified and there are no further subcategories
- A set of filling instructions - The ontology of the categories
- A classification - pattern matching of user responses to questions associated with the categories

---

---

---

---

---

---

---

---

## Utilities for Filling

```
(deffunction ask (?text ?type ?value)
  (bind ?answer "")
  (while (not (isOfType ?answer ?type ?value)) do
    (printout t ?text " ")
    (if (eq ?type multi) then
      (printout t " Valid answers are" crlf)
      (foreach ?item ?value
        (printout t ?item " "))
      (printout t ": ")
      (bind ?answer (read)))
    (return ?answer))
```

## Utilities for Filling

```
(deffunction isOfType (?answer ?type ?value)
  (if (eq ?type multi) then
    (foreach ?item ?value
      (if (eq (sym-cat ?answer) (sym-cat ?item)) then
        (return TRUE)))
    (return FALSE))
  (if (eq ?type number) then
    (return (isNumber ?answer)))
  (return (> (str-length ?answer) 0 )))

(deffunction isNumber (?value)
  (try
   (integer ?value)
   (return TRUE)
  catch
   (return FALSE)))
```

## Ontology Specified

```
(defaults start
  (sequence (items superphylum phylum class))
  (category (name backbone) (level superphylum) (difference backbone) (value yes) (super top))
  (category (name jellyback) (level superphylum) (difference backbone) (value no) (super top))
  (question (id backbone) (text "Does the animal have a backbone?") (type multi) (value yes no))
  (category (name warm) (level phylum) (difference bloodTemp) (value yes) (super backbone))
  (category (name cold) (level phylum) (difference bloodTemp) (value no) (super backbone))
  (question (id bloodTemp) (text "Is the animal warm blooded?") (type multi) (value yes no))
  (category (name soil) (level phylum) (difference soil) (value yes) (super jellyback))
  (category (name elsewhere) (level phylum) (difference soil) (value no) (super jellyback))
  (question (id soil) (text "Does the animal live in soil?") (type multi) (value yes no))
  (category (name breasts) (level class) (difference breasts) (value yes) (super warm))
  (category (name bird/penguin) (level class) (difference breasts) (value no) (super warm))
  (question (id breasts) (text "Does the animal have breasts?") (type multi) (value yes no))
```

## Ontology Specified

```
category (name water) (level class) (difference water) (value yes) (super cold)
(category (name dry) (level class) (difference water) (value no) (super cold))
(question (id water) (text "Does the animal live in water?") (type multi) (value yes
no))
(category (name flatworm) (level class) (difference flatBody) (value yes) (super
soil))
(category (name worm/leech) (level class) (difference flatBody) (value no) (super
soil))
(question (id flatBody) (text "Does the animal have a flat body?") (type
multi) (value yes no))
(category (name segments) (level class) (difference segments) (value yes) (super
elsewhere))
(category (name unified) (level class) (difference segments) (value no) (super
elsewhere))
(question (id segments) (text "Does the animal have a segmented body?") (type
multi) (value yes no))
)
```

## Rules to Enforce Classification

```
(defrule begin
?list <- (sequence (items $?items))
(not (currentLevel (level ?)(super ?)))
=>
(printout t "starting up" crlf)
(assert (currentLevel (level (nth$ 1
(first$ ?items))) (super top)))
(modify ?list (items (rest$ ?items)))
)
```

## Rules to Enforce Classification

```
(defrule process
(currentLevel (level ?l) (super ?s))
(category (name ?n) (level ?l) (difference ?q) (value ?) (super
?s))
(question (id ?q) (text ?t) (type ?type) (value $?v))
(not (answer (id ?q)))
=>
(printout t "ask a question for " ?l crlf)
(bind ?response (sym-cat (ask ?t ?type ?v)))
(if (eq ?response yes) then
(printout t "the response is " ?response crlf)
(assert (answer (id ?q) (value ?response))))
)
(if (eq ?response no) then
(printout t "the response is " ?response crlf)
(assert (answer (id ?q) (value ?response))))
))
```

## Rules to Enforce Classification

```
(defrule classify
?level <- (currentLevel (level ?l))
?answer <- (answer (id ?q) (value ?response) (type none))
?list <- (sequence (items $?items))
(category (name ?n) (level ?l) (difference ?q) (value
?response) (super ?s))
=>
(printout t "classifying " ?l ": " ?n crlf)
(retract ?level)
(modify ?answer (type ?n))
(assert (currentLevel (level (nth$ 1 (first$ ?items)))(super
?n)))
(modify ?list (items (rest$ ?items)))
)
```

---

---

---

---

---

---

---

---

## The Classic Backward Chaining Approach

- Instead of diving into the hierarchy in specified order one might think about the problem in terms of sub-goals.
- In this case one would be looking at implementing a backward chaining solution.
- In Jess backward chaining can be emulated either explicitly or by invoking the Jess backward chaining mechanism.

---

---

---

---

---

---

---

---

## Explicit Backward Chaining

- In this case the domain is a collection of representations of domain classification rules.

```
(deftemplate rule
(multislot if)
(multislot then))
```

---

---

---

---

---

---

---

---

## Sub-goaling

```
(defrule propagate-goal ""
  (goal is ?goal)
  (rule (if ?variable $?)
        (then ?goal ? ?value)))
=>
(assert (goal is ?variable)))

(defrule goal-satisfied ""
  (declare (salience 30))
  ?f <- (goal is ?goal)
  (variable ?goal ?value)
  (answer ? ?text ?goal)
=>
(retract ?f)
(format t "%s%s\n" ?text ?value))
```

---

---

---

---

---

---

---

---

## Rule Matching

```
(defrule remove-rule-no-match ""
  (declare (salience 20))
  (variable ?variable ?value)
  ?f <- (rule (if ?variable ? ~?value $?))
=>
(retract ?f))

(defrule modify-rule-match ""
  (declare (salience 20))
  (variable ?variable ?value)
  ?f <- (rule (if ?variable ? ?value and $?rest))
=>
(modify ?f (if ?rest)))
```

---

---

---

---

---

---

---

---

## Satisfaction

```
(defrule rule-satisfied ""
  (declare (salience 20))
  (variable ?variable ?value)
  ?f <- (rule (if ?variable ? ?value)
            (then ?goal ? ?goal-
value))
=>
(retract ?f)
(assert (variable ?goal ?goal-value)))
```

---

---

---

---

---

---

---

---

## Questioning

```
(defrule ask-question-no-legalvalues ""
  (declare (salience 10))
  (not (legalanswers $?))
  ?f1 <- (goal is ?variable)
  ?f2 <- (question ?variable ? ?text)
  =>
  (retract ?f1 ?f2)
  (format t "%s " ?text)
  (assert (variable ?variable (read))))
```

---

---

---

---

---

---

---

---

## Questioning

```
(defrule ask-question-legalvalues ""
  (declare (salience 10))
  (legalanswers ? ?answers)
  ?f1 <- (goal is ?variable)
  ?f2 <- (question ?variable ? ?text)
  =>
  (retract ?f1)
  (format t "%s " ?text)
  (printout t ?answers " ")
  (bind ?reply (read))
  (if (member (lowercase ?reply) ?answers)
      then (assert (variable ?variable ?reply))
          (retract ?f2)
      else (assert (goal is ?variable))))
```

---

---

---

---

---

---

---

---

## Ontology Sample

```
(defaults knowledge-base
  (goal is type:animal)
  (legalanswers are yes no)
  (rule (if backbone is yes)
        (then superphylum is backbone))
  (rule (if backbone is no)
        (then superphylum is jellyback))
  (question backbone is "Does your animal have a backbone?")
  (rule (if superphylum is backbone and
        warm.blooded is yes)
        (then phylum is warm))
  (rule (if superphylum is backbone and
        warm.blooded is no)
        (then phylum is cold))
  (question warm.blooded is "Is the animal warm blooded?")
  (rule (if superphylum is jellyback and
        live.prim.in.soil is yes)
        (then phylum is soil))
  (rule (if superphylum is jellyback and
        live.prim.in.soil is no)
        (then phylum is elsewhere))
  (question live.prim.in.soil is "Does your animal live primarily in soil?"))
```

---

---

---

---

---

---

---

---

## Animal Codes

- Standard version
- <http://wwwdev.uah.edu/dmr/c530/animals2.txt>
- A shot at learning
- <http://wwwdev.uah.edu/dmr/c530/animals3.txt>
- <http://wwwdev.uah.edu/dmr/c530/animals3dat.txt>

---

---

---

---

---

---

---

---

## Backward Chaining Using JESS

- Jess can backward chain. you first declare that certain fact templates will be backward chaining reactive using the do-backward-chaining function:

```
Jess> (do-backward-chaining factorial)
```

- If the template is unordered -- i.e., if it is explicitly defined with a (deftemplate) construct -- then it must be defined before calling do-backward-chaining. Then you can define rules which match such patterns. Note that do-backward-chaining must be called before defining any rules which use the template.

---

---

---

---

---

---

---

---

## Backward Chaining Using JESS

```
Jess> (defrule print-factorial-10
  (factorial 10 ?r1)
  =>
  (printout t "The factorial of 10 is " ?r1 crlf))
```

- When the rule compiler sees that a pattern matches a backward chaining reactive template, it rewrites the rule and inserts some special code into the internal representation of the rule's LHS. This code asserts a fact onto the fact-list that looks like  

```
(need-factorial 10 nil)
```
- if, when the rule engine is reset, there are no matches for this pattern. The head of the fact is constructed by taking the head of the reactive pattern and adding the prefix "need-".

---

---

---

---

---

---

---

---

## Backward Chaining Using JESS

- Write rules which match these need-(x) facts.

```
Jess> (defrule do-factorial
  (need-factorial ?x ?)
  =>
  (bind ?r 1)
  (bind ?n ?x)
  (while (> ?n 1)
    (bind ?r (* ?r ?n))
    (bind ?n (- ?n 1)))
  (assert (factorial ?x ?r)))
```

- The rule compiler rewrites rules like this too: it adds a negated match for the factorial pattern itself to the rule's LHS.

---

---

---

---

---

---

---

---

## Backward Chaining Using JESS

- The end result is that one can write rules which match on (factorial), and if they are close to firing except they need a (factorial) fact to do so, any (need-factorial) rules may be activated. If these rules fire, then the needed facts appear, and the (factorial)-matching rules fire. This is backward chaining. Jess will chain backwards through any number of reactive patterns.
- There is a special conditional element, (explicit), which you can wrap around a pattern to inhibit backwards chaining on an otherwise reactive pattern.

---

---

---

---

---

---

---

---

## Logical Backtracking

- A logical condition element provides a truth maintenance facility.
- A fact asserted from the RHS can be made logically dependent upon the facts that matched the patterns enclosed in the logical CE on the LHS. The facts matching on the LHS provide logical support for those on the RHS.
- If all logical support is removed for a fact that requires logical support it is removed.

---

---

---

---

---

---

---

---

## Logical Backtracking Example

|                                                                                                                                                                   |                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Define a rule named megan<br>If<br>(logical (a)) (logical (b)) (c)<br>then<br>Display the message<br>"This is from Megan to the world: g h. "<br>(assert (g) (h)) | Define a rule named wow<br>If<br>(f) (g)<br>then<br>Display the message<br>"Wow! This is neat!"<br>(assert (i)) |
| Define a rule named adam<br>If<br>(logical (d)) (logical (e)) (f)<br>then<br>Display the message<br>"This is from Adam to the world: g h. "<br>(assert (g) (h))   | Define a collection of facts named<br>my-little-facts<br>(a)<br>(b)<br>(c)<br>(d)<br>(e)<br>(f)                 |

---

---

---

---

---

---

---

---

## Logical Backtracking Example

|                                                         |                                                                    |
|---------------------------------------------------------|--------------------------------------------------------------------|
| Initialize fact list                                    | Add to WM f-4 (d)                                                  |
| Add to WM f-0 (initial-fact)                            | Add to WM f-5 (e)                                                  |
| Add to WM f-1 (a)                                       | Add to WM f-6 (f)                                                  |
| Add to WM f-2 (b)                                       | The next three facts are<br>asserted and activate the<br>rule adam |
| Add to WM f-3 (c)                                       | Add to agenda Activation 0<br>adam: f-4,f-5,f-6                    |
| Three facts are asserted and<br>activate the rule megan |                                                                    |
| Add to agenda Activation 0<br>megan: f-1,f-2,f-3        |                                                                    |

---

---

---

---

---

---

---

---

## Logical Backtracking Example

|                                                                                  |                                                                                      |
|----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| The command to run the<br>inference engine is issued<br>adam is fired.           | Add to WM f-8 (h)<br>h is asserted and then wow is<br>fired                          |
| <b>FIRE 1 adam: f-4,f-5,f-6</b><br>This is from Adam to the world: g h.          | <b>FIRE 2 wow: f-6,f-7</b><br>Wow! This is neat!                                     |
| Add to WM f-7 (g)<br>g is asserted from adam's<br>action and wow is<br>activated | Add to WM f-9 (i)<br>wow asserts i and then the next<br>rule on the agenda is fired. |
| Add to agenda Activation 0<br>wow: f-6,f-7                                       | <b>FIRE 3 megan: f-1,f-2,f-3</b><br>This is from Megan to the world: g h.            |

---

---

---

---

---

---

---

---

## Logical Backtracking Example

What happens if f-1 is retracted?  
(retract 1)  
Remove from WM f-1 (a)  
The fact a is removed and nothing else.  
But suppose f-4 is now retracted.  
(retract 4)

d is removed, but then g and h are removed because they lack logical support  
Remove from WM f-4 (d)  
Remove from WM f-7 (g)  
Remove from WM f-8 (h)

---

---

---

---

---

---

---

---

## Logical Backtracking Example

Now what happens if h and g are asserted?  
(assert (h))  
Add to WM f-10 (h)  
(assert (g))  
Add to WM f-11 (g)

wow is activated.  
Add to agenda Activation 0  
wow: f-6,f-11  
Running the inference engine yields  
FIRE 1 wow: f-6,f-11  
Wow! This is neat!

---

---

---

---

---

---

---

---

## Confidence Factors

H is the hypothesis  
E<sub>1</sub> and E<sub>2</sub> are the evidence  
If CF(H,E<sub>1</sub>) and CF(H, E<sub>2</sub>) have opposite signs

$$CF(H,E_1 E_2) = \frac{CF(H,E_1) + CF(H,E_2)}{1 - \min[|CF(H,E_1)|, |CF(H,E_2)|]}$$

---

---

---

---

---

---

---

---

## Confidence Factors

- If  $CF(H, E_1)$  and  $CF(H, E_2)$  are both greater or equal to zero :

$$CF(H, E_1 E_2) = CF(H, E_1) + CF(H, E_2) - CF(H, E_1)CF(H, E_2)$$

- If  $CF(H, E_1)$  and  $CF(H, E_2)$  are both less than zero :

$$CF(H, E_1 E_2) = CF(H, E_1) + CF(H, E_2) + CF(H, E_1)CF(H, E_2)$$

---

---

---

---

---

---

---

---