Production Systems

- Knowledge sources: rules, working memory, possibly semantic memory
- Processing is driven by IF-THEN rules
- Rules trigger ACTIONS (additions and deletions from WM, and possibly others)
- Processing continues until a stop condition (e.g., no changes in a given cycle)
An example for bagging groceries

(Winston, 1993)

Knowledge engineer identifies steps:

- Check order
- Bag large items
- Bag medium items
- Bag small items
rule b1
IF the step is bag-large-items
is large bottle to be bagged
is bag with < 6 large items
THEN put the bottle in the bag

rule b2
IF the step is bag-large-items
is large item to be bagged
is bag with < 6 large items
THEN put the large item in the bag

rule b3
IF the step is bag-large-items
is large item to be bagged
THEN start a fresh bag
rule b4
IF the step is bag-large-items
THEN delete step is bag-large-items
add step is bag-medium-items
Some conflict resolution strategies

- Rule ordering
- Context limiting (organize rules in groups active at different times)
- Specificity/size
- Data ordering
- Recency ordering (least recently used)
- Metarules
- Preference analyzer
We need to provide background knowledge …

<table>
<thead>
<tr>
<th>item</th>
<th>container</th>
<th>size</th>
<th>frozen?</th>
</tr>
</thead>
<tbody>
<tr>
<td>pepsi</td>
<td>bottle</td>
<td>L</td>
<td>N</td>
</tr>
<tr>
<td>granola</td>
<td>box</td>
<td>L</td>
<td>N</td>
</tr>
<tr>
<td>coke</td>
<td>bottle</td>
<td>L</td>
<td>N</td>
</tr>
<tr>
<td>turkey</td>
<td>plastic</td>
<td>L</td>
<td>Y</td>
</tr>
</tbody>
</table>

And starting state …
((bag 1 0)
 (num-bags 1)
 (step bag-large-items)
 (item granola box large)
 (item pepsi bottle large)
 (item turkey plastic medium)
 (frozen turkey yes)
 (item coke bottle large)
 (item cereal box large)
 (item ice-cream box medium)
 (frozen ice-cream yes)
 (item detergent box large)
 (unbagged pepsi)
 (unbagged coke)
 (unbagged cereal)
 (unbagged ice-cream)
 (unbagged detergent)
 (unbagged turkey)
 (unbagged granola))
...
Some Bagger Operators

One test:

- **Find** succeeds if it can find an item in WM matching a given pattern (can combine with **not**)

A few actions:

- **Add** places a new item in WM
- **Remove** deletes an item in WM
- **Set** sets a variable to a new value
- **Stop** terminates processing
Some Bagger Rules

((1 ((find (step bag-large-items))
    (find (item ?item bottle large))
    (find (unbagged ?item))
    (find (bag ?bag ?number))
    (do (#<procedure < > ?number 6)))
((remove (unbagged ?item))
  (remove (bag ?bag ?number))
  (set ?number (#<procedure 1+ > ?number))
  (add (bag ?bag ?number))
  (add (location ?item ?bag))))
(2 ((find (step bag-large-items))
   (find (item ?item ?container large))
   (find (unbagged ?item))
   (find (bag ?bag ?number))
   (do (#<procedure < > ?number 6)))
   ((remove (unbagged ?item))
   (remove (bag ?bag ?number))
   (set ?number (#<procedure 1+ > ?number))
   (add (bag ?bag ?number))
   (add (location ?item ?bag))))
(3 ((find (step bag-large-items))
   (find (item ?item ?container large))
   (find (unbagged ?item)))
((find (num-bags ?number))
 (remove (num-bags ?number))
 (set ?number (#<procedure 1+ > ?number))
 (add (num-bags ?number))
 (add (bag ?number 0))))

(4 ((find (step bag-large-items)))
 ((remove (step bag-large-items))
  (add (step bag-medium-items))))
(5 ((find (step bag-medium-items))
  (find (item ?item ?c medium))
  (find (frozen ?item yes))
  (not (find (freezer-bag ?item yes))))
  ((add (freezer-bag ?item yes))))

(6 ((find (step bag-medium-items))
  (find (item ?item ?c medium))
  (find (unbagged ?item))
  (find (bag ?bag ?number))
  (do (#<procedure < > ?number 6)))
  ((remove (unbagged ?item))
   (remove (bag ?bag ?number))
   (set ?number (#<procedure 1+> ?number))
   (add (bag ?bag ?number))
   (add (location ?item ?bag))))
(7 ((find (step bag-medium-items))
   (find (item ?item ?c medium))
   (find (unbagged ?item)))
   ((find (num-bags ?number))
   (remove (num-bags ?number))
   (set ?number (#<procedure 1+ > ?number))
   (add (num-bags ?number))
   (add (bag ?number 0))))

(8 ((find (step bag-medium-items)))
   ((stop))))
A sample run

(With limit of 3 items per bag.)

initial conditions
step
-----

bag-large-items

bag 1
-----

unbagged
-------

pepsi
coke
cereal
ice-cream
detergent
turkey
granola

frozen items in freezer bag?  
-----------------------------------
turkey no
ice-cream no

rules triggered: (4 3 2 1)
rule fired: 1
ACTION: (remove (unbagged pepsi))
ACTION: (remove (bag 1 0))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 1 1))
ACTION: (add (location pepsi 1))

rules triggered: (4 3 2 1)
rule fired: 1
ACTION: (remove (unbagged coke))
ACTION: (remove (bag 1 1))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 1 2))
ACTION: (add (location coke 1))
rules triggered: (4 3 2)
rule fired: 2
ACTION: (remove (unbagged granola))
ACTION: (remove (bag 1 2))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 1 3))
ACTION: (add (location granola 1))
rules triggered: (4 3)
rule fired: 3
ACTION: (find (num-bags ?number))
ACTION: (remove (num-bags 1))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (num-bags 2))
ACTION: (add (bag 2 0))
rules triggered: (4 3 2)
rule fired: 2
ACTION: (remove (unbagged cereal))
ACTION: (remove (bag 2 0))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 2 1))
ACTION: (add (location cereal 2))
rules triggered: (4 3 2)
rule fired: 2
ACTION: (remove (unbagged detergent))
ACTION: (remove (bag 2 1))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 2 2))
ACTION: (add (location detergent 2))
rules triggered: (4)
rule fired: 4
ACTION: (remove (step bag-large-items))
ACTION: (add (step bag-medium-items))
rules triggered: (8 7 6 5)
rule fired: 5
ACTION: (remove (freezer-bag turkey no))
ACTION: (add (freezer-bag turkey yes))
rules triggered: (8 7 6 5)
rule fired: 5
ACTION: (remove (freezer-bag ice-cream no))
ACTION: (add (freezer-bag ice-cream yes))
rules triggered: (8 7 6)
rule fired: 6
ACTION: (remove (unbagged turkey))
ACTION: (remove (bag 2 2))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 2 3))
ACTION: (add (location turkey 2))
rules triggered: (8 7)
rule fired: 7
ACTION: (find (num-bags ?number))
ACTION: (remove (num-bags 2))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (num-bags 3))
ACTION: (add (bag 3 0))
rules triggered: (8 7 6)
rule fired: 6
ACTION: (remove (unbagged ice-cream))
ACTION: (remove (bag 3 0))
ACTION: (set ?number (proc:1+ ?number))
ACTION: (add (bag 3 1))
ACTION: (add (location ice-cream 3))
rules triggered: (8)
rule fired: 8
ACTION: (stop)
execution terminated

final state
step
----

bag-medium-items

bag 3
-----

ice-cream

bag 2
-----
turkey
detergent
cereal

bag 1
-----
granola
coke
pepsi

unbagged
-------

frozen items in freezer bag?
------------------------

ice-cream yes
turkey yes
XCON (McDermott, 1982) configures VAXes for DEC

- 10,000 rules
- Routinely handles orders for 100-200 components

A sample rule:

**IF** context is doing layout
and assigning power supply
an sbi module has been put
in cabinet
position of sbi module is known
there is space for power supply

**THEN** put power supply in the
available space
The algorithm

Determining whether rules apply requires unification.

(unify
   '(find (item ?item bottle large))
   '(item pepsi bottle large))
   ()
=>
((item pepsi))

(unify
   '(find (item ?item ?type large))
   '(item pepsi bottle large))
   '(()(type bottle))
=>
((item pepsi)(type bottle))
unify:

Params: pat1, pat2, substitution
The substitution is a set of variable bindings, e.g., ((x 3) (y z) (z (drop and add)))

If pat1 and pat2 are equal?,
    return substitution
Else if pat1 or pat2 is a var, call unify-var
Else if either pat1 or pat2 is an atom, return #f
Else the patterns are both lists.
Call unify on correspond. parts of pat1 and pat2.
    if #f results, return #f immediately.
    else a substitution results;
        replace old subst with new and continue.
Return the final substitution.
**unify-var:**

Params: var, pat, substitution

If var has a binding in substitution,
    call unify for the binding’s value and pat
Else apply the substitution to pat.
    If var appears anywhere in the result
    of the substitution, return #f.
Else add the binding (var = pat) to
    substitution, and return the new
    substitution.
Forward Chaining, Depth-First

In searching for a substitution that works, search states consist of:
- the antecedents left to satisfy
- the current variable bindings (the substitution).

For each rule,

First check for goal state:
   If a state has no more antecedents to satisfy,
   a goal state has been reached. Add the consequent instantiated with the substitution to working memory.
Else extend:

Let antec be the instantiation (using the substitution) of the first antecedent.

For each assertion in working-memory,

If antec unifies with assertion given substitution, create a new state with the remaining antecedents, the consequent, and the new substitution resulting from the unification.

Else return #f.

If all assertions resulted in #f, return (). Else return the new states.