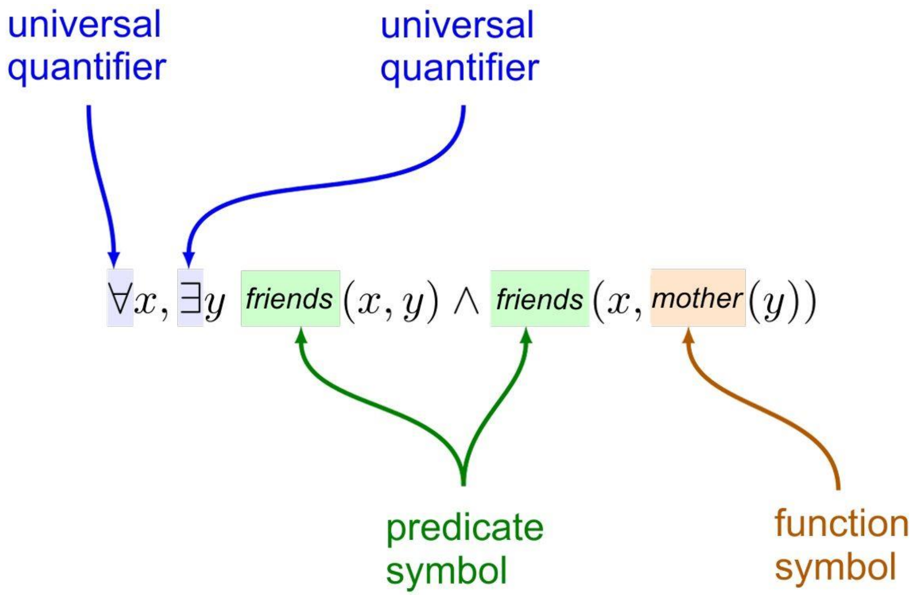
Grado en ingeniería informática
Artificial Intelligence 2021/2022

3.5 Forward and backward chaining: Production systems

Lara Quijano Sánchez



Universidad Autónoma
de Madrid

Lesson's Scheme

- 3. Predicate logic
  - 3.1 Elements of predicate logic
    - 3.1.1. Common elements with propositional logic
    - 3.1.2. Variables and quantifiers
    - 3.1.3. Predicates
    - 3.1.4. Features
    - 3.1.5. Atoms, terms, literals and clauses
    - 3.1.6. Normal shapes
  - 3.2 Substitution and unification
  - 3.3 Inference in predicate logic
    - 3.3.1 Generalized rules of inference
      - 3.3.1.1 Modus ponens
      - 3.3.1.2 Resolution
    - 3.3.2 Extraction of responses using Green's trick.
  - 3.4 The equality predicate
  - 3.5 Forward and backward chaining

Production systems

- Problems with
  - fixed control flow
  - sequential execution
  - not adequate in dynamic environments
- Solution: data driven operations

Components of a Rule-based System

- Facts set or working memory: domain knowledge at any given moment
- Rules set: set of rules (productions)
  - If A THEN B
  - A: conditions of application
  - B: actions on the working memory or outside world
- Control strategy, rule interpreter, or inference engine: responsible for chaining the rules execution

Control strategy

- Facts: P, R
- Rules:
  - R1. If P Then J
  - R2. If R Then K,L
  - R3. If P and R Then L
  - R4. If L Then F

- Which one does the system execute?
- Facts: P, R
- Program: If P Then J

  Else If R Then K,L

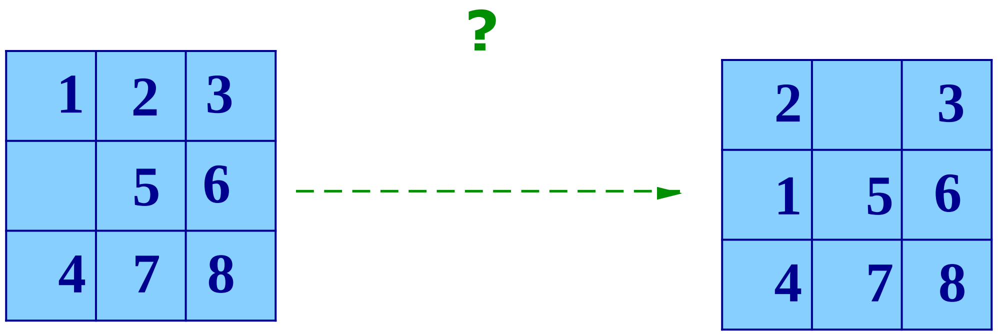
  Else If P and R Then L

  Else If L Then F

Components

- Facts:
  - can be represented using any type of representation paradigm: simple facts, objects, logic, . . .
- Rules:
  - no disjunction → n rules
  - no if-then-else → two rules
  - no explicit reference on the action part of a rule to another rule (only through facts)
  - if then-part only adds: monotonic reasoning
- Inference:
  - several cycles: at each cycle one (or more rules) are selected and applied
  - refraction: in most domains the same rule should not fire with the same values for its variables

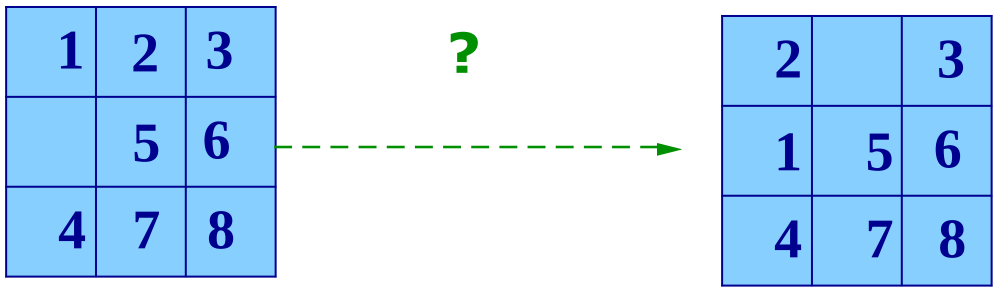
Example: 8 puzzle. Working memory



- lists: (V11,V12,V13,…,V33)
- predicate logic: square(X,Y,Value)
- objects:

Square	
is-a:	
Attribute	**Possible values/Value**
x.	number [1..3]
y.	number [1..3]
value	number [0..8]

8-puzzle. Initial working memory

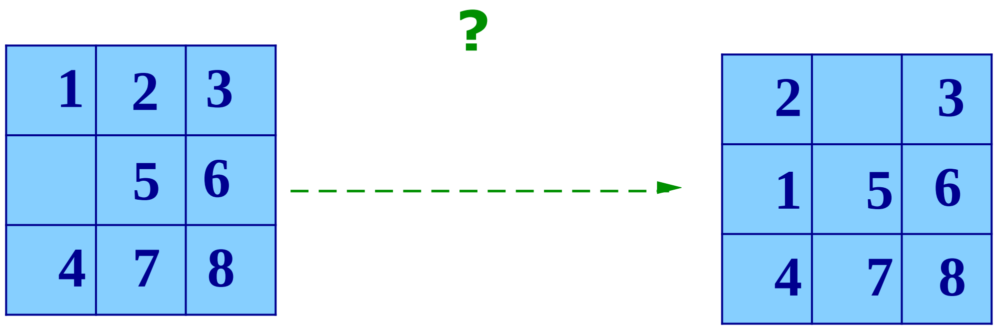


- lists: (1,2,3,0,5,6,4,7,8)
- predicate logic: square(1,1,1),square(2,1,2),. . .,square(3,3,8)
- objects:

Square11	
instance-of: Square	
Attribute	**Possible values/Value**
x. 1	
y. 1	
value 1	

Square21	
instance-of: Square	
Attribute	**Possible values/Value**
x. 2	
y. 1	
value 2	

8-puzzle. Final working memory or goals



- lists: (2,0,3,1,5,6,4,7,8)
- predicate logic: square(1,1,2),square(2,1,0),. . .,square(3,3,8)
- objects:

Square11	
instance-of: Square	
Attribute	**Possible values/Value**
x. 1 y. 1 value 2	

Square21	
instance-of: Square	
Attribute	**Possible values/Value**
x. 2 y. 1 value 0	

8-puzzle. Rule base

- Lists

If(0,X1,X2,X3,X4,X5,X6,X7,X8)
Then(X1,0,X2,X3,X4,X5,X6,X7,X8)

If(0,X1,X2,X3,X4,X5,X6,X7,X8)
Then(X3,X1,X2,0,X4,X5,X6,X7,X8)

- Problem: involves identifying all the possible combinations of the empty position (0) and its possible moves

8-puzzle. Predicate logic

If square(X,Y,0),square(X1,Y,Z),X=X1+1
Then square(X1,Y,0),square(X,Y,Z), ~square(X,Y,0),~square(X1,Y,Z)

If square(X,Y,0),square(X1,Y,Z),X=X1-1
Then square(X1,Y,0),square(X,Y,Z), ~square(X,Y,0),~square(X1,Y,Z)

If square(X,Y,0),square(X,Y1,Z),Y=Y1+1
Then square(X,Y1,0),square(X,Y,Z), ~square(X,Y,0),~square(X,Y1,Z)

If square(X,Y,0),square(X,Y1,Z),Y=Y1-1
Then square(X,Y1,0),square(X,Y,Z), ~square(X,Y,0),~square(X,Y1,Z)

Alternative representation:

If On(x,y), Free(z), Adjacent(y,z)
Then On(x,z), Free(y), NOT On(x,y), NOT Free(z)

It is difficult to do in PROLOG, given that it requires non-monotonic reasoning

Operation of a PS

- Types of systems
  - Forward chaining
    - If P, then Q. P. Therefore, Q
  - Backward chaining
    - If P, then Q. Q?. P?
- Phases
  - Decision phase
    - Reduction (optional)
    - Matching: conflict set, RETE
    - Conflict set resolution
  - Execution phase
- Termination
  - Some specific fact (set of facts) is true (false)
  - No more rules can be executed
  - A given number of rules have been executed
  - A halt signal is issued by a rule

Matching

- First approach: computing and resolving the conflict set at each cycle
- Problem: slow
- Solution: RETE algorithm (algorithm of temporary redundancy)
  - initially it establishes a graph from the rules (RETE network)
  - it propagates the contents of the initial facts base through the network
  - every time a change in the facts base arises (usually, through
  - the consequent of a rule), the changes are propagated
  - at every cycle, a conflict set will be available at the end nodes of the network
- Key idea: structural similarity

Forward chaining

Initial WM, WM_0: A, B, C

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Forward chaining

Initial WM, WM_0: A, B, C

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching:

Forward chaining

Initial WM, WM_0:A, B, C

Rule set:R1. If A, B Then D, E, not C

R2. If A Then F,not B

R3. If C, B Then G

R4. If E, F,G Then H

Matching: R1, R2, R3

Forward chaining

Initial WM, WM_0:A, B, C

Rule set:R1. If A, B Then D, E, not C

R2. If A Then F,not B

R3. If C, B Then G

R4. If E, F,G Then H

Matching: R1, R2, R3

Conflict Set Resolution (for example, first rule):R1

Forward chaining

Initial WM, WM_0: A, B, C

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R1, R2, R3

Conflict Set Resolution (for example, first rule): R1

Execution: + D, E, - C

WM_1: A, B, D, E

Forward chaining

Initial WM, WM_0: A, B, C

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R1, R2, R3

Conflict Set Resolution (for example, first rule): R1

Execution: + D, E, - C

WM_1: A, B, D, E

Matching: R1, R2

Forward chaining

Initial WM, WM_0: A, B, C

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R1, R2, R3

Conflict Set Resolution (for example, first rule): R1

Execution: + D, E, - C

WM_1: A, B, D, E

Matching: R1, R2

Refraction: R1 cannot be used again. Conflict set: R2

Forward chaining

Initial WM, WM_0: A, B, C

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R1, R2, R3

Conflict Set Resolution (for example, first rule): R1

Execution: + D, E, - C

WM_1: A, B, D, E

Matching: R1, R2

Refraction: R1 cannot be used again. Conflict set: R2

Conflict Set Resolution: R2

Execution: + F, - B

WM_2: A, D, E, F

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching:

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Conflict Set Resolution: R4

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Conflict Set Resolution: R4

Execution: + E?, F?, G?, - H?

Goals: E, F, G

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Conflict Set Resolution: R4

Execution: + E?, F?, G?, - H?

Goals: E, F, G

Matching:

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Conflict Set Resolution: R4

Execution: + E?, F?, G?, - H?

Goals: E, F, G

Matching: R1, R2, R3

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Conflict Set Resolution: R4

Execution: + E?, F?, G?, - H?

Goals: E, F, G

Matching: R1, R2, R3

Conflict Set Resolution (first rule): R1

Execution: + A?, B?

True in WM_0

If we execute now R1, WM_1: A, B, D, E

Goals: F, G

Backward chaining

Initial WM, WM_0: A, B, C

Goals: H?

Rule set: R1. If A, B Then D, E, not C

R2. If A Then F, not B

R3. If C, B Then G

R4. If E, F, G Then H

Matching: R4

Conflict Set Resolution: R4

Execution: + E?, F?, G?, - H?

Goals: E, F, G

Matching: R1, R2, R3

Conflict Set Resolution (first rule): R1

Execution: + A?, B?

True in WM_0

If we execute now R1, WM_1: A, B, D, E

Goals: F, G

Backward chaining (like PROLOG)

Goals:(square11 (value 2))

(square21 (value 0))

...

(square33 (value 8))

Reduction:goal selection (for example, the first)

(square11 (value 2))

Matching:

(Up, ?v=2, ?square=#square11)

(Down, ?v=2, ?square=#square11)

(Right, ?v=2, ?square=#square11)

(Left, ?v=2, ?square=#square11)

Resolution of Conflict Set (for example, first rule):

(Up, ?v=2, ?square=#square11)

Backward chaining (like PROLOG)

Execution:introduce the conditions of the instantiated rule on the set of goals

?square=#square11 and (?square ← (square (x ?x) (y ?y) (value 0)))
then ?x=1, ?y=1
and adds goal (square11 (value 0))
?v=2 and (?square1 ← (square (x ?x) (y ?y1) (value ?v)))
then (?square1 ← (square (x 1) (y ?y1) (value 2)))
(test ?y=?y1+1), ?y=1 and (?square1 ← (square (x 1) (y ?y1) (value 2)))
then ?square1=#square21 and ?y1=0!!!
and adds goal (square21 (value 2))
The list of goals is: (square11 (value 0))
(square21 (value 2))
(square21 (value 0))
. ..
(square33 (value 8))

Reduction:. ..

Comparing chaining modes

- Disadvantages of forward chaining
  - does not focus on goals
  - initially all data should be in working memory
  - greater amount of comparisons
- Disadvantages of backward chaining
  - handling goals and subgoals is needed
  - actions that solve the problem are unknown until the very end
- Choosing one
  - number of initial and goal states
  - branching factor
  - justifications needed

Strategy characteristics

- The most general possible
- The most efficient possible (heuristics): implicit or explicit
- Change state
- Be systematic

Resolution strategies

- First rule
- More knowledge
- Greater priority
- More specific
- More general
- Considering the newest element
- No prior execution
- More executions
- Randomly
- Explore all
- Meta rules
- Mixed strategies

Advantages of production systems

- It is natural for experts to express knowledge as rules
  - If patient has fever and sneezes then diagnosis is flu
- There is already a formal analysis of rule-based knowledge (as in logic)
- If rules apply, they generate new knowledge, so that new rules can fire
  - If patient has flu then treatment is stay at home
- A set of rules can be easily mantained by adding or removing rules

Advantages and disadvantages

- Advantages
  - modularity, which facilitates incremental growth
  - declarative character
  - uniformity
  - naturalness
  - flexibility
  - learning
  - modeling of animal and human behavior
- Disadvantages
  - inefficient
  - opacity
  - difficult to represent algorithms

Going to the real world

- Applications
  - Expert systems: medicine, oil discovery, computers configuration, risks analysis, . . .
  - Microsoft problem solving
  - Business rules
  - Laws, vaccination, telecommunications
  - Control
  - Games
- Tools
  - Academic: OPS V, Frulekit
  - Professional: Web Sphere (IBM), Business rules (Oracle), CLIPS/JESS (NASA)
  - Others: RuleML (http://www.ruleml.org/), Prolog

Prolog

- Declarative programming language
- Based on predicate logic
- Example:

```
number-accesses(johnsmith,15).

number-accesses(anntaylor,34).

total-spent(johnsmith,300).

total-spent(anntaylor,50).
good-client(X) :- often-access(X), medium-expenses(X).
good-client(X) :- medium-access(X), high-expenses(X).
low-access(X) :- number-accesses(X,Y), Y < 10.
medium-access(X) :- number-accesses(X,Y), Y >= 10, Y < 30.
often-access(X) :- number-accesses(X,Y), Y >= 30.
low-expenses(X) :- total-spent(X,Y), Y < 100.
medium-expenses(X) :- total-spent(X,Y), Y >= 100, Y < 300.
high-expenses(X) :- total-spent(X,Y), Y >= 300.
```

Extra examples