Pattern Matching

© Gunnar Gotshalks

Pattern Matching

◊ A ubiquitous function for intelligence

  » **IQ tests, for example, contain pattern matching problems**

    > **They are recognized as an important class of problem that people deal with.**

Pattern Matching – 2

◊ Pattern matching means to compare one object with another object and recognize if they are similar

Pattern Matching – 3

- Pattern matching means to compare one object with another object and recognize if they are similar

  - **Basic case is comparing constants**

© Gunnar Gotshalks

Pattern Matching – 4

- ◊ Pattern matching means to compare one object with another object and recognize if they are similar

  - » **Basic case is comparing constants**

  - » **More interesting is to compare parameterized patterns**

Pattern Matching – 5

- Pattern matching means to compare one object with another object and recognize if they are similar
    - **Basic case is comparing constants**
    - **More interesting is to compare parameterized patterns**
        - **A is like B except for ....**

© Gunnar Gotshalks

Pattern Matching – 6

- ◊ Pattern matching means to compare one object with another object and recognize if they are similar
  - » **Basic case is comparing constants**
  - » **More interesting is to compare parameterized patterns**
    - > **A is like B except for ....**
    - > **A is like B where …**

What is a pattern?

- ◊ A pattern is a collection that contains
  - » **Constants – called literals**
  - » **Variables that take on patterns as values**

© Gunnar Gotshalks

What is a pattern? – 2

- A pattern is a collection that contains
  - **Constants – called literals**
  - **Variables that take on patterns as values**
- We need a syntax to differentiate the two
  - **How in Prolog?**

© Gunnar Gotshalks

What is a pattern in Prolog?

- ◊ A pattern is a **compound term** that contains
  - » **Constants – called literals**
  - » **Variables that take on patterns as values**

- ◊ Variables begin with an upper case letter
  - » **for example X Abc**

- ◊ Constants begin with a lower case letter
  - » **for example x abc**

© Gunnar Gotshalks

What is a pattern in Prolog? – 2

- ◊ An abstract pattern could look like
  - » `[ a , b , X , c ,Y ]`

What is a pattern in Prolog? – 3

◊ An abstract pattern could look like

  » **[a , b , X , c ,Y]**

◊ A more meaningful pattern could be

  » **causes (hit (X , Y) , (hurt (Y))**

    > **Interpreted as – X hitting Y, causes Y to be hurt**

When do two patterns match?

◊ Two patterns can be matched when it is possible to **unify** them

© Gunnar Gotshalks

When do two patterns match? – 2

- ◊ Two patterns can be matched when it is possible to **unify** them
- ◊ **Unification** means an assignment can be made to the variables in each pattern such that the patterns become identical.

© Gunnar Gotshalks

When do two patterns match? – 3

- ◊ Two patterns can be matched when it is possible to **unify** them
- ◊ **Unification** means an assignment can be made to the variables in each pattern such that the patterns become identical.
  - » **We usually mean the most general possible assignment**

When do two patterns match? – 4

- ◊ Two patterns can be matched when it is possible to **unify** them
- ◊ **Unification** means an assignment can be made to the variables in each pattern such that the patterns become identical.
  - » **We usually mean the most general possible assignment**
    - > **MGU = Most General Unifier**

When do two patterns match? – 5

◊ Two patterns can be matched when it is possible to **unify** them

◊ **Unification** means an assignment can be made to the variables in each pattern such that the patterns become identical.

  » **We usually mean the most general possible assignment**

    > **MGU = Most General Unifier**

◊ An assignment is shown by a tuple variable = value

  » **X = abc**

  » **X = Y**

© Gunnar Gotshalks

Unification Examples – 1

- [a , X , b]
  [a , y , b]

  match if X = y
  we say that X is bound to y

- [a , X , b]
  [a , Y , b]

  match if X = Y

- [a , X , [b , Z]]
  [a , [[[e]]] , Y]

  match if X = [[[e]]]
  Y = [b , Z]

© Gunnar Gotshalks

Unification Examples – 2

- ◊ More complex examples
  - » [a , X , X] match if X = Y
    [a , Y , c] and Y = c
    - > Cannot naively bind X to Y and then X to c as then we are trying to assign two different values to X need to substitute Y for X and then see that Y binds to c
  - » [a , X , X , X]
    [a , Y , Y , Y]
    - > Cannot naively try to bind X to Y , as on the second attempt, we end up binding Y to Y , then on the third attempt, we have an infinite loop

© Gunnar Gotshalks

Unification Examples – 3

- More complex examples

  -
    ```
    [a ,  X  ,  X    ]
    [a ,  Y ,  [b , Y] ]
    ```

    There is no consistent binding to make a match

    - **Again need to prevent an infinite loop**

Matcher

◊ The function match takes place with a binding list that begins as empty

```
match  ( pattern1 , pattern2 ) :=
  match-with-bindings ( pattern1 , pattern2 , [ ] )
```

© Gunnar Gotshalks

Pattern matcher output

- ◊ Need to distinguish three cases

© Gunnar Gotshalks

Pattern matcher output – 2

- Need to distinguish three cases
  - **No match is possible**

Pattern matcher output – 3

- Need to distinguish three cases
  - **No match is possible**
    - **output is False**

© Gunnar Gotshalks

Pattern matcher output – 4

◊ Need to distinguish three cases

- » **No match is possible**
  - > **output is False**

- » **Match is possible but no variable bindings are required**

Pattern matcher output – 5

- Need to distinguish three cases
  - **No match is possible**
    - **output is False**
  - **Match is possible but no variable bindings are required**
    - **output is True**

© Gunnar Gotshalks

Pattern matcher output – 6

- ◊ Need to distinguish three cases
  - » **No match is possible**
    - > **output is False**
  - » **Match is possible but no variable bindings are required**
    - > **output is True**
  - » **Match is possible with variable bindings**
    - > **output is list of bindings of variables in the query**
    - > **a binding is a pair variable = value**

© Gunnar Gotshalks

Pattern matcher output – 7

◊ Example with a binding required

```
» match ( [ a , X , c , Y , e]
         , [ a , b , Z , d , e] )

  > [Y = d ,  Z = c , X = b]
```

© Gunnar Gotshalks

Matching cases

- Matching two patterns requires a recursive descent into the patterns to match sub-patterns

  - **The following cases can occur**

    - **Pattern1 – a variable, a constant, a ct (compound term)**
    - **Pattern2 – a variable, a constant, a ct**

Matching cases – 2

◊ The matching program has to examine the possible combinations

Pattern1	Pattern2	Result
constant	constant	match if equal, else no match
constant	variable	try to bind constant to variable
constant	ct	no match
variable	constant	try to bind constant to variable
variable	variable	try to bind variable to variable
variable	ct	try to bind ct to variable
ct	constant	no match
ct	variable	try to bind ct to variable
ct	ct	recursive descent into both ct's

© Gunnar Gotshalks