Matching

Once we have objects, it is useful to compare them. The mechanism to do so is called **matching** in Prolog, and is different from the equalities we find in other programming languages.

- Two numbers match if they represent the same mathematical number.
- Two atoms match if they are made of the same characters.
- A variable matches any object.
- Two structures match if
  - their functors match (as atoms),
  - all their corresponding arguments match.

Matching (cont)

Matching and equality agree on **ground terms**, i.e. objects containing no variables. In this case, they both either return true (`Yes` in Prolog) or false (`No`).

The difference between matching and equality concerns **non-ground terms**, i.e. objects containing variables.

Consider the following fragment of a C program:

```
if (x == 5) x++;
```

Here, the run-time has to compare *the value of* `x` with `5`, i.e. it looks up the value to which `x` is bound and then matches it against `5` (remember that equality and matching agree on ground terms).

Matching (cont)

In Prolog, the closest clause, as far as comparison is concerned, is the query

```
?- X = 5.
```

But the scoping rules of Prolog say that this occurrence of variable `X` is visible only in this clause. Therefore it is unbound, i.e., it is not associated to any “previous” value.

Instead, because `X` is a variable, it must match `5`, so the interpreter answers

```
X = 5
Yes
```

Here, the successful matching returns a **binding** for `X`, before answering `Yes`.

Matching (cont)

Imagine now a matching involving a structure, like

```
?- date(D,july,2006) = date(9,M,2006).
```

The interpreter first checks whether the functors match (are equal), which is true. Next, it matches the corresponding arguments against each other: `D` against 9, `july` against `M` and 2006 against 2006.

The first matching involves a variable and a number, so the interpreter chooses the binding `D = 9`.

The second matching involves an atom and a variable, so the interpreter chooses the binding `M = july`.

The last matching is trivial, `2006 = 2006`, and does not require any binding.

Matching (cont)

So the answer is

```
D = 9
M = july
Yes
```

In other words, a successful matching returns bindings for all variables in the terms being matched, such as the corresponding **instantiation** leads to equal ground terms:

```
date(9,july,2006) = date(9,july,2006)
```

Instantiation means to replace all the variables by the object to which they are bound.

Matching/Failure

A failed matching consists of only `No`. For example

```
?- 1 = 2.
No
?- date(9,july,2006) = date(9,july,2007).
No
?- date(X,july,2006) = date(9,july,X).
No
```

In the last case, the interpreter finds two bindings for `X` which are different: `X`=9 and `X`=2006, which leads to failure.

Matching/Most general substitution

In general, a successful matching returns several bindings. A set of bindings is called a **substitution**. So, an instantiation consists in applying a substitution to a clause.

Sometimes there can be several possible substitutions that make the matching a success. For example the matching

```
?- X = Y.
```

can be satisfied by `X = -3, Y = -3` or `X = 7, Y = 7` and so on.

Matching/Most general substitution (cont)

In such cases, Prolog ensures that the most general substitution will be retained. In our example, `X = Y` is the most general, because all the instances can be obtained from it by replacing `X` and `Y` by the same arbitrary object.

In other words, when matching a variable `A` against another variable `B`, the matching succeeds with the most general substitution `A = B`, or

```
A = _G187
B = _G187
```

where `_G187` is a variable generated by the interpreter. In these slides we prefer to write

`A =` α
`B =` α

Matching/Most general substitution (cont)

Consider the special case

```
?- X = X.
X = α
Yes
```

The danger here is that Prolog uses the same syntactical convention, the = character, to denote both variable bindings and matchings: `A = 2` can be a matching or a binding, same for `A = B`.

But `2 = A` is a matching but **not** a variable binding. Same for

```
date(9,july,2006) = date(9,july,2006)
```

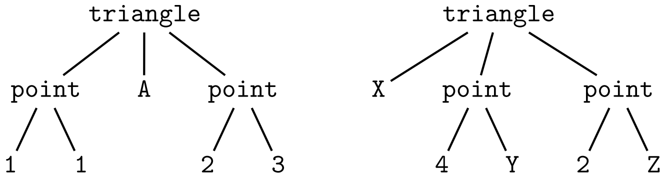
Matching/Tree representation

It is useful to use the tree representation of Prolog terms (page 58) to understand how a matching is performed.

Consider the two terms

```
triangle(point(1,1),A,point(2,3))
triangle(X,point(4,Y),point(2,Z))
```

These terms are represented by the trees



Matching/Tree representation (cont)

The interpreter traverse the two trees from the root to the leaves, following the same order when visiting the sub-trees. Let us assume that order between siblings is from left to right.

It matches first the two roots: if one of them is a variable, it stops and declares success, otherwise it matches the subtrees. Here, `triangle = triangle`, so, next, it matches the first subtree of the first tree with the first subtree of the second tree, i.e. `?- point(1,1) = X.` This is a success with the substitution `X = point(1,1)`.

Then, the second subtrees are matched, i.e.,

`?- A = point(4,Y). A = point(4,`α`) Y = `α

Matching/Tree representation (cont)

Next, the last subtrees are matched, i.e. the interpreter tries to answer the query

```
?- point(2,3) = point(2,Z).
```

The roots are the same: `point = point`. So, it then matches the subtrees, i.e. answers now the queries

```
?- 2 = 2.    ?- 3 = Z.
```

successfully with substitution `Z = 3`. Finally the answer is the substitution which is the union of all the others:

X = point(1,1)
A = point(4,α)
Y = α
Z = 3

Matching/Tree representation (cont)

The proof of the matching can be displayed by means of a proof tree:

$$
\dfrac{\texttt{point(1,1) = X} \qquad \texttt{A = point(4,Y)} \qquad \dfrac{\texttt{2 = 2} \qquad \texttt{3 = Z}}{\texttt{point(2,3) = point(2,Z)}}}{\begin{array}{c}\texttt{triangle(point(1,1),A,point(2,3))} \\ \texttt{= triangle(X,point(4,Y),point(2,Z))}\end{array}}
$$