# IKI30820 – Logic Programming
# Syntactic Aspects of Prolog Programs
# Slide 02

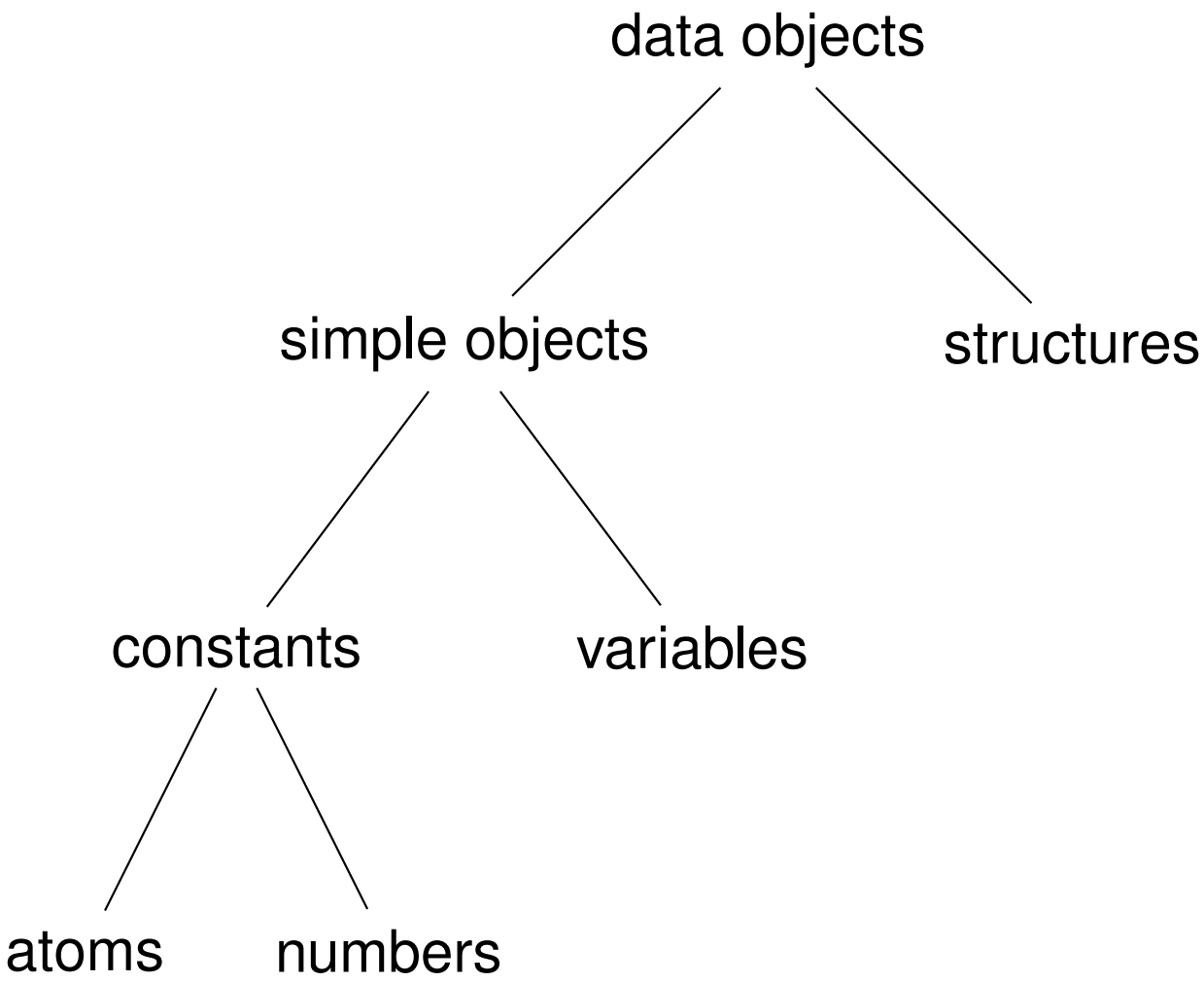
Ari Saptawijaya (slides)  Adila A. Krisnadhi (LaTeXadaptation)

Fakultas Ilmu Komputer
Universitas Indonesia

2009/2010 • Semester Gasal

# Outline

# Data objects in Prolog



# Atoms

- Strings of letters, digits and the underscore character ( _ ), starting with a lower case letter:
  - `jakarta`, `fasilkom_UI`, `x20`, `nil`, `a_b_c`, `k_2n`
- Strings of special characters:
  - `<--->`, , `======>`, `...`, `.:.`, `::=`
  - Be careful: some strings of special characters have a predefined meaning, e.g., `:=`, `=:=`
- Strings of characters enclosed in single quotes:
  - `'kuliah_prolog'`,`'DKI_Jakarta'`,`'Kubuntu Linux'`

# Numbers

- Integer numbers
  - Example: 1, 98315, 32, 0, -85
  - Range of integers depends on implementation; see the documentation.
- Real numbers
  - Example: 3.14, -0.000123, 23.1
  - Also depends on implementation.
  - Real numbers are not used very much in typical Prolog programming because Prolog is primarily a language for symbolic, nonnumeric computation (cf. Fortran).

# Built-in predicates

- **`atom`**`(X)`
  is true if X currently stands for an atom.
- **`integer`**`(X)`
  is true if X currently stands for an integer.
- `float(X)`
  is true if X currently stands for a real number.
- `number(X)`
  is true if X currently stands for a number.
- **`atomic`**`(X)`
  is true if X currently stands for an atom or a number.

```
?- atom(jakarta).
Yes.
?- atom(3.14).
No.
?- atom(<---->)
Yes.
?- integer(-23).
Yes.
?- integer(3.14).
No.
?- float(3.14).
Yes.
?- float(23).
No.
?- atomic(fasilkom).
Yes.
?- atomic(-2.12).
Yes.
```

# Variables

- Strings of letters, digits, and underscore characters, starting with an **uppercase** letter or an **underscore** character.
  - Example: `X`, `Object2`, `ShoppingList`, `_x11`, `_jMan`
- A **single underscore** character represents an **anonymous** variable.
  - Used when a variable appears in a clause once only.
  - Two anonymous variables in the same clause represents *different* variables.

# Anonymous variables I

- Define a rule `hasachild/1`:
  For all X, X has a child if X is a parent of some Y.

```
hasachild(X) :- parent(X,Y).
```

- In the rule `hasachild/1`, the variable Y out of interest, hence we use anoynmous variable.

```
hasachild(X) :- parent(X,_).
```

# Anonymous variables II

- Each time a single underscore occurs in a clause, it represents a new anoynmous variable.

```
somebody_hasachild :- parent(_,_).
```

  Somebody has a child if there are two objects s.t. one is a parent of the other.

- It is equivalent to:

```
somebody_hasachild :- parent(X,Y).
```

- It is **NOT** equivalent to

```
somebody_hasachild :- parent(X,X).
```

# Anonymous variables III

- If anonymous variable appears in question/query clause, then its value is not output when Prolog answers the question:

  ```
  ?- parent(X,_).
  ```

  outputs only the name of the parents without giving the name of corresponding children.

# Lexical scope of variables and atoms

- Lexical scope of variable is one clause.
  - If a variable `W3` occurs in two clauses, it signifies two different variables.
  - But each occurrence of `W3` within the same clause means the same variable.
- On the other hand, the same atom always means the same object in any clause, i.e., its scope is the whole program.

# Built-in predicates

- **var**(X) succeeds if X is currently an uninstantiated variable.
- **nonvar**(X) succeeds if X is not a variable, or X is an already uninstantiated variable.
- Example:

```
?- var(X).
Yes.
?- var(depok).
No.
?- nonvar(X).
No.
?- parent(tukul,X), var(X).
No.
```

the last example fails because either no X that satisfies `parent(tukul,X)`, or the goal `parent(tukul,X)` succeeds and instantiates `X`, thus making **var**(X) fails.

# Structures

- Structured objects (or **structures**) are objects that have several components.
  - The components themselves can, in turn, be structures.
  - Although composed of several components, structures are treated as single objects.
  - To combine the components into a single object, we use **functor**.
  - Example:

    ```
    date(9,sep,2009)
    ```

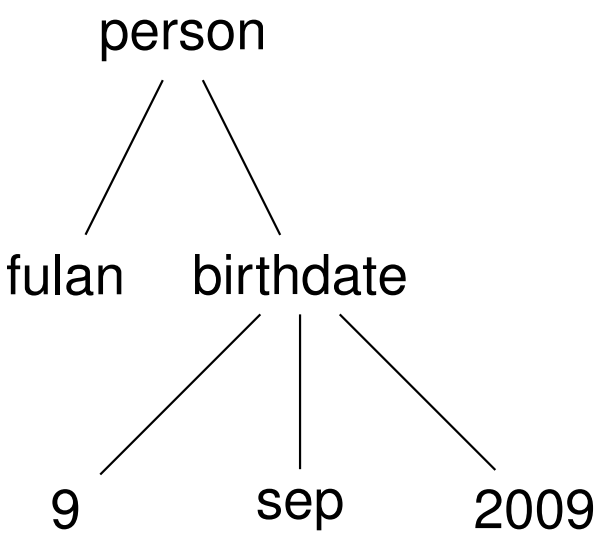
    here, `date` is the functor, whereas `1`, `sep`, and `2009` are the arguments.

# Terms & Ground Terms

- All data objects in Prolog are called **terms**.
  - Example: `date(9,sep,2009)`, `feb`
- Variables are also terms.
- Terms where variables do not occur are called **grounds**. Otherwise, they are called **nonground**.
  - Example:
    `foo(a,b)` is ground,
    `bar(X)` is nonground.

# All structures are trees

- All structures can be seen as a tree.
- The root of the tree is the functor, and the offsprings of the root are the components.
  - If a component of a structure is also a structure, then it is a subtree of the tree that corresponds to the whole structure.
  - The structure `person(fulan, birthdate(9, sep, 2009))` is represented as the tree.



# Functors

- Each functor is defined by two things:
  - the name, whose syntax is like atoms;
  - the arity, i.e., the number of arguments.
- The two functors of the following structures are different:
  - `point(X,Y)`
  - `point(X,Y,Z)`

# Outline

# Matching I

- The most important operation on terms is **matching**.
- Matching is a process that takes two terms as input and checks whether they match.
- Two terms **match** if:
  - They are identical, or
  - The variables in both terms can be instantiated to objects in such a way that after the substitution of variables by these objects, the terms become identical.

# Matching II

- Example:
  Terms `date(D,M,2009)` and (D1,sep,Y1)— match as follows:
  - `D` is instantiated to `D1`
  - `M` is instantiated to `sep`
  - `Y1` is instantiated to `2009`
- The operator = can be used to ask Prolog explicitly to do matching

```
?- date(D,M,2009) = date(D1,sep,Y1)
D = D1,
M = sep,
Y1 = 2009.
```

# Matching III

- Are there other instantiations that make two previous term match?
  - D=26, D1=26, M=sep, Y1=2009
  - D=first, D1=first, M=sep, Y1=2009
  - Etc.
- The instantiation D=D1, M=sep, Y1=2009 is **more general** than the above two instantiations.
  - The above two instantiations constrain the values of D and D1 more strongly than necessary

# Matching IV

- Matching in Prolog always produces the **most general** instantiation.
  - Instantiation that commits the variables to the least possible extent
  - Leaving the greatest possible freedom for further instantiations if further matching is required.
- Example

```
?- date(D,M,2009) = date(D1,sep,Y1),
   date(D,M,2009) = date(9,M,Y).
```

# Unification

- Matching in Prolog corresponds to **unification** in the theory of logic programming
  - Not exactly the same!
  - Unification: **Martelli-Montanari algorithm**

# Martelli-Montanari algorithm

The algorithm unifies finite set of pairs of terms: $\{s_1 = t_1, \ldots, s_n = t_n\}$
It nondeterministically chooses from the set of equations an equation of a form below and perform the associated action until no action applies.

1. $f(s_1, \ldots, s_n) = f(t_1 \ldots, t_n)$
   Replace by the equation $s_1 = t_1, \ldots, s_n = t_n$
2. $f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)$, where $f \neq g$
   Halt with failure
3. $x = x$
   Delete the equation
4. $t = x$, where $t$ is not a variable
   Replace by the equation: $x = t$
5. $x = t$, where $x$ does not occur in $t$ and $x$ occurs elsewhere
   Instantiate every $x$ to $t$ on all other equations
6. $x = t$, where $x$ occurs in $t$ and $t \neq x$
   Halt with failure

# Matching and Unification

- A constant can be seen as a functor with no argument.
- Matching in Prolog vs. unification (Martelli-Montanari algorithm)
  - Matching for rule 2:
    $f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)$, where $f \neq g$ or $n \neq m$
    Halt with failure
  - Matching does not take into account the "occur check" problem. This means omitting the occur check in action (5) and dropping action (6).
- Example:

```
?- date(D,M,2006) = date(D1,sept,Y1),
   date(D,M,2006) = date(26,M,Y).
?- k(Z,f(X,b,Z)) = k(h(X),f(g(a),Y,Z)).
?- date(26,sept) = date(Day, Month, Year)
?- X = f(X).
```