PROLOG.
Constants, Variables, Terms, Atoms, Clauses
Syntax and Semantics

Antoni Ligęza

Katedra Automatyki, AGH w Krakowie

2011
References


Alphabet of Prolog

The alphabet of PROLOG consists of:

- **C** — a set of constant symbols (or constants, for short),
- **V** — a set of variable symbols (or variables, for short),
- **F** — a set of function (term) symbols,
- **P** — a set of relation (predicate) symbols.

Meaning and Notation of Symbols

- **Constants** denote specific objects, items, elements, values, phenomena, etc. Constant names start with lower-case letters. Integers, rational numbers and strings are allowed (e.g. ’A small cat’).
- **Variables** are used to denote the same elements in case the precise name of an element is currently not known, unimportant, or a class of elements is to be represented. Variable names start with an upper-case letter.
- **Functional symbols** serve as complex object constructors. Such objects have a root symbol (an element of **F**) and a number of arguments. They follow the tree-like structure.
- **Predicate symbols** are used to define facts (relations). A fact can be *true* or *false*.
Prolog: Specific Role and Treatment of Variables

The Principal Roles of Variables

- **unknown objects** — ones to be found,
- **place-holders**, assure consistency with the arity of a functional or predicate symbol,
- **coreference constraints** — and data carriers.

- **Variables** may be used to denote *unknown but specific objects*; some variable $X \in V$ may denote an object the properties of which are specified without specifying the object itself; a class of objects can be defined in an implicit way.

- Functional and predicate symbol have assigned a **constant number of arguments**; this is called the *arity* of a symbol, to be denoted as:

  $$f/n,$$

  where $n$ is the arity of $f$ — the constant number of arguments of $f$. The number of arguments cannot change — no argument can be missing.

- **Variables** acts as *coreference constraints* and data carriers. Two or more occurrences of the same variable in an expression denote the same object; if any replacement of an occurrence of some variable takes place, all the occurrences of this variable must be replaced with the same symbol or value.
Motto: Do not kill Variables!!!

- In PROLOG variables can be substituted with certain values. This means that a variable can be assigned some value or be bound to it.
- The assignment can be annulled as a result of backtracking and then a new value can be assigned to the variable.
- Once a value is assigned it cannot be overwritten!!! The variable must be free first.

Example: WRONG!!!

```prolog
?- X=2, X=X+1, write(X).
false.
```

Example: O.K.

```prolog
?- X=2, Y is X+1, write(Y).
3
X = 2.
Y = 3.
```
Variable Assignment

1. The symbol = is for unification; in practice
   
   \[ X = a \]

   means \( X \) is bound to \( a \), while

   \[ X = Y \]

   means \( X \) and \( Y \) are bound with each other.

2. Is denotes assignment in the classic sense; the LHS value is calculated and assigned to the RHS variable, e.g.

   \[ Y \text{ is } 2 + 1. \]

   The RHS must be completely instantiated!

Singular Variable Occurrences

- **Warning:** singular variable occurrences are in fact nonsense! **PROLOG** produces warnings.

- **Anonymous variable** is denoted with _.

- All singular variable occurrences should be replaced with anonymous variable.
Terms

The set of terms $TER$ is one satisfying the following conditions:

- if $c$ is a constant, $c \in C$, then $c \in TER$;
- if $X$ is a variable, $X \in V$, then $X \in TER$;
- if $f$ is an $n$-ary function symbol ($f/n$), $f \in F$, and $t_1, t_2, \ldots, t_n$ are terms, then

$$f(t_1, t_2, \ldots, t_n) \in TER$$

- all the elements of $TER$ are generated only by applying the above rules.

Examples of terms

Assume $a, b, c \in C$, $X, Y, Z \in V$, $f, g \in F$, and arity of $f$ and $g$ is 1 and 2, respectively. The following are examples of terms:

- $a, b, c$;
- $X, Y, Z$;
- $f(a), f(b), f(c), f(X), f(Y), f(Z)$;
- $g(a, b), g(a, X), g(X, a), g(X, Y)$;
- $f(g(a, b)), g(X, f(X)), g(f(a), g(X, f(Z)))$. 
Terms in Prolog: some observations

Properties of terms

- **Warning:** Terms are not functions (nothing is calculated)!
- Terms are used to denote arbitrarily complex structures.
- The definition of terms is recursive (inductive).
- Having one functional symbol (of arity 1) and one constant symbol, an infinite number of terms can be defined.
- Terms and Atomic Formulae (facts) are syntactically identical.
- Terms are closed to records.

Examples of terms in Prolog

```prolog
1 man(socrates)
2 connected(a,b)
3 structure(a,X,f(a,b))
4 book(author(john,doe),title(abd_of_prolog))
5 tree(node(N),left(X),right(Y))
6 list(a,list(b,list(c,nil)))
7 f(f(f(f(f(a)))))
```
Structural object

```
book (book_title,
    author(first_name, last_name),
    publisher_name,
    year_of_publication
)
```

Structural object: XML

```
<book>
    <author>
        <first_name> Erik </first_name>
        <last_name> Ray </last_name>
    </author>
    <publisher_name>
        O Reilly and Associates, Inc.
    </publisher_name>
    <year_of_publication> 2003 </year_of_publication>
</book>
```
Structural object

```prolog
book (  
    title(book_title),
    author(author_name),
    publisher(publisher_name),
    year(year_of_publication)
)
```

Structural object: YAML

```yaml
book:
    title: book_title
    author: author_name
    publisher: publisher_name
    year: year_of_publication
```
Terms: examples

A LaTeX structure

\[ \frac{x}{y}, \sqrt{1 + \frac{x}{y}} \]

A LaTeX structure: Prolog view

\begin{verbatim}
frac(frac(x, y), sqrt(plus(1, frac(x, y))))
\end{verbatim}

A LaTeX structure — as term

\begin{verbatim}
\frac{
  \frac{x}{y}
}{
  \sqrt{1 + \frac{x}{y}}
}
\end{verbatim}
List construction as a term

1 list(red,list(green,list(blue,list(yellow,nil))))

Tree as a term

1 tree ( 
2 node (name, value), 
3 tree (node_left, left_left, left_right), 
4 tree (node_right, right_left, right_right) 
5 )

example

1 tree (root,list_of_subtrees)
Logical connectives

- `:-` is equivalent of implication (if),
- `,` is equivalent of conjunction (and),
- `;` is equivalent of disjunction (or).

Facts

1. `pred(arg1, arg2, ... argN).`

Clauses

1. `h :- p1, p2, ..., pk.`
2. `h :- q1, q2, ..., qm.`

Clauses — disjunction

1. `h :- p1, p2, ..., pk; q1, q2, ..., qm.`
Example Prolog Predicates

1. **var(+Term)** (nonvar(+Term))
   Succeeds if Term currently is (is not) a free variable.

2. **number(+Term)**
   Succeeds if Term is bound to an integer or floating point number.

3. **integer(+Term)**
   Succeeds if Term is bound to an integer.

4. **float(+Term)**
   Succeeds if Term is bound to a floating point number.

5. **rational(+Term)**
   Succeeds if Term is bound to a rational number.

6. **atom(+Term)**
   Succeeds if Term is bound to an atom.

7. **atomic(+Term)**
   Succeeds if Term is bound to an atom, string, integer or float.

8. **compound(+Term)**
   Succeeds if Term is bound to a compound term.

9. **ground(+Term)**
   Succeeds if Term holds no free variables.
Example Prolog Predicates

functor(?Term, ?Functor, ?Arity)
  Succeeds if Term is a term with functor Functor and arity Arity. If Term is a variable it is unified with a new term holding only variables.

arg(?Arg, +Term, ?Value)
  Term should be instantiated to a term, Arg to an integer between 1 and the arity of Term. Value is unified with the Arg-th argument of Term.

?Term =.. ?List
  List is a list which head is the functor of Term and the remaining arguments are the arguments of the term. Each of the arguments may be a variable, but not both. This predicate is called ‘Univ’.

Examples:

?- foo(hello, X) =.. List.
List = [foo, hello, X]

?- Term =.. [baz, foo(1)]
Term = baz(foo(1))