

---

# Prolog

## A Glossary

---

[www.opal-project.de/sae\\_project](http://www.opal-project.de/sae_project)

Michael Eichberg (mail@michael-eichberg.de)

29. Dezember 2010

---

## Introduction

This is a small Prolog glossary.

In general, we use the terminology as defined in Learn Prolog Now! and in The Art of Prolog.<sup>1</sup>

The intention of this document is just to make sure that the terminology is used consistently and unambiguously. This document is not an introductory text on Prolog programming.

## Reference Example

The reference example code that we use as the foundation for explaining various Prolog terms.

Line Number	Prolog Code
1	<code>:- module('Utils',[lookup/3,membercheck_d1/2]).</code>
2.1	<code>lookup(Key,[(Key,Value) _Dict],Value).</code>
2.2.1	<code>lookup(Key,[(Key1,_) Dict],Value) :-</code>
2.2.2	<code>Key \= Key1,lookup(Key,Dict,Value).</code>
3.1	<code>membercheck_d1(E,[E _]-_) :- !. % green cut</code>
3.2	<code>membercheck_d1(OtherE,[E Rest]-Last) :-</code>
3.2.1	<code>E \= OtherE,</code>
3.2.2	<code>membercheck_d1(OtherE,Rest-Last).</code>

This code snippet defines two predicates (lookup/3 and membercheck\_d1/2) that belong to the module Utils and which are exported by the latter. Both predicates are each defined by two rules (2.1, 2.2 and 3.1, 3.2).

---

<sup>1</sup> The book “Learn Prolog Now” is freely available ([www.learnprolognow.org](http://www.learnprolognow.org)) and well suited as an introductory course on Prolog.

## Terminology

### Term

A term is either a string (atom), an integer value/atom, a floating-point value/atom, a variable, an anonymous variable or a complex term.

<code>'Utils'</code>	Utils is a string atom (Line 1); the “'” are used to quote the string atom
<code>3</code>	an integer atom (Line 1)
<code>Key</code>	a variable (Line 2.1)
<code>_Dict</code>	an anonymous variable (Line 2.1)
<code>Rest-Last</code>	a complex term, where the functor is “-” and where the complex term’s arguments are Rest and Last (Line 3.2.2)
<code>membercheck_d1(     OtherE, Rest-Last )</code>	a complex term, where membercheck_d1 is the functor and (OtherE, Rest-Last) are the complex term’s arguments (Line 3.2.2)

### Complex Term

A complex term<sup>2</sup> has a functor, which is a string atom and a non-empty sequence of arguments; i.e, a complex term always has at least one argument.

### Functor

The name of a complex term.

<code>Rest-Last</code>	“-” is the functor (Line 3.2.2)
<code>membercheck_d1/2</code>	“/” is the functor of this complex term (“/” is an infix operator); membercheck_d1 and 2 are the arguments of this term. (Line 3.1)
<code>membercheck_d1(     OtherE, Rest-Last )</code>	membercheck_d1 is the functor (Line 3.2.2)

### Arguments

The arguments of a complex term.

<code>Rest-Last</code>	Rest and Last are the arguments of the complex term “-” (Line 3.2.2)
<code>membercheck_d1(     OtherE, Rest-Last )</code>	OtherE and Rest-Last are the two arguments of the complex term membercheck_d1 (Line 3.2.2)

### Arity

<sup>2</sup> Complex terms are also called *compound terms* or *structures*.

The number of arguments of a term.

!	“!” (the cut) is an atom and, hence, the arity is 0
3	the arity of integer atoms is 0
lookup/3	the arity of the given complex term (“/”) is two; “lookup” and 3 are the arguments of the term
lookup( Key, [(Key,Value) _Dict], Value )	the arity of lookup is 3. (line 3.2.2)

## Clause

A clause is either (a fact, ) a rule or a directive.

In many Prolog text books directives are not explicitly mentioned/are ignored when the syntax is discussed. However, due to their relevance when compiling/analyzing Prolog programs, we explicitly distinguish between rules and directives at the top-level.<sup>3</sup>

Technically, a clause is a term that is terminated by “.” and which is either a complex term or a string atom. A clause is never (just) a variable or a numerical value. For example, “do\_it.” is a valid clause and “X <=> Y.” is also a valid clause provided that “<=>” is an infix operator. However, “X.” or “1.” are not valid clauses.

## Rule

In general, a rule is a clause where the functor is “:-” and the arity is 2. The first argument of a rule is called the head of the rule and the second argument is called the body of the rule.

lookup( Key,[(Key,Value)  _Dict],Value ).	a rule where the head is the given complex term (lookup(...)), the body is (implicitly) the goal “true”.
membercheck_d1(E,[E _]-_) :- !.	a rule where the head is the complex term membercheck_d1 and where the body just consists of the cut. (The goal is the cut.)

## Head

The first argument of a rule. The head is either a complex term or a string atom.

## Body

The second argument of a rule. The body of a rule is always *exactly one* term; often it is a complex term with the functor ‘,’ and arity 2; i.e., the logical and/the conjunction of two (sub-)terms.

<sup>3</sup> Directives are (conceptually) executed by the Prolog compiler/interpreter when they are encountered.

## Fact

Conceptually, a fact is a clause that states something that is universally true; all variables are (implicitly) universally quantified. Hence, a fact can contain variables, as long as no variable is used more than once and no two variables are put into a specific relation/are compared to each other. For example, the clause “loves(X, good\_weather).” is a fact that states that everyone loves good weather. However, the clause “loves(X, X).” is considered a rule that states that everyone loves oneself - the first and the second argument are unified and hence, put into relation.

A fact can always be considered as a rule where the body is just the goal “true”.

## Predicate

A predicate is defined by the set of all rules where the head term has the same functor and arity. A predicate is identified by the term “<functor>/<arity>”.

Line 2.1 and 2.2 define the predicate lookup/3, Line 3.1 and Line 3.2 define the predicate membercheck\_d1/2.

## Directive

A clause with the functor “:-” and arity 1. A directive is always directly evaluated by the compiler/the interpreter and is not executed at runtime.

## Operator

A string atom that is in the (current) operator table. An operator is always also the functor of the corresponding complex term.

Key \= Key1	“\=” is an infix operator and the functor of the (complex) term
:- module('Utils', [...]).	“:-” is a prefix operator
Rest-Last	here, “-” is the infix operator and the functor of the term

An operator either consists of special characters only (“=”, “>”, “:”, ...) or is a string atom.

## Goal

A goal is a term that Prolog may try to call (to prove) when the body of a clause is evaluated. In general, all terms that Prolog calls when it tries to prove a clause are called goals.

However, strictly speaking, the body of a clause is always exactly one term (one goal) and in many cases the top-level goal is the “,” (*logical and*). In this case, Prolog will try to prove the two sub-goals of the and-goal and – if both succeed – the and-goal as a whole succeeds.

The control-flow related built-in goals are: “,” (*logical and*), “;” (*logical or*), “!” (*cut*), “C -> T ; E” (*if-then-else*), “C -> B” (*if-then-fail*), “C \*-> B” (*soft-cut*), true and fail.

A goal sequence are those goals that are connected using the logical and (“,”).

<pre>... :-     Key \= Key1,     lookup(Key,Dict,Value).</pre>	Prolog will try to prove three goals, the top-level goal “,”(logical and) and the two subgoals “\=”(does-not-unify) and lookup/3.
<pre>lookup(     Key,     [(Key,Value) _Dict],     Value ).</pre>	When lookup/3 is called, Prolog will try to unify the first argument with the “free” variable Key and will then try to unify the second argument with the specified term and then will try to unify the third argument with the value (if any) bound to the variable Value.

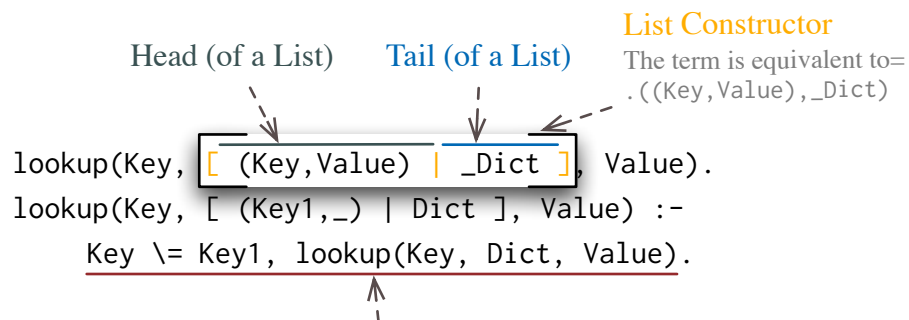
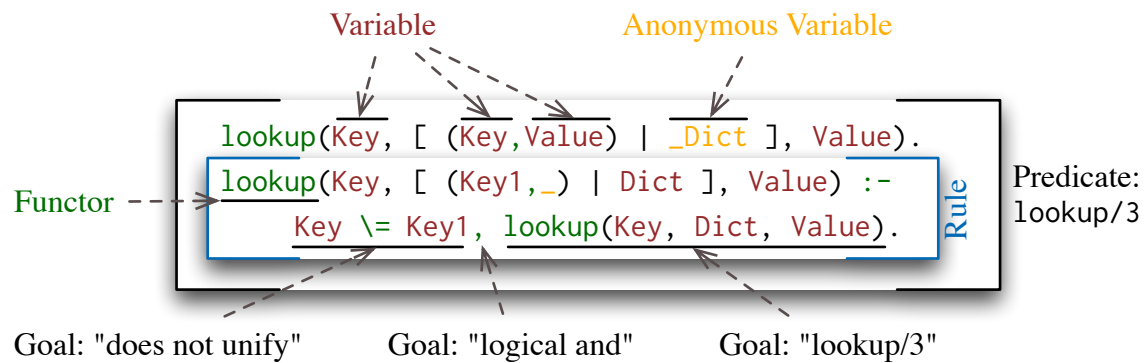
## Variable

A character sequence that starts with a capital letter. A variable is either *free* or *instantiated*. Two variables that are unified are also said to *share*.

## Anonymous Variable

A character sequence that starts with an underscore. Anonymous variables that have the same name and which are used in the same term, do not share; i.e., they are effectively different variables.

## Cheat Sheet



Equivalent to: `' , ' ( ' \= ' (Key,Key1), lookup(Key, Dict, Value) )`

