Elements of Prolog Programs

A Glossary

www.opal-project.de/sae_project
Michael Eichberg (mail@michael-eichberg.de)
13. November 2010

Introduction

This document briefly explains the terminology used to refer to the core elements of Prolog programs; i.e., it is basically a glossary w.r.t. the structure of Prolog programs. In general, we use the terminology as defined in Lear Prolog Now! and in The Art of Prolog.[1]

The intention of this document is just to make sure that the terminology is used consistently and unambiguously. This document does not explain the syntax or grammar rules of Prolog programs.

Reference Example

The reference example code is:

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

This code snippet defines two predicates (`lookup/3` and `membercheck_dl/2`) that belong to the module `Utils` and which are exported by the latter. Both predicates are each defined by two rules.

[1] The book “Learn Prolog Now” is freely available (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.

`'Utils'`	`Utils` is a string atom (Line 1); the "'" are used to quote the string atom
`3`	an integer atom (Line 1)
`Key`	a variable (Line 2.1)
`_Dict`	an anonymous variable (Line 2.1)
`Rest-Last`	a complex term, where the functor is "`-`" and where the complex term's arguments are `Rest` and `Last` (Line 3.2.2)

`membercheck_dl(`
`  OtherE,Rest-Last`
`)`
	a complex term, where `membercheck_dl` is the functor and `(OtherE,Rest-Last)` are the complex term's arguments (Line 3.2.2)

Complex Term

A complex term[2] 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.

`Rest-Last`	"`-`" is the functor (Line 3.2.2)
`membercheck_dl/2`	"`/`" is the functor of this complex term ("`/`" is an infix operator); `membercheck_dl` and `3` are the arguments of this term. (Line 3.1)
`membercheck_dl(OtherE,Rest-Last)`	`membercheck_dl` is the functor (Line 3.2.2)

Arguments

The arguments of a complex term.

`Rest-Last`	`Rest` and `Last` are the arguments of the complex term "`-`" (Line 3.2.2)

`membercheck_dl(`
`  OtherE,Rest-Last`
`)`
	`OtherE` and `Rest-Last` are the two arguments of the complex term `membercheck_dl` (Line 3.2.2)

Arity

[2] 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.[3]

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.

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_dl(E,[E|_]-_) :-` `  !.`	a rule where the head is the complex term `membercheck_dl` 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.

[3] 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,snow).`" is a fact that states that everyone loves snow. 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_dl/2`.

Directive

A clause with the functor "`:-`" and arity `1`.

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 term
`:- module('Utils',[...]).`	"`:-`" is a prefix operator
`Rest-Last`	here, "`-`" is the infix operator and the functor of the term

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.

Built-in goals are in particular the "`,`"(*logical and*), "`;`"(*logical or*), "`!`"(*cut*), "`->`"(*if-then-else*), `true` and `fail`.

A goal sequence is a list of goals that are connected using the logical and("`,`").

`... :-`
`  Key \= Key1,`
`  lookup(Key,Dict,Value).`
	Prolog will try to prove three goals, the top-level goal "`,`"(logical and) and the two subgoals "`\=`"(does-not-unify) and `lookup/3`.

`lookup(`
`  Key,`
`  [(Key,Value)|_Dict],`
`  Value`
`).`
	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.

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

Elements of Prolog Programs