The {log} Programming Language

Gianfranco Rossi

December 15, 1997

Contents

1 Syntax

$\{\mathit{log}\}$ (read 'set-log') uses most of the syntactic conventions and notations of standard PROLOG (e.g., C-Prolog, SICStus Prolog). In this section we briefly describe the syntactic differences between $\{\mathit{log}\}$ and standard Prolog, and provide some hints on their (informal) semantics.

1.1 Terms

Terms in $\{\mathit{log}\}$ have the usual form as in standard Prolog. The only difference is the presence of a few interpreted function symbols used to build extensional and intensional sets.

1.1.1 Set terms

The following two function symbols are used to build extensional sets, that is sets defined by explicitly enumerating all their elements:

- $'\{\}'$, of arity 0, interpreted as the empty set;
- $\{\cdot/\cdot\}$, of arity 2, interpreted as the set constructor: $\{t\,/\,s\}$ is the set obtained by adding the element t to the set s, that is $\{t\} \cup s$.

Terms whose main functor is either $'\{\}'$ or $\{\cdot/\cdot\}$ are called *set terms*. Precisely, set terms are recursively defined as follows:

> Variables and $'\{\}'$ are <u>set terms</u>. If s is a set term and t is any term then $\{t\,/\,s\}$ is a <u>set term</u> (s is said the *tail* part of the set term).

As a syntactic convenience, the set term

$$\{t_1\,/\,\{\,t_2\,/\ldots\{t_n\,/\,s\}\ldots\}\}$$

where $t_1,\ldots,t_n$ are terms, and s is a set term, can be denoted simply as

$$\{t_1,t_2,\ldots,t_n\,/\,s\}$$

and, if s is the empty set $'\{\}'$, as

$$\{t_1,t_2,\ldots,t_n\}.$$

Both ground and *non-ground* set terms are allowed. Moreover set terms can be *nested* at any depth. (The kind of sets which can be represented in this way are the so called *hereditarily finite sets*).

Examples.

- $\mathsf{f(a,\{b\})}$ is a term (containing a set term), but not a set term;
- $\mathsf{\{2,g(3),a\}}$ is a ground set term;
- $\mathsf{\{1,X,Y,2\}}$ and $\mathsf{\{1,1/X\}}$ are non-ground set terms;
- $\mathsf{\{1,\{1,\{1\}\}\}}$ is a set term containing nested set terms.

The only non-variable terms allowed in the tail part of $\{\cdot/\cdot\}$ are set terms.[1] Any other term built using $\{\cdot/\cdot\}$ and not satisfying this condition (e.g., $\{\mathsf{a}\,/\,\mathsf{b}\}$) is considered illegal (a 'wrong set term' error message may be produced).[2]

Set terms are stored internally using the `with` function symbol, written infix, letf-associative. The set term

$$\{t_1, t_2, \ldots, t_n \,/\, s\}$$

is represented in the `with`-notation as

$$s \,\texttt{with}\, t_n \,\texttt{with} \ldots \texttt{with}\, t_2 \,\texttt{with}\, t_1.$$

For the sake of flexibility, the `with`-notation is allowed to be used also at the top-level, and can be freely mixed with the representation of sets using curly brackets. For example, the terms f('{}' `with` b `with` a, {1,2}) and {a / '{}' `with` b `with` {c}} are accepted as syntactically correct by the { *log* } interpreter.

A difference between using the `with`-notation and the curly brackets notation is that with the latter the { *log* } interpreter automatically removes ground duplicate terms possibly encountered in set terms during their input/output, whereas duplicates are always maintaned with the `with`-notation.

1.1.2 Intensional set terms

Intensional set terms are built using the function symbol $\{\cdot\}$, of arity 1, applied to a term whose main functor is the symbol :, of arity 2, used infix, in the following way:

$$\{X \;:\; G\}$$

[1] An approach in which, on the contrary, such terms (called *colored* sets) are allowed has been developed in [?].

[2] *** N.B. *** No check on the tail part of a set is performed during execution (hence it can happen that a variable set tail gets bound to a non-set term); syntax checks on set tails are performed during input of programs and goals and output of results.

where X is a variable, and G an arbitrary $\{\mathit{log}\}$ goal containing X.

The meaning of such a term is: "the set of all X satisfying the formula G".

Intensional set terms can occur everywhere ordinary terms can occur. In particular, the head of a clause can contain an intensional set term, or a RUQ atom (cf. next section) can contain an intensional set term.

Examples.

- {X : likes(X,wine)} ;
- union(S1,S2,{X : X in S1 or X in S2}): an atom containing an intensional set term;
- {X : X in {Y : between(Y,1,N)} & is_prime(X)} : nested intensional set terms;

1.2 Atoms and clauses

1.2.1 Predefined predicates

The following predefined predicates (all of arity 2, written infix) are provided for set manipulation:

- t in s (**Membership**): true if there is a substitution σ s.t. t^σ belongs to s^σ, i.e., $t^\sigma \in s^\sigma$;[3]
- t nin s (**Non-membership**); true if there is no substitution σ s.t. t^σ belongs to s^σ, i.e., $\nexists\sigma(t^\sigma \in s^\sigma) \equiv \forall\sigma(t^\sigma \notin s^\sigma)$;
- $t = s$ (**Equality**); true if there is a substitution σ s.t. t^σ is equal to s^σ, i.e. $t^\sigma = s^\sigma$;
- t neq s (**Inequality**); true if there is no substitution σ s.t. t^σ is equal to s^σ, i.e., $\nexists\sigma(t^\sigma = s^\sigma) \equiv \forall\sigma(t^\sigma \neq s^\sigma)$;

where t and s are terms.[4] Set manipulation atoms can occur in clause bodies and goals everywhere ordinary atoms (based on both built-in and user defined predicates) can occur.

[3] More precisely, one should say, ' if there is a substitution σ s.t. the object associated by the selected interpretation to t^σ belongs to the object associated by this interpretation to s^σ.

[4] s can be any term, hence, also a non-set term. If this is the case, the interpretation of s is such that no element belongs to s (like for the empty set).

1.2.2 Restricted Universal Quantifiers

A {*log*} atom can be a Restricted Universal Quantifiers (RUQs), that is a formula of the form:

$$\mathsf{forall}(X \ \mathsf{in} \ s\ ,\ G)$$

where X is a variable, s a term, and G an arbitrary goal containing X.

This form stands for the quantified implication

$$\forall X((X \in s) \rightarrow G).$$

RUQs atoms can occur in clause bodies and goals everywhere ordinary atoms can occur.

The goal G in a RUQ can be any {*log*} goal, in particular, one containing a built-in {*log*} predicate, or another RUQ (in fact, RUQs can be nested at any depth), or an arbitrary conjunction of {*log*} atoms.[5]

Examples.

- forall(X in {1,2,4}, p(X, Y)).
- forall(X in S1,X in S2).
- (nested RUQ's) forall(X in {1,2}, forall(Y in S, p(X, Y))).
- forall(X in S1, X in S2 & p(X,Y) & write(Y) & nl).[6]

RUQs can be freely combined with intensional set formers, that is, the goal part of a set former can contain a RUQ, as well as the set in a RUQ can be defined intensionally rather than extensionally. For example:

- forall(X in {Y : between(Y,1,M)}, p(X)). [7]

1.2.3 {*log*} clauses

Syntactic differences w.r.t. standard Prolog clauses:

- '&' is used in place of ',' to represent goal conjunction;

[5] G must contain at least one occurrence of the control variable X; otherwise an error is signalled at compile time.

[6] See Sect. 1.2.3 for the syntactic form of {*log*} clauses.

[7] *** WARNING *** Some problems when using an intensional set term within a RUQ can still occur in the current implementation. So, please, avoid such kind of nesting by using an extra variable and equality, e.g. S = {Y : between(Y,1,M)} & forall(X in S, p(X)).

- ‘non’ is used in place of ‘not’ to represent negation;
- ‘or’ is used in place of ‘;’ to represent goal disjunction.

Examples.

- q(X) :- X in {a,b,c,d} , p(X).
- singleton({_}).
- in_difference(X,Set1,Set2) :- X in Set1, X nin Set2.
- subset(S1,S2) :-
  forall(X in S1,X in S2).

2 Set unification

Set unification extends standard unification by incorporating the axioms of a simple theory of sets, so as to enforce the permutativity and absorption properties of the set constructor.

*** TO BE COMPLETED ***

Examples (without tail variables).

Using permutativity:

- <- {john,mary} = {mary,john}

  yes
- <- {X,Y} = {mary,john}

  X = mary, Y = john
  X = john, Y = mary

Using absorption:

- <- {mary} = {mary,mary}

  yes
- <- {X} = {mary,mary}

  X = mary

- <- {X} = {mary,mary,john}

  no

Using permutativity and absorption

- <- {X,Y,Z} = {mary,mary,john}

  X = mary, Y = john, Z = mary
  X = john, Y = mary, Z = john
  X = mary, Y = mary, Z = john
  X = john, Y = john, Z = mary
  X = john, Y = mary, Z = mary
  X = mary, Y = john, Z = john

- (with variables in both set terms)
  <- {X,Y} = {mary,Z}

  X = mary, Y = mary, Z = mary
  X = mary, Y = _1, Z = _1
  X = _1, Y = mary, Z = _1

Examples (with tail variables).

- <- {john/R} = {mary/S}

  R = {mary/_1}, S = {john/_1}

- <- {john/R} = {john/S}

  R = _1, S = _1 (i.e., R = S)
  R = {john/_1}, S = _1
  R = _1, S = {john/_1}

- (special cases (i))
  <- S = {john/S}

  S = {john/_1}

- (special cases (ii))
  <- {john/S} = {mary/S}

  S = {john,mary/_1}.

3 Other predifined predicates

{*log*} is an instance of the general CLP scheme. Basic set operations (namely, in and nin), along with disequality (neq) are dealt with as *constraints*. Satisfiability of the current set of active constraints is checked at each step during the computation. Irreducible constraints (i.e., those which are not enough instantiated) are passed to the next computation step, waiting for a sufficint instantiations of variables occurring in them. Constraints which at the end of the computation are still irreducible are returned as part of the computed answer.

Irreducible constraints

- $X = t$, X variable, t any term other than $\{t_1, \ldots, t_n/X\}$;
- t nin X
- X neq t.

Examples.

- <- X in S

  X = _1, S = {_1/_2}

- <- X in {john,mary/Others}

  X = john, Others = _1
  X = mary, Others = _1
  X = _1, Others = {_1/_2}

- <- X in S & X nin S

  no

- <- X in {a} & X neq a

  no

- <- f(X,b) neq f(a,Y)

  X = _1, Y = _2, _1 neq a X = _1, Y = _2, _2 neq b

- <- a nin {X,Y/S}

  X neq a, Y neq a, a nin S

- <– a nin {X,Y/S} & Y in {a,b}

  Y = b, X neq a, a nin S

- <– {a} = {X/S} & X nin S

  X = a, S = '{}'
  (without the nin-constraint, also X = a, S = {a} is a correct solution)

- <– X neq a & {X,Y} = {a,b}

  X = b, Y = a
  (without the neq-constraint, also X = a, Y = b is a correct solution; note that the order of atomic constraints in the goal does not matter).

4 Restricted Universal Quantifiers

*** TO BE COMPLETED ***

5 Intensional set terms

*** TO BE COMPLETED ***

6 Basic set operations (i.e., the { *log* } library)

It is shown that the usual set-theoretic operations, such as union, intersection, etc., can be effectively programmed in { *log* } .

*** TO BE COMPLETED ***

7 Programming examples

7.1 Simple set theoretic operations

Example

Program:

```
in_difference(X,Set1,Set2) :-
        X in Set1,X nin Set2.
```

Goal:

```
<- in_difference(X,{1,2},{1,3}),
```

Goal:

```
<- in_difference(X,Set1,{1,3}),
```

Example

Tests whether S1 and S2 are disjoint sets.

Program:

```
disj(S1,S2) :-
        forall(X in S1, forall(Y in S2, X neq Y)).
```

*** TO BE COMPLETED ***

7.2 Combinatorial problems

*** TO BE COMPLETED ***

8 Extra-logical built-in predicates

*** TO BE COMPLETED ***

9 The { *log* } interpreter

*** TO BE COMPLETED ***