Common Syntax of the DFG-Schwerpunktprogramm “Deduktion”

Reiner Hähnle
Fakultät für Informatik
Universität Karlsruhe
D-76128 Karlsruhe
reiner@ira.uka.de

Manfred Kerber
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, England
M.Kerber@cs.bham.ac.uk

Christoph Weidenbach
Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
weidenb@mpi-sb.mpg.de

Abstract
A common exchange format for logic problems to be used by members of the DFG-Schwerpunktprogramm “Deduktion” is introduced. It is thought to be an internal format that can easily be parsed such that it forms a compromise between the needs of the different groups. It is not intended to be a high-level general logic language that is easy to read or to write. The language is more general than other popular exchange formats such as Otter or TPTP in allowing non-clausal and sorted formulas as well as user-defined operators and quantifiers. The latter feature makes it also useful for non-classical logics.

1 Introduction

The language proposed in the following is intended to be a common exchange format for logic problem settings. It is thought to be an internal format that can easily be parsed such that it forms a compromise between the needs of the different groups. Therefore, it is kept as simple as possible. This language is not intended to be a high-level general logical language that is easy to read or to write.

In any case it will be necessary to provide tools that transform files from the present syntax into other standard formats (e.g., Otter [6] or TPTP [9]) and vice versa.

Terminal symbols are underlined or one of the following symbols ‘,’ ‘(’, ‘)’, ‘ ‘, or ‘-‘.

2 Problems

What we exchange are problems. Problems are stand alone files, with respect to possible inclusions (see Section 6), that describe one specific first-order logic problem.

\[
\text{problem ::= begin\_problem(label). \hspace{1cm} description} \\
\hspace{1cm} \hspace{1cm} \text{logical\_part} \\
\hspace{1cm} \hspace{1cm} \{settings\} \hspace{1cm} \text{end\_problem.}
\]
The description provides at least information for the identification of the problem like problem’s name, author’s name, version, date, and informal description (see Section 4). If the logical part contains non-standard operators or quantifiers they have to be explained here. The explanation should contain, if possible, translation rules to standard first-order logic. Settings contain problem specific setting information as well as prov dependent strategic directives like restrictions on the term depth, directives for the compiler, set-of-support etc. The logical part contains the pure problem. The problem should be uniquely determined by this part alone.

3 The Logical Parts

Identifiers for function, predicate and sort symbols, constants are subject to the unique name assumption and mutually exclusive with the set of terminal symbols. The same holds for variables within the same scope of quantifiers.

\[
\text{logical part} :: \{\text{symbol list}\} \{\text{declaration list}\} \{\text{formula list}\}^* \\
\]

All signature symbols as well as additional operators and quantifiers have to be declared in advance in the symbol list of a problem. An arity may be given in the declaration. The arity may either be “-1”, meaning arbitrary arity, or a non-negative number for some fixed arity. If no arity is given, the symbol is declared with fixed arity according to the formula part. Therefore, symbols with varying arity have to be declared explicitly.

\[
\text{symbol list} :: \text{list of symbols.} \\
\{\text{functions}\{\text{func sym arity}\} \{\text{func sym arity}\}^*\}. \{\text{predicates}\{\text{pred sym arity}\} \{\text{pred sym arity}\}^*\}. \{\text{sorts}\{\text{sort sym sort sym}\} \{\text{sort sym sort sym}\}^*\}. \{\text{operators}\{\text{op sym arity}\} \{\text{op sym arity}\}^*\}. \{\text{quantifiers}\{\text{quant sym arity}\} \{\text{quant sym arity}\}^*\}. \text{end of list.} \\
\text{arity} :: \{-1 \mid \text{number}\} \\
\]

The sort declarations are optional. They can be mapped to standard first-order logic by the usual relativization rules. Sorts may be used as unary predicate symbols in the logical part.

\[
\text{declaration list} :: \text{list of declarations.} \\
\{\text{declaration}\}. \text{end of list.} \\
\text{declaration} :: \text{subsort decl} \mid \text{term decl} \mid \text{pred decl} \\
\text{subsort decl} :: \text{subsort}\{\text{sort sym sort sym}\}. \\
\text{sort sym} :: \text{identifier} \\
\text{term decl} :: \text{forall}\{\text{var list, term expr}\} \mid \text{term expr.} \\
\text{term expr} :: \text{sort sym}(\text{term}) \\
\text{pred decl} :: \text{predicate}\{\text{pred sym, sort sym}\}. \\
\]

We treat axioms and conjectures differently. Strategies like goal-oriented reasoning are specified in the settings part. Optionally each formula can be given a label. The list of formulas is logically interpreted to the conjunction of its parts. Logically, the axioms list and the conjectures list form an implication: \(\text{axioms} \supset \text{conjectures}\). Therefore, the formulas to be proved should not be put in negated form in the conjecture list.
Formulas are either quantified expressions or ground expressions.

\[ \text{formula} ::= \text{quantifier}(\text{term_list}, \text{formula}) | \text{quantifier}(\text{var_list}, \text{formula}) \]

Clauses are a subset of the set of formulas. We do not allow implicit quantification, that is, formulas and clauses must not contain free variables. All unbound simple terms are interpreted as constants. The rationale behind this is that the interpretation of a problem should be determined by the problem format, not by the prover. In Otter, e.g., free variables in clauses are implicitly universally quantified, while variables in formulas are implicitly existentially quantified. Moreover, variables must begin with \( u, v, w, \ldots \), constants with \( a, b, c, \ldots \). We find this confusing.

Note that unit clauses are of the form \( \text{or}([\text{literal}]) \). The empty clause may be specified as \( \text{or}(\text{false}) \).

Formulas that do not start with a quantifier are built in the usual way.

\[ \text{atom} ::= \text{zero_op} | \text{pred_expr} | \text{equation} \]

The operators \( \text{or} \) and \( \text{and} \) are available for all arities greater than 0.
We think of 0-ary connectives (truth values) as atoms.

\[
\text{pred_expr} ::= \text{pred_sym}(\text{term},\text{term})) \mid \text{sort_sym}(\text{term})
\]
\[
\text{equation} ::= \text{equal}(\text{term},\text{term})
\]
\[
\text{pred_sym} ::= \text{identifier}
\]

0-ary predicates are allowed.

\[
\text{term} ::= \text{constant} \mid \text{variable} \mid \text{fun_sym}(\text{term},\text{term})
\]
\[
\text{constant} ::= \text{identifier}
\]
\[
\text{fun_sym} ::= \text{identifier}
\]
\[
\text{identifier} ::= \{\text{letter} \mid \text{digit} \mid \text{special_symbol}\}^+
\]
\[
\text{letter} ::= \text{a} \to \text{z} \mid \text{A} \to \text{Z}
\]
\[
\text{number} ::= \{\text{digit}\}^+
\]
\[
\text{digit} ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\]
\[
\text{special_symbol} ::= \#
\]

### 3.1 Example without Sorts

We start with a complete description of Pelletier’s [7] problem No. 57:

```prolog
begin_problem(Pelletier57).

list_of_descriptions.
name(Problem No. 57 from the Pelletier Collection).
status(unsatisfiable).
description(This is a simple problem.).
end_of_list.

list_of_symbols.
functions[f_2,a,b,c].
predicates[F].
end_of_list.

list_of_axioms.
formula(F(f(a,b),f(b,c)),ax1).
formula(F(f(b,c),f(a,c)),ax2).
formula(forall([x,y,z],impl(and(F(x,y),F(y,z)),F(x,z))),ax3).
end_of_list.

list_of_conjectures.
formula(F(f(a,b),f(a,c)),co1).
end_of_list.

end_problem.
```
3.2 Example with Sorts

We describe the logic part of an equality problem containing the natural numbers.

```
list_of_symbols.
functions(plus,s,zero).
sorts(even,nat).
end_of_list.

list_of_declarations.
subsort(even,nat).
even(zero).
forall([(nat(x)],nat(s(x)))).
forall([(nat(x),nat(y)],nat(plus(x,y)))).
forall([(even(x),even(y)],even(plus(x,y)))).
forall([(even(x)],even(s(s(x)))).
forall([(nat(y)],even(plus(y,y)))).
end_of_list.

list_of_axioms.
formula(forall([(nat(y)],equal(plus(y,zero),y)), ind_start)).
formula(forall([(nat(y),nat(z)],equal(plus(y,s(z)),s(plus(y,z))]), ind_step)).
end_of_list.
```

4 Descriptions

```
description ::= list_of_descriptions.
    name(text).
    author(text).
    {version(text).}
    {logic(text).}
    status(log_state).
    description(text).
    {date(text).}
end_of_list.

log_state ::= satisfiable | unsatisfiable | unknown
```

We allow arbitrary text that is compatible with the syntax. The logic part is mandatory, if the problem contains non-standard operators or quantifiers.

5 Settings

The settings contain problem and system specific information such as switches, lists of formulas that are treated specially, hints for the compiler etc. The settings consist of a general setting section and various system dependent sections. The content of the general setting section is currently restricted to an enumeration of “hypotheses”, that are formulas of the conjecture part which are compatible with the axioms. The axioms in conjunction with the hypothesis are satisfiable. The general section part is open to extensions if needed.

For the system dependent sections we require a unique label for each system. There are no restrictions for the content of these sections except the compatibility with the syntax.
settings ::= list_of_general_settings {setting_entry}+ end_of_list. | list_of_settings(setting_label). text end_of_list.
setting_entry ::= hypothesis(label {,label})*
setting_label ::= KIV | LEM | PROTEIN | SATURATE | 3TAP | SETHEO | SPASS

The labels name the following systems: KIV [8], LEM [4], PROTEIN [1], SATURATE [3], 3TAP [2], SETHEO [5], SPASS [10]. For example, to specify a set of support one may (i) give labels to the formulas/quotes that have support and (ii) list these labels in a list under some keyword set_of_support:

list_of_settings(SPASS).
set_of_support(ax1,ax2,ax3).
precedence(a,b,c,f,F).
end_of_list.

6 Miscellaneous

6.1 Comments
After the '%' symbol the rest of line is ignored. Comments stretching over several lines may be enclosed by a '/*', '···', '*/' pair.

6.2 Includes
At any part of a problem, another file might be included by include(filename)

Includes are a necessary means to structure problems. However, they should be used with some care because with "includes", problem files are no longer stand alone documents. In order to avoid the problem of different pathname conventions on different machines, only files being in the same directory can be included.

Acknowledgements
We would like to thank all members of the German “Schwerpunkt Deduktion” group who contributed to this paper. Special thanks to Enno Keen and Andreas Nonnengart who proof-read several preliminary versions of this paper.

References


6


[10] Christoph Weidenbach, Bernd Gaede, and Georg Rock. SPASS & FLOTTER, Version 0.42. Submitted, available via ftp from ftp.mpi-sb.mpg.de in the directory pub/SPASS named fass.dvi, see also the SPASS distribution in the very same directory spass.0.42.tgz, 1996.