The ProSpec and EQTrafo User's Manual

Peter Baumgartner and Bernd Thomas
Universität Koblenz-Landau
Rheinau 1
56075 Koblenz

{peter,bthomas}@informatik.uni-koblenz.de

Draft of July 26, 1997

1 Introduction

This document describes two tools developed at the AI Group at the University of Koblenz. Both are intended as frontends to be used in conjunction with classical, first order clause logic theorem provers (such as PROTEIN [Baumgartner and Furbach, 1994]).

ProSpec: ProSpec takes as input a file consisting of a *sort declaration* and some *well-sorted formulas* (not neccessarily clauses), possibly containing the equality predicate. The output is a set of unsorted clauses with equality. In brief, ProSpec transforms sorted logic with equality into unsorted logic with equality in a semantics preserving way.

EQTrafo: EQTrafo takes as input a file consisting of clauses, possibly containing the equality predicate. The output is a set of clauses not containing the equality predicate. In brief EQTrafo transforms clause logic with equality into clause logic without equality in a semantics preserving way.

Hence, typically a sorted specification with equality is proved using the command sequence ProSpec— EQTrafo— Protein.

2 ProSpec

We will first describe the semantics of the transformation; the subsequent ProSpec user's guide will describe the concrete syntax and how to run the program.

2.1 Syntax and Semantics of the Transformation

By a *sorted specification* we mean a pair consisting of a *sort declaration* and a set of *well-sorted sorted formulas*.

Following standard terminology, a *sort declaration* for a signature Σ consists of

- a set $\textsc{Sort}$ of symbols, called *sorts*,
- a mapping $s : \mathit{Var} \mapsto \textsc{Sort}$ which maps each variable of Σ to a sort,
- a set of *term declarations* of the form $t \in S$, where $S \in \textsc{Sort}$,
- a set of *predicate declarations* of the form $P : S_1 \times \cdots \times S_n$, where P is an n-ary predicate symbol and $S_i \in \textsc{Sort}$,
- and a set of *subsort declarations* of the form $S_1 \sqsubseteq S_2$ where $S_1, S_2 \in \textsc{Sort}$.

For instance, using the convention that a variable x with sort s (i.e. $s(x) = S$) is displayed as x_S, a valid term declaration is $\mathit{succ}(\mathit{succ}(x_{\mathit{Even}})) \in \mathit{Even}$ (provided $\mathit{Even} \in \textsc{Sort}$). An example for a subsort declaration would be $\mathit{Nat} \sqsubseteq \mathit{Integer}$.

Although these notions all are standard, we gave them here in order to formulate restrictions apply in our case:

- The sort hierarchie must be a tree: whenever $S \sqsubseteq S_1$ and $S \sqsubseteq S_2$ then $S_1 = S_2$.
- For every $n - \mathit{ary}$ predicate symbol P there is exactly one predicate declaration

  $$P : S_1 \times \cdots \times S_n \quad \text{where } S_i \in \textsc{Sort}, \text{ for } 1 \leq i \leq n.$$

- For every $n - \mathit{ary}$ function symbol f there is exactly one term declaration

  $$f(x^1_{S_1}, \ldots, x^n_{S_n}) \in S \quad \text{where } S,\ S_i \in \textsc{Sort},\ x_i \neq x_j, \text{ for } 1 \leq i, j \leq n \text{ and } i \neq j.$$

  This is also noted as

  $$f : S_1 \times \cdots \times S_n \mapsto S$$

  This restriction is also called the *elementary* restriction of sorted signatures, and the term declarations are also called *function declarations*.

By a *declaration* we mean either a predicate declaration or a function declaration. The tree-restriction and the restrictions posed in the declarations are quite severe. Netherthe-less they turned out to be useful in practice.

The restrictions stated so far are precisely the same as in [Schmitt and Wernecke, 1989]. The reasons is that our ProSpec transformation coincides with the predicative encoding of sorts of Schmitt and Wernecke, and hence has to make the same assumptions about the sort declarations.

However, we want to be more general and allow *polymorphism* in our declarations (again, this is standard, see [?]). The idea is to allow a *variable* in place where a *sort* is required. These *sort variables* are taken from a set disjoint to the object variables. Further, we would like to have declarations of the form $cons : U \times list(U) \mapsto list(U)$. This motivates the following definition of a *sort*

Definition 2.1 (Polymorphic Sorts)
Let $\mathcal{B} \neq \emptyset$ be a set of symbols, called *basic sorts*, let $\mathcal{U}$ be a set of variables, called *sort variables*, and let $\mathcal{F}$ be a set of function symbols of given arity, called *sort functions*. The set SORT is the smallest set consisting of sorts, where a sort is defined inductively as follows:

- Every basic sort $B \in \mathcal{B}$ is a sort.
- Every variable $U \in \mathcal{U}$ is a sort.
- If $S_1, \ldots, S_n$ are sorts, then $F(S_1, \ldots, S_n)$ is a sort, where $F \in \mathcal{F}$.

$\square$

When taking this modified definition of SORT, the restrictions on sort declarations posed above can remain unchanged, except that the following restricion on subsort declarations has to be made:

Every *subsort declaration* is of the form $S_1 \sqsubseteq S_2$ where $S_1, S_2 \in \mathcal{B}$.

With this restriction, the transitive closure of "$\sqsubseteq$" can be extended to a partial order on SORT (see [?]).

Now, equality is declared as follows: $= \; : U \times U$, where $U \in \mathcal{U}$. Notice that this is an "unusual" declaration of equality, since it permits equations only on terms of the *same* sort (the "usual" declaration would be $= \; : t \times t$ where t is a basic sort and is the least upper bound of all basic sorts). This restriction is motivated by the equality transformation EQTrafo, which is complete for this declaration only.

A further consequence of this declaration concerns semantics: for interpretations it has to be required that different sorts are mapped to domains which do not have any elements in common.

STOP 2.6.97

It has to be clarified what a *well-sorted formula* is. In brief, one has to take care when building formulas that all terms and atoms obey the sort declarations: whenever according to the declaration part a term of sort S is expected, then only a term of sort S or subsort of S may be used. Since this is completely standard we will not repeat the definition here (See e.g. [Oberschelp, 1989]). The only thing to note is that our restrictions imply that each term gets a *unique* sort[1]

With respect to semantics, one has to use *sorted interpretations*. Again, this is standard. The only thing to mention here is that a sorted interpretation $\mathcal{I}$ maps each sort $S \in$ SORT to a *non-empty* domain $\mathcal{I}(S)$.

[1]Our sorted signatures are *subterm-closed*: every subterm of a well-sorted term is well-sorted as well.

ProSpec transforms a well-sorted specification Φ_S into an unsorted clause set Φ in such a way that the semantics of the sorts is preserved. Such transformations are well-known in the literature and are referred to as *sort relativations*. Relativations are expected to satisfy the following property:

Sort theorem: Φ_S is satisfiable with some sort interpretation if and only if Φ is satisfiable in some unsorted interpretation.

Since the relativation carried out by ProSpec satisfies the sort theorem, we thus have a semantics for sorted specifications, by simply taking the semantics of the relativised version.

Designing a sort relativation is a tradeoff between *expressibility* and *efficiency*. In ProSpec we decided to have rather severe restrictions on sort declarations, which, on the other hand, allow for a very simple transformation. Further, and more important, the relativised clause set can be treated very efficiently, because all the reasoning stemming from the sort information is carried out by *ordinary* unification. For instance, we do not allow the declaration that different sorts have a *common* subsort, because this would either need a non-standard unification algorithm or a much more complex transformation.

2.2 ProSpec User's Guide

References

[Baumgartner and Furbach, 1994] Peter Baumgartner and Ulrich Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *Lecture Notes in Aritificial Intelligence*, pages 769–773. Springer, 1994. Available in the WWW, URL: `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

[Oberschelp, 1989] Arnold Oberschelp. Order sorted predicate logic. In *Sorts and Types in Artificial Intelligence*, volume 418 of *Lecture Notes in Aritificial Intelligence*, pages 8–17. Springer, 1989.

[Schmitt and Wernecke, 1989] P.H. Schmitt and W. Wernecke. Tableau calculus for sorted logics. In *Sorts and Types in Artificial Intelligence*, volume 418 of *Lecture Notes in Aritificial Intelligence*, pages 49–60. Springer, 1989.