Predicate Subtyping in HOL

Joe Hurd

University of Cambridge

1. Motivation
2. Architecture
3. Automation
4. Contextual Rewriter
5. Set Membership Prover
6. Comparison with PVS
7. Evaluation

Motivation

A predicate subtype $\mathbf{P} : \alpha \to \mathbb{B}$ is the set of elements x in the simple type α that satisfy $\mathbf{P}\ x$. They are used to refine the type-system.

Here are some examples:

$$\forall\, x \in \mathbf{even}.\ \exists\, p\, q \in \mathbf{prime}.\ 4 \leq x \ \Rightarrow\ x = p + q$$

$$/ \in \mathbf{real} \to \mathbf{nzreal} \to \mathbf{real}$$

Note: **real** has simple type $\mathbb{R} \to \mathbb{B}$ and always returns true.

Predicate subtyping is very useful for formalizing abstract algebra:

$$\begin{aligned} &\forall\, \mathbf{G}\, *.\ \mathrm{group}\ (\mathbf{G}, *)\ \Rightarrow \\ &\qquad * \in \mathbf{G} \to \mathbf{G} \to \mathbf{G}\ \wedge \\ &\qquad \forall\, x\, y\, z \in \mathbf{G}.\ (x * y) * z = x * (y * z) \end{aligned}$$

Architecture

Two competing predicate subtyping architectures:

- **PVS:** Predicate subtyping is part of the type-system, making it undecidable, and the user must prove theorems to show all the terms are well-typed. Mike Jones' work emulated this approach in HOL.
- **HOL:** The type-system is decidable, and any predicate subtyping must be explicit in the terms. During verification, 'type-checking' subgoals will naturally arise. Wai Wong's restricted quantifier library falls into this category.

Our work extends the second design.

Automation

How can we apply the theorem

$$\forall x \in \mathbf{nzreal}.\ x/x = 1$$

to y/y?

We must prove the condition $y \in \mathbf{nzreal}$.

However, if the term y/y type-checks according to the predicate subtype of $/$, we already know this must be true.

Therefore we can safely perform the rewrite, and assume the condition.

Exactly the same situation arises with restricted beta reduction:

$$\begin{aligned} &\Gamma \vdash (\lambda x \in \mathbf{P}.\ M\ x)\ N \\ &\quad \to \quad \Gamma \cup \{\mathbf{P}\ N\} \vdash M\ N \end{aligned}$$

Contextual Rewriter

We have implemented a contextual rewriter, so that the assumptions that are made have the proper logical context at the top-level.

For example, applying the theorem

$$\forall x \in \mathbf{nzreal}.\ x/x = 1$$

to the term

$$P\ y \Rightarrow Q\ (y/y)$$

yields

$$\{P\ y \Rightarrow y \in \mathbf{nzreal}\} \vdash P\ y \Rightarrow Q\ 1$$

These type-checking assumptions that are made during rewriting are passed on to the user as extra subgoals.

Set Membership Prover

Many of the extra type-checking subgoals are trivially solved, and we have implemented a naive prover. It works by collecting facts of the form $x \in S$ and $S \subset T$, and executing a fixed-depth prolog search with the following rules:

$$
\begin{array}{rcl}
 & & x \in \mathrm{UNIV} \\
 & & x \in (x\ \mathrm{INSERT}\ S) \\
x \in S & \Rightarrow & x \in (y\ \mathrm{INSERT}\ S) \\
\mathbf{f} \in (\mathbf{S} \to \mathbf{T}) \ \wedge \ \mathbf{x} \in \mathbf{S} & \Rightarrow & \mathbf{f}\ \mathbf{x} \in \mathbf{T} \\
S \subset T \ \wedge \ x \in S & \Rightarrow & x \in T \\
x \in S \ \wedge \ x \in T & \Rightarrow & x \in (S \cap T) \\
x \in S & \Rightarrow & x \in (S \cup T) \\
x \in T & \Rightarrow & x \in (S \cup T) \\
x \in S & \Rightarrow & f\ x \in (\mathrm{IMAGE}\ f\ S)
\end{array}
$$

This was sufficient to solve automatically every type-checking subgoal that arose in my development.

Comparison with PVS

PVS	**HOL + these tools**
Permanent layer	Phantom layer
Part of type-system	Atop simple type theory
All terms must subtype-check	Subtype-checking is not enforced (or enforceable)
Type-checking phase	As properties are needed
TV licence	Pay-per-view (could end up paying more)
Finds bugs in specs before verification	These same bugs will only appear at verification time (sooner using these tools)
GRIND	Contextual rewriter
Type judgements	Set membership prover
One type per constant	Unlimited (be careful though!)

Evaluation

- 1000 line group theory development using restricted quantifiers and these tools.
- Full predicate subtyping has always been possible in HOL. Restricted quantifiers simplified the notation; these tools increase the level of automation.
- Relative proof cost compared to an explicit type-checking phase is lowest when predicate subtyping is kept to a minimum.
- More predicate subtyping gives more debugging benefits, but also more type-checking subgoals.
- Moral: need automatic type-checking tools (like the set membership prover) that handle virtually every case.