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

Here are some examples:

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

$$/ \in \text{real} \rightarrow \text{nzreal} \rightarrow \text{real}$$

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

Predicate subtyping is very useful for formalizing abstract algebra:

$$\forall G \ast. \text{group } (G, \ast) \Rightarrow$$

$$\ast \in G \rightarrow G \rightarrow G \land$$

$$\forall x y z \in G. \ (x \ast y) \ast z = x \ast (y \ast z)$$
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.
How can we apply the theorem

\[ \forall x \in \text{nzreal}. \, \frac{x}{x} = 1 \]

to \( y/y \)?

We must prove the condition \( y \in \text{nzreal} \).

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:

\[
\Gamma \vdash (\lambda x \in P. \, M \, x) \, N \\
\rightarrow \quad \Gamma \cup \{P \, N\} \vdash M \, N
\]
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

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

to the term

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

yields

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

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 \( S \subseteq T \), and executing a fixed-depth prolog search with the following rules:

\[
\begin{align*}
    x & \in \text{UNIV} \\
    x & \in (x \ \text{INSERT} \ S) \\
    x & \in S \implies x \in (y \ \text{INSERT} \ S) \\
    f \in (S \rightarrow T) \land x \in S & \implies f \ x \in T \\
    S \subseteq T \land x \in S & \implies x \in T \\
    x \in S \land x \in T & \implies x \in (S \cap T) \\
    x \in S & \implies x \in (S \cup T) \\
    x \in T & \implies x \in (S \cup T) \\
    x \in S & \implies f \ x \in (\text{IMAGE} \ f \ S)
\end{align*}
\]

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

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