An Experiment in Automated Theorem Proving in Type Theory

Marcin Benke, Fredrik Lindblad

13th April 2004

- A tool, Agsy, for automated proving in the Agda/Alfa system.
- Agda is a dependent functional programming language based on Martin-Löf type theory.
- Alfa adds means for interactive proving as well as a GUI.
- Agsy produces proof terms which can be verified by the user or by an independent program.

Related Work

Coq

The tactics of Agsy are similar to the elementary tactics of Coq. The `auto` tactic of Coq doesn't do induction or generalization.

ACL2

ACL2 implements FOL. Does induction and generalization mechanically. Does not produce formal proofs.

Gandalf

Optimized encodings into FOL by Tammet and Smith. Implementation part of Gandalf. Does induction, but the complexity tends to become unrelated to that of the original problem.

Comparison and Unification

- Apart from head normalization we use a second kind of reduction, *simplification*.
- At simplification, reduction is performed on the subexpressions.
- As equality, syntactical comparison is done on simplified types.
- Unification in type theory is higher-order. This kind of comparison essentially escapes that.

Refinement Rules

- Used for the elementary steps of the proof search.
- There are refinement rules for introduction, elimination, case analysis/induction, generalization and equalities.
- Each refinement contains zero or more unknown subexpressions. These are classified as either proof or parameter terms.
- Search is performed only for proofs. Parameters are bound at unification.

Example – Multi-Level Induction

$$a + b == b + a$$

First, induction on a. In the inductive step, let $a = \mathrm{s}\ a'$.

IH : $\underline{a' + b} == b{+}a' \quad \vdash \quad \mathrm{s}\ (\underline{a' + b}) == b{+}\mathrm{s}\ a' \quad \Rightarrow \quad \mathrm{s}\ (b{+}a') == b{+}\mathrm{s}\ a'$

Proof term:

let $r\ a\ b\ :\ (a + b == b + a)\ =$ **case** a **of**

$\mathsf{0} \rightarrow$ **let** $r'\ b\ :\ (b == b + \mathsf{0})\ =$ **case** b **of** $\cdots$

in $r'\ b$

$\mathrm{s}\ a' \rightarrow$ [*equality rewriting*]

let $r'\ b\ a'\ :\ (\mathrm{s}\ (b + a') == b + \mathrm{s}\ a')\ =$ **case** b **of** $\cdots$

in $r'\ b\ a'$

in $r\ a\ b$

Example – Generalization

$$\mathit{rev}\,(\mathit{rev}\;\mathit{xs}) == \mathit{xs}$$

Induction on *xs*. In the inductive step, let $\mathit{xs} = x :: \mathit{xs}'$.

$$\mathit{IH} : \mathit{rev}\,(\mathit{rev}\;\mathit{xs}') == \underline{\mathit{xs}'} \quad \vdash \quad \mathit{rev}\,(\mathit{rev}\;\mathit{xs}' +\!+ \;(x :: [\,])) == x :: \underline{\mathit{xs}'}$$

$$\mathit{rev}\,(\underline{\mathit{rev}\;\mathit{xs}'} +\!+ \;(x :: [\,])) == x :: \mathit{rev}\,(\underline{\mathit{rev}\;\mathit{xs}'})$$

$$\mathit{rev}\,(\mathit{ys} +\!+ \;(x :: [\,])) == x :: \mathit{rev}\;\mathit{ys}$$

Proof term:

$$\begin{array}{l} \textbf{let}\; g\;\mathit{ys} \;:\; \mathit{rev}\,(\mathit{ys} +\!+ \;(x :: [\,])) == x :: \mathit{rev}\;\mathit{ys} \\ \quad g\;\mathit{ys} \;=\; \cdots \\ \textbf{in}\;\; g\,(\mathit{rev}\;\mathit{xs}') \end{array}$$

Example – Higher-Order Problems

$$\textit{subst} \;:\; (P : N \to *) \to (x_1, x_2 : N) \to (h : x_1 == x_2) \to$$
$$(h' : P\, x_1) \to P\, x_2$$

In the inductive step, let $x_1 = \mathsf{s}\; x'_1$, $x_2 = \mathsf{s}\; x'_2$.

$$[\textit{subst} \;:\; \cdots, \cdots] \quad \vdash \quad P\,(\mathsf{s}\; x'_2)$$

$$subst\, ?_1\; ?_2\; ?_3\; ?_4\; ?_5 \;: ?_1\; ?_3 \qquad\qquad ?_1\; ?_3 \doteq P\,(\mathsf{s}\; x'_2)$$

$$?_1 = \lambda(x : N) \to P\,(\mathsf{s}\; x'_2) \qquad\qquad \text{or}$$
$$?_1 = \lambda(x : N) \to P\, x, \quad ?_3 = \mathsf{s}\; x'_2 \qquad \text{or}$$
$$?_1 = \lambda(x : N) \to P\,(\mathsf{s}\; x), \quad ?_3 = x'_2 \qquad\qquad \Leftarrow$$

Proof term:

$$\textit{subst}\,(\lambda(x : N) \to P\,(\mathsf{s}\; x))\; x'_1\; x'_2\; h\; h'$$

Bigger Examples

Time consumption measured as number of refinements.

Small problems – ~ 50

Larger problems – ~ 10000

Quicksort

The correctness of the algorithm proved by dividing the problem into approx. 10 smaller ones.

Integer

Establishing the fundamental properties of integers.

Toy Compiler

The correctness of a stack based expression evaluator proved in one step.

Definitions

$$
\begin{aligned}
0 + b &= b \\
\mathrm{s}\ a + b &= \mathrm{s}\ (a + b)
\end{aligned}
$$

$$
\begin{aligned}
\textit{rev}\ [\,] &= [\,] \\
\textit{rev}\ (x :: \textit{xs}) &= \textit{rev}\ \textit{xs} ++ \ (x :: [\,]) \\
[\,] ++ \ \textit{ys} &= \textit{ys} \\
(x :: \textit{xs}) ++ \ \textit{ys} &= x :: (\textit{xs} ++ \ \textit{ys})
\end{aligned}
$$