Advanced Topics in Software Verification

Gerwin Klein
Formal Methods
Intro & motivation, getting started with Isabelle

Foundations & Principles
- Lambda Calculus
- Higher Order Logic, natural deduction
- Term rewriting

Proof & Specification Techniques
- Inductively defined sets, rule induction
- Datatypes, recursion, induction
- Calculational reasoning, mathematics style proofs
- Hoare logic, proofs about programs
LAST TIME ON HOL

- Defining HOL
- Higher Order Abstract Syntax
- Deriving proof rules
- More automation
THE THREE BASIC WAYS OF INTRODUCING THEOREMS

→ Axioms:

Example: \texttt{axioms refl: \textasciitilde\textasciitilde} \texttt{t = t}''

Do not use. Evil. Can make your logic inconsistent.

→ Definitions:

Example: \texttt{defs inj\_def: \textasciitilde\textasciitilde} \texttt{\textasciitilde\textasciitilde} \texttt{inj f \equiv \forall x y. f x = f y \rightarrow x = y}''

→ Proofs:

Example: \texttt{lemma \textasciitilde\textasciitilde} \texttt{inj (\lambda x. x + 1)}''

The harder, but safe choice.
The Three Basic Ways of Introducing Types

→ **typedecl**: by name only

Example: `typedecl names`  
Introduces new type *names* without any further assumptions

→ **types**: by abbreviation

Example: `types α rel = "α ⇒ α ⇒ bool"`  
Introduces abbreviation *rel* for existing type $\alpha \Rightarrow \alpha \Rightarrow \text{bool}$  
Type abbreviations are immediately expanded internally

→ **typedef**: by definition as a set

Example: `typedef new_type = "\{some set\}" <proof>`  
Introduces a new type as a subset of an existing type.  
The proof shows that the set on the rhs is non-empty.
HOW TYPEDEF WORKS

new type

existing type

Rep

Abs
HOW typedef WORKS

new type

existing type

Rep

Abs
EXAMPLE: PAIRS

\[(\alpha, \beta) \text{ Prod}\]

1. Pick existing type: \(\alpha \Rightarrow \beta \Rightarrow \text{bool}\)
2. Identify subset:

\[(\alpha, \beta) \text{ Prod} = \{ f. \exists a b. f = \lambda(x :: \alpha) (y :: \beta). x = a \land y = b\}\]

3. We get from Isabelle:
   - functions Abs_Prod, Rep_Prod
   - both injective
   - Abs_Prod (Rep_Prod \(x\)) = \(x\)

4. We now can:
   - define constants Pair, fst, snd in terms of Abs_Prod and Rep_Prod
   - derive all characteristic theorems
   - forget about Rep/Abs, use characteristic theorems instead
DEMO: INTRODUCTING NEW TYPES
TERM REWRITING
Given a set of equations

\[ l_1 = r_1 \]
\[ l_2 = r_2 \]
\[ \vdots \]
\[ l_n = r_n \]

does equation \( l = r \) hold?

Applications in:

- **Mathematics** (algebra, group theory, etc)
- **Functional Programming** (model of execution)
- **Theorem Proving** (dealing with equations, simplifying statements)
use equations as reduction rules

\[ l_1 \rightarrow r_1 \]
\[ l_2 \rightarrow r_2 \]
\[ \vdots \]
\[ l_n \rightarrow r_n \]

decide \( l = r \) by deciding \( l \leftrightarrow^* r \)
\[\begin{align*}
0 \rightarrow &= \{(x, y) | x = y\} \quad \text{identity} \\
n+1 \rightarrow &= n \rightarrow \circ \rightarrow \quad \text{n+1 fold composition} \\
+ \rightarrow &= \bigcup_{i > 0} i \rightarrow \quad \text{transitive closure} \\
* \rightarrow &= + \rightarrow \cup \rightarrow 0 \rightarrow \quad \text{reflexive transitive closure} \\
\rightarrow &= \rightarrow \cup \rightarrow 0 \rightarrow \quad \text{reflexive closure} \\
-1 \rightarrow &= \{(y, x) | x \rightarrow y\} \quad \text{inverse} \\
\leftarrow &= -1 \rightarrow \quad \text{inverse} \\
\leftrightarrow &= \leftrightarrow \cup \leftrightarrow \quad \text{symmetric closure} \\
\leftarrow+ \rightarrow &= \bigcup_{i > 0} \leftarrow i \rightarrow \quad \text{transitive symmetric closure} \\
\leftrightarrow+ \rightarrow &= \leftrightarrow + \rightarrow \cup \leftrightarrow 0 \rightarrow \quad \text{reflexive transitive symmetric closure}
\end{align*}\]
How to Decide $l \leftrightarrow^* r$

Same idea as for $\beta$: look for $n$ such that $l \rightarrow^* n$ and $r \rightarrow^* n$

Does this always work?
- If $l \rightarrow^* n$ and $r \rightarrow^* n$ then $l \leftrightarrow^* r$. Ok.
- If $l \leftrightarrow^* r$, will there always be a suitable $n$? **No!**

Example:

Rules: $f x \rightarrow a$, $g x \rightarrow b$, $f (g x) \rightarrow b$

$f x \leftrightarrow^* g x$ because $f x \rightarrow a \leftarrow f (g x) \rightarrow b \leftarrow g x$

But: $f x \rightarrow a$ and $g x \rightarrow b$ and $a, b$ in normal form

Works only for systems with **Church-Rosser** property:

$l \leftrightarrow^* r \iff \exists n. l \rightarrow^* n \land r \rightarrow^* n$

**Fact:** $\rightarrow^*$ is Church-Rosser iff it is confluent.
Problem:
is a given set of reduction rules confluent?

undecidable

Local Confluence

Fact: local confluence and termination $\implies$ confluence
Termination

$\rightarrow$ is **terminating** if there are no infinite reduction chains
$\rightarrow$ is **normalizing** if each element has a normal form
$\rightarrow$ is **convergent** if it is terminating and confluent

Example:
$\rightarrow_\beta$ in $\lambda$ is not terminating, but confluent
$\rightarrow_\beta$ in $\lambda \rightarrow$ is terminating and confluent, i.e. convergent

**Problem**: is a given set of reduction rules terminating?

**undecidable**
Basic Idea: when the $r_i$ are in some way simpler than the $l_i$

More formally: $\rightarrow$ is terminating when there is a well-founded order $<$ in which $r_i < l_i$ for all rules.

(well-founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Example: $f(g \, x) \rightarrow g \, x$, $g(f \, x) \rightarrow f \, x$

This system always terminates. Reduction order:

$s <_r t$ iff $\text{size}(s) < \text{size}(t)$ with

$\text{size}(s) = \text{number of function symbols in } s$

1. $g \, x <_r f(g \, x)$ and $f \, x <_r g(f \, x)$

2. $<_r$ is well-founded, because $<$ is well-founded on $\mathbb{N}$
Term rewriting engine in Isabelle is called **Simplifier**

```
apply simp
```

- uses simplification rules
- (almost) blindly from left to right
- until no rule is applicable.

**termination:** not guaranteed
(may loop)

**confluence:** not guaranteed
(result may depend on which rule is used first)
Equations turned into simplification rules with \texttt{[simp]} attribute

Adding/deleting equations locally:
\begin{verbatim}
apply (simp add: \texttt{<rules>}) and apply (simp del: \texttt{<rules>})
\end{verbatim}

Using only the specified set of equations:
\begin{verbatim}
apply (simp only: \texttt{<rules>})
\end{verbatim}
DEMO