Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

Slide 1

Content

- Intro & motivation, getting started
- Foundations & Principles
  - Lambda Calculus, natural deduction
  - Higher Order Logic
  - Term rewriting
- Proof & Specification Techniques
  - Isar
  - Inductively defined sets, rule induction
  - Datatypes, recursion, induction
  - Calculational reasoning, mathematics style proofs
  - Hoare logic, proofs about programs

Rough timeline

- [1]
- [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]

Slide 2

Last Time on HOL

- Defining HOL
- Higher Order Abstract Syntax
- Deriving proof rules
- More automation

Slide 3

The Three Basic Ways of Introducing Theorems

- Axioms:
  Example: \texttt{axioms refl: "t = t"}
  Do not use. Evil. Can make your logic inconsistent.

- Definitions:
  Example: \texttt{definition inj where "inj f ≡ ∀ x y. f x = f y → x = y"}
  Introduces a new lemma called \texttt{inj}.

- Proofs:
  Example: \texttt{lemma "inj (λ x. x + 1)"}
  The harder, but safe choice.

Slide 4
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 `α → α → 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.
  - More on `typedef` in later lectures.

---

The Problem

Given a set of equations

\[
\begin{align*}
l_1 &= r_1 \\
l_2 &= r_2 \\
\vdots \\
l_n &= r_n
\end{align*}
\]

does equation \( l = r \) hold?

Applications in:

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

---

Term Rewriting: The Idea

use equations as reduction rules

\[
\begin{align*}
l_1 &\rightarrow r_1 \\
l_2 &\rightarrow r_2 \\
\vdots \\
l_n &\rightarrow r_n
\end{align*}
\]

decide \( l = r \) by deciding \( l \xrightarrow{\cdot} r \)
Arrow Cheat Sheet

- $\frac{n}{n+1} = \{(x,y) | x = y\}$ identity
- $\frac{n}{n+1} = \frac{n}{n} \circ \frac{n}{n+1}$ $n+1$ fold composition
- $\frac{1}{1} = \frac{1}{1} \cup \frac{1}{1}$ transitive closure
- $\frac{1}{1} = \frac{1}{1} \cup \frac{1}{1}$ reflexive transitive closure
- $\frac{1}{1} = \frac{1}{1}$ reflexive closure
- $\frac{1}{1} = \{(y,x) | x \rightarrow y\}$ inverse
- $\frac{1}{1} = \frac{1}{1}$ inverse
- $\frac{1}{1} = \frac{1}{1} \cup \frac{1}{1}$ symmetric closure
- $\frac{1}{1} = \frac{1}{1} \cup \frac{1}{1}$ transitive symmetric closure
- $\frac{1}{1} = \frac{1}{1} \cup \frac{1}{1}$ reflexive transitive symmetric closure

How to Decide $l \rightarrow r$

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 $l \rightarrow r$. Ok.
- If $l \rightarrow r$, will there always be a suitable $n$? No!

Example:

Rules:
- $f x \rightarrow a$, $g z \rightarrow b$, $f (g x) \rightarrow b$
- $f x \rightarrow g x$ because $f x \rightarrow a \rightleftharpoons f (g x) \rightarrow b$ $\rightarrow g x$
- But: $f x \rightarrow a$ and $g x \rightarrow b$ and $a, b$ in normal form

Works only for systems with Church-Rosser property:

$\frac{l}{r} \rightleftharpoons \exists n, l \rightarrow n \land r \rightarrow n$

Fact: $\rightarrow$ is Church-Rosser iff it is confluent.

Confluence

Problem: is a given set of reduction rules confluent?

undecidable

Local Confluence

Fact: local confluence and termination $\Rightarrow$ 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:

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

Problem: is a given set of reduction rules terminating?

undecidable
When is $\rightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

More formally: $\rightarrow$ is terminating when there is a well founded order $<\,\neq\,\text{no infinite decreasing chains } a_1 \,\geq\, a_2 \,\geq\, \ldots$.

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

- both rules always decrease size by 1 when applied to any term $t$
- $<$ is well founded, because $<$ is well founded on $\mathbb{N}$

Example Termination Proof

Problem: Rewrite formulae containing $\neg, \land, \lor$ and $\rightarrow$, so that they don’t contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

- Remove implications:
  - $\text{imp}: \, (A \rightarrow B) = (\neg A \lor B)$
- Push $\neg s$ down past other operators:
  - $\text{notnot}: \, (\neg\neg P) = P$
  - $\text{notand}: \, (\neg(A \land B)) = (\neg A \lor \neg B)$
  - $\text{notor}: \, (\neg(A \lor B)) = (\neg A \land \neg B)$

We show that the rewrite system defined by these rules is terminating.

Order on Terms

Each time one of our rules is applied, either:

- an implication is removed, or
- something that is not $\neg$ is hoisted upwards in the term.

This suggests a 2-part order, $<\,\neq\, s < r t$ iff:

- $\text{num}\_\text{imps} \, s < r \,\text{num}\_\text{imps} \, t$, or
- $\text{num}\_\text{imps} \, s = \text{num}\_\text{imps} \, t \land \text{size} \, s < \text{size} \, t$.

Let:

- $s < r t \equiv \text{num}\_\text{imps} \, s < \text{num}\_\text{imps} \, t$ and
- $s < r t \equiv \text{size} \, s < \text{size} \, t$

Then $<\,\land\, <\,\text{num}\_\text{imp}$ are both well-founded orders (since both functions return nat).

$<\,\text{lex}$ is the lexicographic order over $<\,\land\, <\,\text{num}\_\text{imp}$ and is well-founded since $<\,$ and $\text{num}\_\text{imp}$ are both well-founded.
Order Decreasing

imp clearly decreases num_imps.

osize adds up all non-operators and variables/constants, weights each one according to its depth within the term.

\[
\begin{align*}
\text{osize'}(\epsilon) & = acm = 2^{acm} \\
\text{osize'}(\neg P) & = acm = \text{osize'} P (acm + 1) \\
\text{osize'}(P \land Q) & = acm = 2^{acm} + (\text{osize'} P (acm + 1)) + (\text{osize'} Q (acm + 1)) \\
\text{osize'}(P \lor Q) & = acm = 2^{acm} + (\text{osize'} P (acm + 1)) + (\text{osize'} Q (acm + 1)) \\
\text{osize'}(P \rightarrow Q) & = acm = 2^{acm} + (\text{osize'} P (acm + 1)) + (\text{osize'} Q (acm + 1)) \\
\text{osize'} P & = \text{osize'} P 0
\end{align*}
\]

The other rules decrease the depth of the things osize counts, so decrease osize.

---

Term Rewriting in Isabelle

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)

---

Control

- Equations turned into simplification rules with [simp] attribute
- Adding/deleting equations locally:
  apply (simp add: <rules>) and apply (simp del: <rules>)
- Using only the specified set of equations:
  apply (simp only: <rules>)

---

DEMO
We have seen today...

- Equations and Term Rewriting
- Confluence and Termination of reduction systems
- Term Rewriting in Isabelle

Exercises

- Show, via a pen-and-paper proof, that the osize function is monotonic with respect to the structure of terms from that example.