COMP 4161

NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

$\longrightarrow$

Slide 1

Content

Rough timeline

- ➜ Intro & motivation, getting started [1]
- ➜ Foundations & Principles
  - Lambda Calculus, natural deduction [2,3,4[a]]
  - Higher Order Logic [5,6[b],7]
  - Term rewriting [8,9,10[c]]
- ➜ Proof & Specification Techniques
  - Isar [11,12[d]]
  - Inductively defined sets, rule induction [13[e],15]
  - Datatypes, recursion, induction [16,17[f],18,19]
  - Calculational reasoning, mathematics style proofs [20]
  - Hoare logic, proofs about programs [21[g],22,23]

[a] a1 out; [b] a1 due; [c] a2 out; [d] a2 due; [e] session break; [f] a3 out; [g] a3 due

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**:

Expample: **axioms** refl: "$t = t$"

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

➜ **Definitions:**

Example: **definition** inj **where** "inj $f \equiv \forall x\ y.\ f\ x = f\ y \longrightarrow x = y$"
Introduces a new lemma called inj_def.

➜ **Proofs:**

Example: **lemma** "inj $(\lambda x.\ x + 1)$"

The harder, but safe choice.

Slide 4

The Three Basic Ways of Introducing Types

NICTA

- **typedecl**: by name only

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

- **types**: by abbreviation

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

- **typedef**: by definiton 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 in non-empty.

  More on **typedef** in later lectures.

Slide 5

NICTA

TERM REWRITING

Slide 6

The Problem

NICTA

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)

Slide 7

Term Rewriting: The Idea

NICTA

use equations as reduction rules

$$l_1 \longrightarrow r_1$$
$$l_2 \longrightarrow r_2$$
$$\vdots$$
$$l_n \longrightarrow r_n$$

decide $l = r$ by deciding $l \stackrel{*}{\longleftrightarrow} r$

Slide 8

Arrow Cheat Sheet

$\stackrel{0}{\longrightarrow}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\stackrel{n+1}{\longrightarrow}$	$=$	$\stackrel{n}{\longrightarrow} \circ \longrightarrow$	n+1 fold composition
$\stackrel{+}{\longrightarrow}$	$=$	$\bigcup_{i>0} \stackrel{i}{\longrightarrow}$	transitive closure
$\stackrel{*}{\longrightarrow}$	$=$	$\stackrel{+}{\longrightarrow} \cup \stackrel{0}{\longrightarrow}$	reflexive transitive closure
$\stackrel{=}{\longrightarrow}$	$=$	$\longrightarrow \cup \stackrel{0}{\longrightarrow}$	reflexive closure
$\stackrel{-1}{\longrightarrow}$	$=$	$\{(y,x) \mid x \longrightarrow y\}$	inverse
$\longleftarrow$	$=$	$\stackrel{-1}{\longrightarrow}$	inverse
$\longleftrightarrow$	$=$	$\longleftarrow \cup \longrightarrow$	symmetric closure
$\stackrel{+}{\longleftrightarrow}$	$=$	$\bigcup_{i>0} \stackrel{i}{\longleftrightarrow}$	transitive symmetric closure
$\stackrel{*}{\longleftrightarrow}$	$=$	$\stackrel{+}{\longleftrightarrow} \cup \stackrel{0}{\longleftrightarrow}$	reflexive transitive symmetric closure

Slide 9

Confluence

$$\begin{tikzcd} & s \arrow[dl, "*"'] \arrow[dr, "*"] & \\ x \arrow[dr, dashed, "*"'] & & y \arrow[dl, dashed, "*"] \\ & t & \end{tikzcd}$$

Problem:
is a given set of reduction rules confluent?

undecidable

Local Confluence

$$\begin{tikzcd} & s \arrow[dl] \arrow[dr] & \\ x \arrow[dr, dashed, "*"'] & & y \arrow[dl, dashed, "*"] \\ & t & \end{tikzcd}$$

Fact: local confluence and termination $\Longrightarrow$ confluence

Slide 11

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

If $l \stackrel{*}{\longleftrightarrow} r$, will there always be a suitable n? **No!**

Example:

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

$f\ x \stackrel{*}{\longleftrightarrow} g\ x$ because $f\ x \longrightarrow a \longleftarrow f\ (g\ x) \longrightarrow b \longleftarrow g\ x$

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

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

$$l \stackrel{*}{\longleftrightarrow} r \Longrightarrow \exists n.\ l \stackrel{*}{\longrightarrow} n \wedge r \stackrel{*}{\longrightarrow} n$$

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

Slide 10

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains

$\longrightarrow$ is **normalizing** if each element has a normal form

$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:

$\longrightarrow_\beta$ in λ is not terminating, but confluent

$\longrightarrow_\beta$ in $\lambda^\rightarrow$ is terminating and confluent, i.e. convergent

Problem: is a given set of reduction rules terminating?

undecidable

Slide 12

When is $\longrightarrow$ Terminating?

NICTA

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

More formally: $\longrightarrow$ is terminating when
there is a well founded order $<$ on terms for which $s < t$ whenever $t \longrightarrow s$
(well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

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

This system always terminates. Reduction order:

$s <_r t$ iff $size(s) < size(t)$ with
$size(s) =$ number of function symbols in s

① Both rules always decrease $size$ by 1 when applied to any term t
② $<_r$ is well founded, because $<$ is well founded on $\mathbb{N}$

Slide 13

Termination in Practice

NICTA

In practice: often easier to consider just the rewrite rules by themselves,
rather than their application to an arbitrary term t.

Show for each rule $l_i = r_i$, that $r_i < l_i$.

Example:
$g\ x <_r f\ (g\ x)$ and $f\ x < g\ (f\ x)$

Requires t to become smaller whenever any subterm of t is made smaller.

Formally:
Requires $<$ to be **monotonic** with respect to the structure of terms:
$s < t \longrightarrow u[s] < u[t]$.

True for most orders that don't treat certain parts of terms as special cases.

Slide 14

Example Termination Proof

NICTA

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

➜ Remove implications:

imp: $(A \longrightarrow B) = (\neg A \vee B)$

➜ Push $\neg$s down past other operators:

notnot: $(\neg\neg P) = P$

notand: $(\neg(A \wedge B)) = (\neg A \vee \neg B)$

notor: $(\neg(A \vee B)) = (\neg A \wedge \neg B)$

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

Slide 15

Order on Terms

NICTA

Each time one of our rules is applied, either:

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

This suggests a 2-part order, $<_r$: $s <_r t$ iff:

➜ num_imps $s <$ num_imps t, or
➜ num_imps $s =$ num_imps $t \wedge$ osize $s <$ osize t.

Let:

➜ $s <_i t \equiv$ num_imps $s <$ num_imps t and
➜ $s <_n t \equiv$ osize $s <$ osize t

Then $<_i$ and $<_n$ are both well-founded orders (since both functions return nats).

$<_r$ is the lexicographic order over $<_i$ and $<_n$. $<_r$ is well-founded since $<_i$ and $<_n$ are both well-founded.

Slide 16

Order Decreasing

imp clearly decreases num_imps.

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

$$
\begin{array}{lll}
\mathsf{osize}'\ c & \mathsf{acm} = 2^{\mathsf{acm}} & \\
\mathsf{osize}'\ (\neg P) & \mathsf{acm} = \mathsf{osize}'\ P\ (\mathsf{acm}+1) & \\
\mathsf{osize}'\ (P \wedge Q) & \mathsf{acm} = 2^{\mathsf{acm}} + (\mathsf{osize}'\ P\ (\mathsf{acm}+1)) + (\mathsf{osize}'\ Q\ (\mathsf{acm}+1)) & \\
\mathsf{osize}'\ (P \vee Q) & \mathsf{acm} = 2^{\mathsf{acm}} + (\mathsf{osize}'\ P\ (\mathsf{acm}+1)) + (\mathsf{osize}'\ Q\ (\mathsf{acm}+1)) & \\
\mathsf{osize}'\ (P \longrightarrow Q) & \mathsf{acm} = 2^{\mathsf{acm}} + (\mathsf{osize}'\ P\ (\mathsf{acm}+1)) + (\mathsf{osize}'\ Q\ (\mathsf{acm}+1)) & \\
\mathsf{osize}\ \ P & \qquad\ = \mathsf{osize}'\ P\ 0 &
\end{array}
$$

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.