COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification
Toby Murray, June Andronick, Gerwin Klein

Slide 1

Content

- Intro & motivation, getting started [1]
- Foundations & Principles
  - Lambda Calculus, natural deduction [1,2]
  - Higher Order Logic [3,4]
  - Term rewriting [4]
- Proof & Specification Techniques
  - Inductively defined sets, rule induction [5]
  - Datatypes, recursion, induction [6, 7]
  - Hoare logic, proofs about programs, C verification [8, 9]
  - (mid-semester break)
- Writing Automated Proof Methods [10]

\*a1 due, *a2 due, *a3 due

Last Time

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

Slide 2

Applying a Rewrite Rule

- \( l \rightarrow r \) applicable to term \( t[s] \)
  - if there is substitution \( \sigma \) such that \( \sigma l = s \)
- Result: \( t[\sigma r] \)
- Equationally: \( t[s] = t[\sigma r] \)

Example:

Rule: \( 0 + n \rightarrow n \)

Term: \( a + (0 + (b + c)) \)

Substitution: \( \sigma = \{ n \mapsto b + c \} \)

Result: \( a + (b + c) \)
Conditional Term Rewriting

Rewrite rules can be conditional:

\[ [P_1 \ldots P_n] \Rightarrow l = r \]

is applicable to term \( t|\sigma \) with \( \sigma \) if

\[ \sigma l = s \quad \text{and} \quad \sigma P_1, \ldots, \sigma P_n \] are provable by rewriting.

Preprocessing

Preprocessing (recursive) for maximal simplification power:

\[ \neg A \Rightarrow A = False \]
\[ A \Rightarrow B \Rightarrow A \Rightarrow B \]
\[ A \land B \Rightarrow A, B \]
\[ \forall x. A x \Rightarrow A ? x \]
\[ A \Rightarrow A = True \]

Example:

\[ (p \Rightarrow q \land \neg r) \land s \]
\[ \Rightarrow \]
\[ p = q = True \quad p = r = False \quad s = True. \]

Rewriting with Assumptions

Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

\[ \text{lemma} \ "f x = g x \land g x = f x \Rightarrow f x = 2" \]

\[ \text{simp} \quad \text{use and simplify assumptions} \]
\[ \text{(simp (no_asm))} \quad \text{ignore assumptions} \]
\[ \text{(simp (no_asm_use))} \quad \text{simplify, but do not use assumptions} \]
\[ \text{(simp (no_asm_simp))} \quad \text{use, but do not simplify assumptions} \]

Copyright NICTA 2014, provided under Creative Commons Attribution License
Case splitting with simp

\[ P \text{ (if } A \text{ then } s \text{ else } t) = (A \rightarrow P s) \land (\neg A \rightarrow P t) \]

Automatic

\[ P \text{ (case } c \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) = (c = 0 \rightarrow P a) \land (\forall n. c = \text{Suc } n \rightarrow P b) \]

Manually: apply (simp split: nat.split)

Similar for any data type \( t \): \( t \cdot \text{split} \)

Slide 9

Congruence Rules

Congruence rules are about using context

Example: in \( P \rightarrow Q \) we could use \( P \) to simplify terms in \( Q \)

For \( \Rightarrow \) hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: \[ [P = P'; P' \rightarrow Q = Q'] \Rightarrow (P \rightarrow Q) = (P' \rightarrow Q') \]

Read: to simplify \( P \rightarrow Q \)

\( \Rightarrow \) first simplify \( P \) to \( P' \)
\( \Rightarrow \) then simplify \( Q \) to \( Q' \) using \( P' \) as assumption
\( \Rightarrow \) the result is \( P' \rightarrow Q' \)

Slide 10

More Congruence

Sometimes useful, but not used automatically (slowdown):

\( \text{cong} \): \[ [P = P'; P' \rightarrow Q = Q'] \Rightarrow (P \rightarrow Q) = (P' \rightarrow Q') \]

Context for if-then-else:

\( \text{if_cong} \): \[ b = c \Rightarrow x = v; \neg c \Rightarrow y = v \Rightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y) \]

Prevent rewriting inside then-else (default):

\( \text{if_weak_cong} \): \[ b = c \Rightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y) \]

\( \Rightarrow \) declare own congruence rules with [cong] attribute

\( \Rightarrow \) delete with [cong del]

\( \Rightarrow \) use locally with e.g. apply (simp cong: \(<\text{rule}>\))

Slide 11

Ordered rewriting

Problem: \( x + y \rightarrow y + x \) does not terminate

Solution: use permutative rules only if term becomes lexicographically smaller.

Example: \( b + a \sim a + b \) but not \( a + b \sim b + a \).

For types nat, int etc:

- lemmas \( \text{add_ac} \) sort any sum (\( + \))
- lemmas \( \text{times_ac} \) sort any product (\( \ast \))

Example: apply (simp add: \( \text{add_ac} \)) yields \( (b + c) + a \sim \cdots \sim a + (b + c) \)

Slide 12
AC Rules

Example for associative-commutative rules:

**Associative:**
\((x \odot y) \odot z = x \odot (y \odot z)\)

**Commutative:**
\(x \odot y = y \odot x\)

These 2 rules alone get stuck too early (not confluent).

Example:
\((z \odot x) \odot (y \odot v)\)

We want:
\((z \odot x) \odot (y \odot v) = v \odot (x \odot (y \odot z))\)

We get:
\((z \odot x) \odot (y \odot v) = v \odot (y \odot (x \odot z))\)

We need: AC rule \(x \odot (y \odot z) = y \odot (x \odot z)\)

If these 3 rules are present for an AC operator Isabelle will order terms correctly

Back to Confluence

Last time: confluence in general is undecidable.

But: confluence for terminating systems is decidable!

Problem: overlapping lhs of rules.

Definition:

Let \(I_1 \rightarrow r_1\) and \(I_2 \rightarrow r_2\) be two rules with disjoint variables.

They form a **critical pair** if a non-variable subterm of \(I_1\) unifies with \(I_2\).

Example:

Rules: (1) \(f x \rightarrow a\) (2) \(g y \rightarrow b\) (3) \(f (g z) \rightarrow b\)

Critical pairs:

\((1)+(3)\) \(\{ x \mapsto g z \}\) \(a \xrightarrow{(1)} f (g z) \xrightarrow{(3)} b\)

\((3)+(2)\) \(\{ z \mapsto y \}\) \(b \xrightarrow{(3)} f (g y) \xrightarrow{(5)} f b\)

Completion

\((1) f x \rightarrow a\) (2) \(g y \rightarrow b\) (3) \(f (g z) \rightarrow b\)

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:

\((1)+(3)\) \(\{ x \mapsto g z \}\) \(a \xrightarrow{(1)} f (g z) \xrightarrow{(3)} b\)

shows that \(a = b\) (because \(a \xrightarrow{(1)} b\)), so we add \(a \rightarrow b\) as a rule

This is the main idea of the Knuth-Bendix completion algorithm.
Orthogonal Rewriting Systems

Definitions:
A rule \( l \rightarrow r \) is left-linear if no variable occurs twice in \( l \).
A rewrite system is left-linear if all rules are.
A system is orthogonal if it is left-linear and has no critical pairs.

Orthogonal rewrite systems are confluent

Application: functional programming languages

We have learned today ...

- Conditional term rewriting
- Congruence rules
- AC rules
- More on confluence

DEMO: WALDMEISTER