COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

Content

Rough timeline

- Intro & motivation, getting started
  - [1]

- Foundations & Principles
  - Lambda Calculus, natural deduction
    - [2,3,4a]
  - Higher Order Logic
    - [5,6,7]
  - Term rewriting
    - [8,9,10c]

- Proof & Specification Techniques
  - Isar
    - [11,12d]
  - Inductively defined sets, rule induction
    - [13*,15]
  - Datatypes, recursion, induction
    - [16,17,18,19]
  - Calculational reasoning, mathematics style proofs
    - [20]
  - Hoare logic: proofs about programs
    - [21*,22,23]

Last Time

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

Applying a Rewrite Rule

- \( \alpha \to \beta \) applicable to term \( t[\sigma] \)
  if there is substitution \( \sigma \) such that \( \sigma \alpha = \beta \)
- Result: \( t[\sigma \beta] \)
- Equationally: \( t[\sigma] = t[\sigma \beta] \)

Example:

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

\( \sigma l = s \) and

\( \sigma P_1, \ldots, \sigma P_n \) are provable by rewriting.

Rewriting with Assumptions

Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

```
lemma "f x = g z \land g x = f x \Rightarrow f x = T"
```

Preprocessing

Preprocessing (recursive) for maximal simplification power:

\[ \neg A \Rightarrow A = \text{False} \]

\[ A \Rightarrow B \Rightarrow A \Rightarrow B \]

\[ A \land B \Rightarrow A, B \]

\[ \forall x. A x \Rightarrow A \ ?x \]

\[ A \Rightarrow A = \text{True} \]

Example:

\( (p \Rightarrow q \land \neg r) \land s \Rightarrow p \Rightarrow q = \text{True} \)

\( p = \text{False} \)

\( s = \text{True} \)
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

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

Manually: apply \((\text{simp split: nat.split})\)

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

---

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 \(P' \rightarrow Q'\)

---

More Congruence

Sometimes useful, but not used automatically (slowdown):

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

Context for if-then-else:

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

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 \([\text{cong}]\) attribute

\(\rightarrow\) delete with \([\text{cong del}]\)

---

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 \text{nat}, \text{int} etc:

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

Example: apply \((\text{simp add: add_ac})\) yields

\((b + c) + a \sim \cdot \sim a + (b + c)\)
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: \((z \odot x) \odot (y \odot v) = v \odot (y \odot (x \odot z))\)

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

Last time: confluence in general is undecidable.
But: confluence for terminating systems is decidable!

Problem: overlapping lhs of rules.

Definition:
Let \(l_1 \rightarrow r_1\) and \(l_2 \rightarrow r_2\) be two rules with disjoint variables. They form a critical pair if a non-variable subterm of \(l_1\) unifies with \(l_2\).

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

Critical pairs:
(1)+(3) \(\{x \rightarrow g z\}\) \(a \leftrightarrow b\)
(3)+(2) \(\{z \rightarrow y\}\) \(b \leftrightarrow b\)

is not confluent

But it can be made confluent by adding rules!
How: join all critical pairs

Example:
(1)+(3) \(\{x \rightarrow g z\}\) \(a \leftrightarrow b\)
shows that \(a = b\) (because \(a \leftrightarrow b\)), 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