COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Simon Winwood, Toby Murray, June Andronick, Gerwin Klein

Slide 1

Content

• 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

Slide 2

Last Time

• Introducing new Types
• Equations and Term Rewriting
• Confluence and Termination of reduction systems
• Term Rewriting in Isabelle

Slide 3

Exercises

• use typedef to define a new type v with exactly one element.
• define a constant u of type v
• show that every element of v is equal to u
• design a set of rules that turns formulas with \( \land, \lor, \rightarrow, \neg \)
  into disjunctive normal form
  (\( \neg \) disjunction of conjunctions with negation only directly on variables)
• prove those rules in Isabelle
• use simp only with these rules on \( (\neg B \rightarrow C) \rightarrow A \rightarrow B \)

Slide 4
ISAR
A LANGUAGE FOR STRUCTURED PROOFS

Slide 5

Isar
apply scripts What about..

➔ unreadable ➔ Elegance?
➔ hard to maintain ➔ Explaining deeper insights?
➔ do not scale ➔ Large developments?

No structure. Isar!

Slide 6

A typical Isar proof

proof
  assume formula₀
  have formula₁ by simp
  ...
  have formulaₙ by blast
  show formulaₙ₊₁ by ...
qed

proves formula₀ ⇒ formulaₙ₊₁
(analogous to assumes/shows in lemma statements)

Slide 7

Isar core syntax

proof = proof [method] statement* qed
  | by method
method = (simp ... ) | (blast ... ) | (rule ... ) | ...
statement = fix variables (\(A\))
  | assume proposition (\(\Rightarrow\))
  | [from name]* (have | show) proposition proof
  | next (separates subgoals)
proposition = [name:] formula

Slide 8
lemma "[A; B] ≤ A ∧ B"
proof (rule conjI)
  assume A: "A"
  from A show "A" by assumption
next
  assume B: "B"
  from B show "B" by assumption
qed

Slide 9

How do I know what to Assume and Show?

Look at the proof state!

lemma "[A; B] ≤ A ∧ B"
proof (rule conjI)
  ⇒ proof (prove) applies method to the stated goal
  ⇒ proof (state) applies a single rule that fits
  ⇒ proof (chain) does nothing to the goal

Slide 10

The Three Modes of Isar

→ [prove]: goal has been stated, proof needs to follow.
→ [state]: proof block has opened or subgoal has been proved.
  new from statement, goal statement or assumptions can follow.
→ [chain]: from statement has been made, goal statement needs to follow.

lemma "[A; B] ≤ A ∧ B" [prove]
proof (rule conjI) [state]
  assume A: "A" [state]
next [state] …

Slide 11

Have

Can be used to make intermediate steps.

Example:
lemma "(x :: nat) + 1 = 1 + x"
proof -
  have A: "x + 1 = Suc x" by simp
  have B: "1 + x = Suc x" by simp
  show "x + 1 = 1 + x" by (simp only: A B)
qed
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 $t[s]$ 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 x ∧ g x = f x ⇒ f x = 2"
```

```
simp use and simplify assumptions
(simp (no_asm)) ignore assumptions
(simp (no_asm_use)) simplify, but do not use assumptions
(simp (no_asm_simp)) use, but do not simplify assumptions
```

Preprocessing

Preprocessing (recursive) for maximal simplification power:

\[\neg A \rightarrow A = \text{False} \]
\[A \rightarrow B \rightarrow A \rightarrow B\]
\[A \land B \rightarrow A, B\]
\[\forall x. A x \rightarrow \forall x. A x\]
\[A \rightarrow A = \text{True}\]

Example:

\[(p \rightarrow q \land \neg r) \land s\]
\[p \rightarrow q = \text{True} \quad r = \text{False} \quad s = \text{True}\]
Congruence Rules

Congruence rules are about using context

Example: in P → Q we could use P to simplify terms in Q

For → hardwired (assumptions used in rewriting)
For other operators expressed with conditional rewriting.

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

Read: to simplify P → Q
→ first simplify P to P'
→ then simplify Q to Q' using P' as assumption
→ the result is P' → Q'

More Congruence

Sometimes useful, but not used automatically (slowdown):
\texttt{conj} \[ (P = P'; P' \rightarrow Q = Q') \rightarrow (P 
\land Q ) = (P' 
\land Q) \]

Context for if-then-else:
\texttt{if} \[ \begin{array}{l}
[b = c; e \rightarrow x = w; e' \rightarrow y = v]\rightarrow \\
(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)
\end{array} \]

Prevent rewriting inside then-else (default):
\texttt{if\_weak\_cong} \[ b = c \rightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)\]

→ declare own congruence rules with \texttt{[cong]} attribute
→ delete with \texttt{[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 nat, int etc:
• \texttt{lemmas add\_ac} sort any sum (+)
• \texttt{lemmas times\_ac} sort any product (·)

Example: apply (simp add: add\_ac) yields
\( (b + c) + a \leadsto \cdots \leadsto a + (b + c) \)

AC Rules

Example for associative-commutative rules:

\textbf{Associative:} \( x \circ (y \circ z) = x \circ ( y \circ z) \)
\textbf{Commutative:} \( x \circ y = y \circ x \)

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

Example: \( (z \circ x) \circ (y \circ v) \)
We want: \( (z \circ x) \circ (y \circ v) = v \circ (x \circ (y \circ z)) \)
We get: \( (z \circ x) \circ (y \circ v) = v \circ (y \circ (x \circ z)) \)
We need: \textbf{AC rule} \( x \circ (y \circ z) = y \circ (x \circ z) \)

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 \( l_1 \rightarrow r_1 \) 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. \((1) + (3)\) \( x \rightarrow g z \) \( a = \frac{(1)}{f g t} \rightarrow b \)
2. \((3) + (2)\) \( z \rightarrow y \) \( b = \frac{(3)}{f g t} \rightarrow b \)

Completion

\( f x \rightarrow a \) \( g y \rightarrow b \) \( f (g z) \rightarrow b \)

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:
\( (1) + (3) \) \( x \rightarrow g z \) \( a = \frac{(1)}{f g t} \rightarrow b \)
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.

DEMO: WALDMEISTER
We have learned today ...

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