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

- Conditional rewriting
- Rewriting with assumptions
- Case splitting
- Congruence rules
- Permutative rewriting, AC rules

Slide 3

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)+(3) \[ x \rightarrow g \ z \]\[ a \rightarrow \text{rule (1)} \]b \rightarrow \text{rule (3)} \]b
(3)+(2) \[ z \rightarrow y \]\[ b \rightarrow \text{rule (3)} \]f g t \rightarrow \text{rule (2)} \]b

Slide 4
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:

\[
(1) \times (3) \ \{x \mapsto g \ z\} \ a \ \overset{(1)}{\rightarrow} \ f \ g \ t \ \overset{(3)}{\rightarrow} \ b
\]

shows that \( a = b \) (because \( a \overset{*}{\rightarrow} b \)), 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

DEMO: WALDMEISTER

THAT WAS TERM REWRITING
MORE ISAR

Last Time on Isar

- basic syntax
- proof and qed
- assume and show
- from and have
- the three modes of Isar

Backward and Forward

Backward reasoning: \begin{itemize}
  \item have \( A \land B \) proof
  \item proof picks an intro rule automatically
  \item conclusion of rule must unify with \( A \land B \)
\end{itemize}

Forward reasoning: \begin{itemize}
  \item assume \( A \land B \)
  \item from \( A \land B \) have \( \ldots \) proof
  \item now proof picks an elim rule automatically
  \item triggered by from
  \item first assumption of rule must unify with \( A \land B \)
\end{itemize}

General case: from \( A_1 \ldots A_n \) have \( R \) proof
\begin{itemize}
  \item first \( n \) assumptions of rule must unify with \( A_1 \ldots A_n \)
  \item conclusion of rule must unify with \( R \)
\end{itemize}

Fix and Obtain

\texttt{fix} \( v_1 \ldots v_n \)
\begin{itemize}
  \item Introduces new arbitrary but fixed variables
    \item \( \left( \sim \text{parameters}, \overline{A} \right) \)
\end{itemize}

\texttt{obtain} \( v_1 \ldots v_n \textbf{ where } \langle \text{prop} \rangle \langle \text{proof} \rangle \)
\begin{itemize}
  \item Introduces new variables together with property
\end{itemize}
Fancy Abbreviations

- this = the previous fact proved or assumed
- then = from this
- thus = then show
- hence = then have
- with $A_1 \ldots A_n$ = from $A_1 \ldots A_n$ this
- ?thesis = the last enclosing goal statement

Moreover and Ultimately

\[
\begin{align*}
\text{have } X_1: P_1, \ldots, & \quad \text{have } P_1, \ldots \\
\text{have } X_2: P_2, \ldots, & \quad \text{moreover have } P_2, \ldots \\
\vdots & \quad \vdots \\
\text{have } X_n: P_n, \ldots, & \quad \text{moreover have } P_n, \ldots \\
\text{from } X_1 \ldots X_n & \text{ show } \ldots \\
\text{ultimately show } \ldots & \\
\text{wastes lots of brain power} & \\
\text{on names } X_1 \ldots X_n &
\end{align*}
\]

General Case Distinctions

\[
\begin{align*}
\text{show } \text{ formula} \\
\text{proof -} & \\
\text{have } P_1 \lor P_2 \lor P_3 & \text{ < proof >} \\
\text{moreover} & \{ \text{ assume } P_1 \ldots \text{ have } ?\text{thesis < proof >} \} \\
\text{moreover} & \{ \text{ assume } P_2 \ldots \text{ have } ?\text{thesis < proof >} \} \\
\text{moreover} & \{ \text{ assume } P_3 \ldots \text{ have } ?\text{thesis < proof >} \} \\
\text{ultimately show } ?\text{thesis by blast} \\
\text{qed} & \\
\{ \ldots \} & \text{ is a proof block similar to } \text{ proof } \ldots \text{ qed} \\
\{ \text{ assume } P_1 \ldots \text{ have } P \text{ < proof >} \} & \\
\text{stands for } P_1 \implies P
\end{align*}
\]
Mixing proof styles
from ... have ... apply - make incoming facts assumptions
apply {...} ...
apply {...} done

Slide 17

Sets in Isabelle
Type 'a set: sets over type 'a
◆ {} ≤ {e_1, ..., e_n} ≤ {x. P x}
◆ e ∈ A, A ⊆ B
◆ A ∪ B, A ∩ B, A − B, −A
◆ ⋃ x ∈ A. B x, ⋂ x ∈ A. B x, ⋃ A, ⋂ A
◆ {[...]} insert ◦ α ⇒ α set ⇒ α set
◆ f A ≡ {y. ∃ x ∈ A. y = f x}
◆ ...

Slide 19

Proofs about Sets
Natural deduction proofs:
◆ equality: [A ⊆ B, B ⊆ A] ⇒ A = B
◆ subset: (⋀ x. x ∈ A ⇒ x ∈ B) ⇒ A ⊆ B
◆ ... (see Tutorial)

Slide 20

BUILDING UP SPECIFICATION TECHNIQUES: SETS

Slide 18
Bounded Quantifiers

- $\forall x \in A. P x \equiv \forall x. x \in A \rightarrow P x$
- $\exists x \in A. P x \equiv \exists x. x \in A \land P x$

- $\textbf{ball} \colon (\forall x. x \in A \rightarrow P x) \rightarrow \forall x \in A. P x$
- $\textbf{bspec} \colon (\forall x. P x ; x \in A] \rightarrow P x$

- $\textbf{bex} \colon [P x ; x \in A] \rightarrow \exists x \in A. P x$
- $\textbf{bexE} \colon [\exists x \in A. P x ; \forall x. [x \in A. P x] \rightarrow Q] \rightarrow Q$

\textbf{DEMO: SETS}