COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Simon Winwood, Toby Murray, June Andronick, Gerwin Klein

$$\longrightarrow \qquad \{\}$$

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

Last Time

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

Back to Confluence

Last time: confluence in general is undecidable.
But: confluence for terminating systems is decidable!
Problem: overlapping lhs of rules.

Definition:

Let $l_1 \longrightarrow r_1$ and $l_2 \longrightarrow r_2$ 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) $f\ x \longrightarrow a$ (2) $g\ y \longrightarrow b$ (3) $f\ (g\ z) \longrightarrow b$
Critical pairs:

(1)+(3)	$\{x \mapsto g\ z\}$	$a \xleftarrow{(1)} f\ g\ t \xrightarrow{(3)} b$
(3)+(2)	$\{z \mapsto y\}$	$b \xleftarrow{(3)} f\ g\ t \xrightarrow{(2)} b$

Completion

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

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:

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

shows that $a = b$ (because $a \overset{*}{\longleftrightarrow} b$), so we add $a \longrightarrow b$ as a rule

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

Slide 5

Orthogonal Rewriting Systems

Definitions:

A **rule** $l \longrightarrow r$ 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

Slide 7

DEMO: WALDMEISTER

Slide 6

THAT WAS TERM REWRITING

Slide 8

NICTA

MORE ISAR

Slide 9

Last Time on Isar

NICTA

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

Slide 10

Backward and Forward

NICTA

Backward reasoning: . . . **have** "$A \wedge B$" **proof**

- ➜ **proof** picks an **intro** rule automatically
- ➜ conclusion of rule must unify with $A \wedge B$

Forward reasoning: . . .

assume AB: "$A \wedge B$"
from AB **have** ". . ." **proof**

- ➜ now **proof** picks an **elim** rule automatically
- ➜ triggered by **from**
- ➜ first assumption of rule must unify with AB

General case: from $A_1 \ldots A_n$ **have** R **proof**

- ➜ first n assumptions of rule must unify with $A_1 \ldots A_n$
- ➜ conclusion of rule must unify with R

Slide 11

Fix and Obtain

NICTA

fix $v_1 \ldots v_n$

Introduces new arbitrary but fixed variables
($\sim$ parameters, $\bigwedge$)

obtain $v_1 \ldots v_n$ **where** <prop> <proof>

Introduces new variables together with property

Slide 12

DEMO

Slide 13

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

Slide 14

Moreover and Ultimately

have X_1: P_1 ...	**have** P_1 ...
have X_2: P_2 ...	**moreover have** P_2 ...
$\vdots$	$\vdots$
have X_n: P_n ...	**moreover have** P_n ...
from $X_1 \ldots X_n$ **show** ...	**ultimately show** ...

wastes lots of brain power
on names $X_1 \ldots X_n$

Slide 15

General Case Distinctions

show $formula$
proof -
  have $P_1 \vee P_2 \vee P_3$ <proof>
  moreover { **assume** P_1 ... **have** ?thesis <proof> }
  moreover { **assume** P_2 ... **have** ?thesis <proof> }
  moreover { **assume** P_3 ... **have** ?thesis <proof> }
  ultimately show ?thesis **by** blast
qed

{ ... } is a proof block similar to **proof** ... **qed**

{ **assume** P_1 ... **have** P <proof> }
stands for $P_1 \Longrightarrow P$

Slide 16

Mixing proof styles

from ...

have ...

  apply - make incoming facts assumptions

  apply (...)

  ⋮

  apply (...)

  done

Slide 17

BUILDING UP SPECIFICATION TECHNIQUES: SETS

Slide 18

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$, $\{x.\ P\ x\}$
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, $A - B$, $-A$
- $\bigcup x \in A.\ B\ x$, $\bigcap x \in A.\ B\ x$, $\bigcap\ A$, $\bigcup A$
- $\{i..j\}$
- insert :: $\alpha \Rightarrow \alpha$ set $\Rightarrow \alpha$ set
- $f\text{`}A \equiv \{y.\ \exists x \in A.\ y = f\ x\}$
- ...

Slide 19

Proofs about Sets

Natural deduction proofs:

- equalityI: $[\![A \subseteq B;\ B \subseteq A]\!] \Longrightarrow A = B$
- subsetI: $(\bigwedge x.\ x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$
- ... (see Tutorial)

Slide 20

Bounded Quantifiers

- $\forall x \in A.\ P\ x \equiv \forall x.\ x \in A \longrightarrow P\ x$
- $\exists x \in A.\ P\ x \equiv \exists x.\ x \in A \wedge P\ x$

- ballI: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \Longrightarrow \forall x \in A.\ P\ x$
- bspec: $[\![\forall x \in A.\ P\ x; x \in A]\!] \Longrightarrow P\ x$

- bexI: $[\![P\ x; x \in A]\!] \Longrightarrow \exists x \in A.\ P\ x$
- bexE: $[\![\exists x \in A.\ P\ x; \bigwedge x.\ [\![x \in A; P\ x]\!] \Longrightarrow Q]\!] \Longrightarrow Q$

Slide 21

DEMO: SETS

Slide 22