COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein
Formal Methods
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

- Isar, structured proofs
- Term rewriting, rule applications
- Conditional term rewriting
- Congruence rules
ADVANCED TERM REWRITING
Problem: \( x + y \longrightarrow y + x \) 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 `add_ac` sort any sum (+)
- lemmas `times_ac` sort any product (*)

Example: apply (simp add: add_ac) yields

\[
(b + c) + a \sim \cdots \sim a + (b + c)
\]
Example for associative-commutative rules:

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

**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: **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
DEMO
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\}  
(3)+(2) \{z \rightarrow y\}
(1) \( f \ x \to a \)  \( (2) \ g \ y \to b \)  \( (3) \ f \ (g \ z) \to b \)

is not confluent

But it can be made confluent by adding rules!

**How:** join all critical pairs

Example:

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

shows that \( a = b \) (because \( a \xleftarrow{*} b \)), so we add \( a \to b \) as a rule

This is the main idea of the Knuth-Bendix completion algorithm.
DEMO: WALDMEISTER
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
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 reasoning: . . . have ”$A \land B$” proof
  $\rightarrow$ proof picks an intro rule automatically
  $\rightarrow$ conclusion of rule must unify with $A \land B$

Forward reasoning: . . .
  assume AB: ”$A \land B$”
  from AB have ”. . .” proof
  $\rightarrow$ now proof picks an elim rule automatically
  $\rightarrow$ triggered by from
  $\rightarrow$ first assumption of rule must unify with AB

General case: from $A_1 \ldots A_n$ have $R$ proof
  $\rightarrow$ first $n$ assumptions of rule must unify with $A_1 \ldots A_n$
  $\rightarrow$ conclusion of rule must unify with $R$
\textbf{fix \( v_1 \ldots v_n \)}

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

\textbf{obtain \( v_1 \ldots v_n \) where <prop> <proof>}

Introduces new variables together with property
DEMO
**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

have \( X_1 : P_1 \ldots \)
have \( X_2 : P_2 \ldots \)
: 
have \( X_n : P_n \ldots \)
from \( X_1 \ldots X_n \) show \ldots

have \( P_1 \ldots \)
moreover have \( P_2 \ldots \)
: 
moreover have \( P_n \ldots \)
ultimately show \ldots

wastes lots of brain power
on names \( X_1 \ldots X_n \)
show \textit{formula}

proof -

\begin{itemize}
\item have $P_1 \lor P_2 \lor P_3$ \textit{<proof>}
\item moreover \{ assume $P_1$ \ldots have \textit{?thesis} \textit{<proof>} \}
\item moreover \{ assume $P_2$ \ldots have \textit{?thesis} \textit{<proof>} \}
\item moreover \{ assume $P_3$ \ldots have \textit{?thesis} \textit{<proof>} \}
\end{itemize}

ultimately show \textit{?thesis} by blast

\textit{qed}

\{ \ldots \} is a proof block similar to \textit{proof} \ldots \textit{qed}

\begin{itemize}
\item \{ assume $P_1$ \ldots have $P$ \textit{<proof>} \}
\end{itemize}

stands for $P_1 \implies P$
MIXING PROOF STYLES

from . . .

have . . .

apply - make incoming facts assumptions

apply (. . .)

:

apply (. . .)

done