COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray
Rough timeline

**Intro & motivation, getting started**

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

**Proof & Specification Techniques**
- Isar [11,12]
- 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]

---

a1 out; b1 due; c2 out; d2 due; e session break; f a3 out; g a3 due
Last Time

- Equations and Term Rewriting
- Confluence and Termination of reduction systems
- Term Rewriting in Isabelle
Applying a Rewrite Rule

$\rightarrow l \rightarrow r$ applicable to term $t[s]$ if there is substitution $\sigma$ such that $\sigma l = s$

$\rightarrow$ Result: $t[\sigma r]$

$\rightarrow$ Equationally: $t[s] = t[\sigma r]$

Example:

Rule: $0 + n \rightarrow n$

Term: $a + (0 + (b + c))$

Substitution: $\sigma = \{n \mapsto b + c\}$

Result: $a + (b + c)$
Rewrite rules can be conditional:

$$[P_1 \ldots P_n] \implies l = r$$

is applicable to term $t[s]$ with $\sigma$ if

$\implies \sigma l = s$ and

$\implies \sigma P_1, \ldots, \sigma P_n$ are provable by rewriting.
Rewriting with Assumptions

Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

```plaintext
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 (recursive) for maximal simplification power:

\[ \neg A \mapsto A = False \]
\[ A \rightarrow B \mapsto A \implies B \]
\[ A \land B \mapsto A, B \]
\[ \forall x. A x \mapsto A \ ? x \]
\[ A \mapsto A = True \]

Example:

\[ (p \rightarrow q \land \neg r) \land s \]

\[ \mapsto \]

\[ p \implies q = True \quad p \implies r = False \quad s = True \]
DEMO
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**

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

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

Similar for any data type \( t \): \textbf{t.split}
Congruence Rules

congruence rules are about using context

Example: in $P \rightarrow Q$ we could use $P$ to simplify terms in $Q$

For $\implies$ hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: $[P = P'; P' \implies Q = Q'] \implies (P \rightarrow Q) = (P' \rightarrow Q')$

Read: to simplify $P \rightarrow Q$

- first simplify $P$ to $P'$
- then simplify $Q$ to $Q'$ using $P'$ as assumption
- the result is $P' \rightarrow Q'$
More Congruence

Sometimes useful, but not used automatically (slowdown):

**conj_cong**: \([P = P'; P' \implies Q = Q']\) \implies (P \land Q) = (P' \land Q')

Context for if-then-else:

**if_cong**: \([b = c; c \implies x = u; \neg c \implies y = v]\) \implies (if b then x else y) = (if c then u else v)

Prevent rewriting inside then-else (default):

**if_weak_cong**: \(b = c \implies (if b then x else y) = (if c then x else y)\)

→ declare own congruence rules with [cong] attribute
→ delete with [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:

- lemmas \texttt{add_ac} sort any sum (+)
- lemmas \texttt{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:** \((x \odot y) \odot z = x \odot (y \odot z)\)

**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: \((z \odot x) \odot (y \odot v) = v \odot (x \odot (y \odot z))\)

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
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:

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

\[
(3) + (2) \quad \{z \mapsto y\} \quad b \xrightarrow{(3)} f \ g \ t \quad \xrightarrow{(2)} b
\]
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:

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

shows that \(a = b\) (because \(a \xrightarrow{*} b\)), so we add \(a \rightarrow b\) 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
We have learned today ...

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