**COMP 4161**
NICTA Advanced Course

**Advanced Topics in Software Verification**

Gerwin Klein, June Andronick, Toby Murray

## Content

| | Rough timeline |
|---|---|
| ➜ Intro & motivation, getting started | [1] |
| ➜ Foundations & Principles | |
| • Lambda Calculus, natural deduction | [2,3,4[a]] |
| • Higher Order Logic | [5,6[b],7] |
| • Term rewriting | [8,9,10[c]] |
| ➜ Proof & Specification Techniques | |
| • Isar | [11,12[d]] |
| • Inductively defined sets, rule induction | [13[e],15] |
| • Datatypes, recursion, induction | [16,17[f],18,19] |
| • Calculational reasoning, mathematics style proofs | [20] |
| • Hoare logic, proofs about programs | [21[g],22,23] |

---

[a] a1 out; [b] a1 due; [c] a2 out; [d] a2 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

- ➜ $l \longrightarrow r$ **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:** $0 + n \longrightarrow n$

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

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

**Result:** $a + (b + c)$

## Conditional Term Rewriting

Rewrite rules can be conditional:

$$[\![P_1 \dots P_n]\!] \Longrightarrow l = r$$

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

- ➜ $\sigma\ l = s$ and
- ➜ $\sigma\ P_1, \dots, \sigma\ P_n$ are provable by rewriting.

## Rewriting with Assumptions

Last time: Isabelle uses assumptions in rewriting.

**Can lead to non-termination.**

**Example:**

**lemma** ”$f\ x = g\ x \wedge g\ x = f\ x \Longrightarrow f\ x = 2$¨

| | |
|---|---|
| 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:

$$\begin{array}{rcl} \neg A & \mapsto & A = False \\ A \longrightarrow B & \mapsto & A \Longrightarrow B \\ A \wedge B & \mapsto & A,\ B \\ \forall x.\ A\ x & \mapsto & A\ ?x \\ A & \mapsto & A = True \end{array}$$

**Example:**

$$(p \longrightarrow q \wedge \neg r) \wedge s$$

$$\mapsto$$

$$p \Longrightarrow q = True \qquad p \Longrightarrow r = False \qquad s = True$$

## DEMO

## Case splitting with simp

$$\begin{gathered} P\ (\text{if } A \text{ then } s \text{ else } t) \\ = \\ (A \longrightarrow P\ s) \wedge (\neg A \longrightarrow P\ t) \end{gathered}$$

**Automatic**

$$\begin{gathered} P\ (\text{case } e \text{ of } 0\ \Rightarrow\ a \mid \text{Suc } n\ \Rightarrow\ b) \\ = \\ (e = 0 \longrightarrow P\ a) \wedge (\forall n.\ e = \text{Suc } n \longrightarrow P\ b) \end{gathered}$$

**Manually: apply** (simp split: nat.split)

Similar for any data type t: **t.split**

## Congruence Rules

**congruence rules are about using context**

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

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

For other operators expressed with conditional rewriting.

**Example**: $[\![P = P'; P' \Longrightarrow Q = Q']\!] \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

**Read**: to simplify $P \longrightarrow Q$

- ➜ first simplify $P$ to $P'$
- ➜ then simplify $Q$ to $Q'$ using $P'$ as assumption
- ➜ the result is $P' \longrightarrow Q'$

## More Congruence

Sometimes useful, but not used automatically (slowdown):
**conj_cong**: $[\![P = P'; P' \Longrightarrow Q = Q']\!] \Longrightarrow (P \wedge Q) = (P' \wedge Q')$

Context for if-then-else:
**if_cong**: $[\![b = c; c \Longrightarrow x = u; \neg c \Longrightarrow y = v]\!] \Longrightarrow$
$(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$

Prevent rewriting inside then-else (default):
**if_weak_cong**: $b = c \Longrightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)$

- ➜ declare own congruence rules with **[cong]** attribute
- ➜ delete with **[cong del]**

## Ordered rewriting

**Problem:** $x + y \longrightarrow y + x$ does not terminate

**Solution:** use permutative rules only if term becomes lexicographically smaller.

**Example:** $b + a \leadsto a + b$ but not $a + b \leadsto b + a$.

For types nat, int etc:

- lemmas **add_ac** sort any sum ($+$)
- lemmas **times_ac** sort any product ($*$)

**Example:** **apply** (simp add: add_ac) yields
$(b + c) + a \leadsto \cdots \leadsto a + (b + c)$

## AC Rules

**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

---

## 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.

# DEMO: WALDMEISTER

## 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

## We have learned today ...

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