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

- Isar, structured proofs
- shows, assumes
- the three modes of Isar

BACK TO TERM REWRITING ...
Applying a Rewrite Rule

→ l → r applicable to term t[s]
   if there is substitution σ such that σ l = s
→ Result: t[σ r]
→ Equationally: t(s) = t(σ r)

Example:
Rule: 0 + n → n
Term: a + (0 + (b + c))
Substitution: σ = {n ↦ b + c}
Result: a + (b + c)

Conditional Term Rewriting

Rewrite rules can be conditional:
[Ps . . . Pn] → l = r
is applicable to term t[s] with σ if
→ σ l = s and
→ σ P1, . . . , σ Pn 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 ∧ 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

Preprocessing (recursive) for maximal simplification power:

¬A =⇒ A = False
A → B =⇒ A =⇒ B
A ∧ B =⇒ A, B
∀x. A x =⇒ A ?x
A =⇒ A = True

Example:
(p → q ∧ ¬r) ∧ s

p =⇒ q = True  r = False  s = True
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**

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

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

Similar for any data type \(t: \text{t.split}\)

---

**Congruence Rules**

Congruence rules are about using context

**Example:** in \(P \rightarrow Q\) we could use \(P\) to simplify terms in \(Q\)

For \(\rightarrow\) hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

**Example:** \([P = P'; P' \Rightarrow Q = Q'] \Rightarrow (P \rightarrow Q) = (P' \rightarrow Q')\)

**Read:** to simplify \(P \rightarrow Q\)

\[ \rightarrow \]

1. first simplify \(P\) to \(P'\)
2. then simplify \(Q\) to \(Q'\) using \(P'\) as assumption
3. the result is \(P' \rightarrow Q'\)

---

**More Congruence**

Sometimes useful, but not used automatically (slowdown):

**\text{conjCong}**:
\([P = P'; P' \Rightarrow Q = Q'] \Rightarrow (P \land Q) = (P' \land Q')\)

Context for if-then-else:

**\text{ifCong}**:
\([b = c; e \Rightarrow x = u; c \Rightarrow y = v] \Rightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)\)

Prevent rewriting inside then-else (default):

**\text{ifWeakCong}**:
\([b = c \Rightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)\)

\rightarrow declare own congruence rules with \([\text{cong}]\) attribute

\rightarrow delete with \([\text{cong del}]\)
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 $\text{add_ac}$ sort any sum (+)
- lemmas $\text{times_ac}$ sort any product (*)

Example: apply (simp add: $\text{add_ac}$) yields $(b + c) + a \sim \cdots \sim a + (b + c)$

AC Rules

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.

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:
$(1)+(3)\ \{x \mapsto g\ z\}$  $a\ \frac{(1)\ a}{(1)\ b}$
$(3)+(2)\ \{z \mapsto y\}$  $b\ \frac{(3)\ b}{(2)\ b}$
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:

\((1)+(3)\) \( \{x \mapsto g \ z\} \)

shows that \(a = b\) (because \(a \xleftarrow{1} b\)), 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

We have learned today ...

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