Course 2D1453, 2006-07

Advanced Formal Methods

Lecture 5: Isabelle – Proofs and Rewriting

Mads Dam
KTH/CSC

Some slides from Paulson

Isabelle's Metalogic

Basic constructs:

- t = s
  Equations on terms
- $A_1 \Rightarrow A_2$
  Implication
  Example: $x = y \Rightarrow$ append x xs = append y xs
  If A_1 is valid then so is A_2

$\bigwedge x.\ A$
  Universal quantification
  A[t/x] is valid for all t (of appropriate type)

These are meta-connectives, not object-logic connectives

Isabelle Proof Goals

Proof goals, or judgments:

- The basic shape of proof goal handled by Isabelle
- Local proof state, subgoal

General shape: $\bigwedge x_1,\ldots,x_m.\ [\![A_1 ; \ldots ; A_n]\!] \Rightarrow A$

- $x_1,\ldots,x_m$: Local variables
- $A_1,\ldots,A_n$: Local assumptions
- A: local proof goal

Meaning: For all terms $t_1,\ldots,t_m$, if all $A_i[t_1/x_1,\ldots,t_m/x_m]$ are provable then so is $A[t_1/x_1,\ldots,t_m/x_m]$

Global Proof State

An Isabelle proof state consists of number of unproven judgments

1. $\bigwedge x_{1,1},\ldots,x_{m,1}.\ [\![A_{1,1} ; \ldots ; A_{n,1}]\!] \Rightarrow A_1$

....

k. $\bigwedge x_{1,k},\ldots,x_{m,k}.\ [\![A_{1,k} ; \ldots ; A_{n,k}]\!] \Rightarrow A_k$

If k = 0 proof is complete

Judgment #1 is the one currently being worked on

Commands to list subgoals, toggle between subgoals, to apply rules to numbered subgoals, etc.

Goal-Driven Proof - Intuition

Proof goal:

* $\bigwedge x_1,\ldots,x_m.\ [\![A_1 ; \ldots ; A_n]\!] \Rightarrow A$

Find some "given fact" B, under assumptions $B_1,\ldots,B_k$ such that A "is" B

Replace subgoal * by subgoals

$\bigwedge x_1,\ldots,x_m.\ [\![A_1 ; \ldots ; A_n]\!] \Rightarrow$ B_1

...

$\bigwedge x_1,\ldots,x_m.\ [\![A_1 ; \ldots ; A_n]\!] \Rightarrow$ B_k

But, "is" is really "is an instance of" so story must be refined

Unification

Substitution:
  Mapping σ from variables to terms

[t/x]: Substitution mapping x to t, otherwise the identity

tσ: Capture-avoiding substitution σ applied to t

Unification:
  Try to make terms t and s equal
  Unifier: Substitution σ on terms s, t such that $s\sigma = t\sigma$
  Unification problem: Given t, s, is there a unifier on s, t

Higher-Order Unification

In Isabelle:

- Terms are terms in Isabelle = extended $\lambda_{\rightarrow}$ Terms
- Equality on terms are modulo α, β, η
- Variables to be unified are *schematic*
- Schematic variables can have function type
  - (= higher order)

Examples:

$?X \wedge ?Y =_{\alpha\beta\eta} x \wedge x$	under [x/?X,x/?Y]
$?P\ x =_{\alpha\beta\eta} x \wedge x$	under [λx.x $\wedge$ x/?P]
$P\ (?f\ x) =_{\alpha\beta\eta}\ ?Y\ x$	under [λx.x/?f,P/Y]

First Order Unification

Decidable

Most general unifiers (mgu's) exist:

- σ is mgu for t and s if
  - σ unifies t and s
  - Whenever σ' unifies t and s then tσ, tσ', and sσ, sσ' are both unifiable

Exercise 1: Show that [h(?Y)/?X,g(h(?Y))/?Z] is mgu for f(?X,g(?X)) and f(h(?Y),?Z).

Applications in e.g. logic programming

Higher Order Unification

HO unification modulo α, β is semi-decidable

HO unification modulo α,β,η is undecidable

Higher order pattern:

- Term t in β normal form (*value* in slides for lecture 3)
- Schematic variables only in head position
  - $?f\ t_1 \dots t_n$
- Each t_i η-convertible to n distinct bound variables

Unification on HO patterns is decidable

Exercises

Exercise 2: Determine whether each pair of terms is unifiable or not. If it is, exhibit a unifier. If it is not, show why.

1. $f(x_1, ?x_2, ?x_2)$ and $f(?y_1, ?y_2, k)$
2. $f(x_1, ?x_2, ?x_2)$ and $f(y_1, g\ ?x_2, k)$
3. f (?p x y (h z)) and ?q (g(x,y),h(?r))
4. $?p\ (g\ x_1)\ (h\ x_2)$ and $?q\ (g\ y_2)\ (h\ y_1)$
5. ?p (g ?q, h z) and f(h ?r, h ?r)

Term Rewriting

Use equations t = s as rewrite rules from left to right

Example: Use equations:

1. 0 + n = n
2. (suc m) + n = suc(m + n)
3. (suc m $\leq$ suc n) = (m $\leq$ n)
4. (0 $\leq$ m) = true

Then:

0 + suc 0 $\leq$ (suc 0) + x	(by (1))
= suc 0 $\leq$ (suc 0) + x	(by (2))
= suc 0 $\leq$ suc (0 + x)	(by (3))
= 0 $\leq$ 0 + x	(by (4))
= true	

More Formally

Rewrite rule l = r is *applicable* to term t[s/x] if:

- There is a substitution σ such that $l\sigma =_{\alpha\beta\eta} s$
- σ unifies l and s

Result of rewrite is t[sσ/x]

Note: t[s/x] = t[sσ/x]

Example:
Equation: 0 + n = n
Term: a + (0 + (b + c))
Substitution: [b+c/n]
Result: a + (b + c)

Conditional Rewriting

Assume conditional rewrite rule

RId: $A_1 \Rightarrow ... \Rightarrow A_n \Rightarrow l = r$

Rule RId is *applicable* to term t[s/x] if:

- There is a substitution σ such that $l\sigma =_{\alpha\beta\eta} s$
- σ unifies l and s
- $A_1\sigma, ..., A_n\sigma$ are provable

Again result of rewrite is t[sσ/x]

Basic Simplification

Goal: $[\![A_1; ... ; A_m]\!] \Rightarrow B$

Apply(simp add: $eq_1, ... , eq_n$)

Simplify B using

- Lemmas with attribute *simp*
- Rules from **primrec** and **datatype** declarations
- Additional lemmas $eq_1,...,eq_n$
- Assumptions $A_1, ... , A_m$

Variation:

- (simp ... del: ...) removes lemmas from simplification set
- add, del are optional

Termination

Isabelle uses simp-rules (almost) blindly from left to right

Termination is *the* big issue

Example: f(x) = g(x), g(x) = f(x)

Rewrite rule

$[\![A_1; ... ; A_n]\!] \Rightarrow l = r$

suitable for inclusion in simplification set only if rewrite from l to r *reduces* overall complexity of the global proof state

So: l must be "bigger" than r and each A_i

$n < m = true \Rightarrow (n < suc\ m) = true$ (may be good)

$(suc\ n < m) = true \Rightarrow n < m = true$ (not good)

Case Splitting

$P(\text{if } A \text{ then } s \text{ else } t) = (A \rightarrow P(s)) \wedge (\neg A \rightarrow P(t))$

Included in simp by default

$P(\text{case } t \text{ of } 0 \Rightarrow s_1 \mid Suc\ n \Rightarrow s_2)$

$= (t = 0 \rightarrow P(s_1)) \wedge (\forall n.\ t = Suc\ n \rightarrow P(s_2))$

Not included – use (simp split: nat.split)

Similar for other datatypes T: T.split

Ordered Rewriting

Problem: ?x + ?y = ?y + ?x does not terminate

Isabelle: Use permutative rewrite rules only when term becomes lexicographically smaller

Example: ?b + ?a ⇝ ?a + ?b but not ?a + ?b ⇝ ?b + ?a

For types nat, int, etc.

- Lemmas add_ac sort any sum
- Lemmas times_ac sort any product

Example: (simp add:add_ac) yields

(b + c) + a ⇝ a + (b + c)

Preprocessing

Simplification rules are preprocessed recursively:

$\neg A \mapsto A = False$

$A \rightarrow B \mapsto A \Rightarrow B$

$A \wedge B \Rightarrow A, B$

$\forall x.\ A(x) \mapsto A(?x)$

$A \mapsto A = True$

Example:

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

$\mapsto p = True \Rightarrow q = True, p = True \Rightarrow r = False, s = True$