Topics in Automated Deduction (CS 576)

Elsa L. Gunter

2112 Siebel Center

egunter@cs.uiuc.edu

http://www.cs.uiuc.edu/class/sp06/cs576/

Variables

Three kinds of variables in Isabelle:

- bound: $\forall \mathtt{x}.\ \mathtt{x} = \mathtt{x}$
- free: $\mathtt{x} = \mathtt{x}$
- *schematic*: $?\mathtt{x} = ?\mathtt{x}$

  ("unknown", a.k.a. *meta-variables*)

Can be mixed in term or formula: $\forall b.\ \exists y.\ f\ ?a\ y = b$

Variables

- Logically: free = bound at meta-level
- Operationally:
  - free variabes are fixed
  - schematic variables are instantiated by substitutions

From `x` to `?x`

State lemmas with free variables:

```
lemma app_Nil2 [simp]:  "xs @ [ ] = xs"

 ⋮
 done
```

After the proof: Isabelle changes `xs` to `?xs` (internally):

```
?xs @ [ ] = ?xs
```

Now usable with arbitrary values for `?xs`

Example: rewriting

```
rev(a @ [ ]) = rev a
```

using `app_Nil2` with $\sigma = \{?\mathtt{xs} \mapsto \mathtt{a}\}$

Basic Simplification

Goal: 1. $[\![\mathtt{P}_1;\ldots;\mathtt{P}_m]\!] \Longrightarrow \mathtt{C}$

`apply (simp add:` $eq\_thm_1 \ \ldots \ eq\_thm_n$`)`

Simplify (mostly rewrite) $P_1;\ldots;P_m$ and C using

- lemmas with attribute `simp`
- rules from `primrec` and `datatype`
- additional lemmas $eq\_thm_1 \ \ldots \ eq\_thm_n$
- assumptions $P_1;\ldots;P_m$

Variations:

- `(simp ... del: ...)` removes `simp`-lemmas
- `add` and `del` are optional

`auto` versus `simp`

- `auto` acts on all subgoals
- `simp` acts only on subgoal 1
- `auto` applies `simp` and more
  - `simp` concentrates on rewriting
  - `auto` combines rewriting with resolution

Termination

Simplification may not terminate.

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

Example: $f(x) = g(x)$, $g(x) = f(x)$ will not terminate.

$$[\![P_1, \ldots P_n]\!] \Longrightarrow l = r$$

is only suitable as a `simp`-rule only if l is "bigger" than r and each P_i.

$$\begin{array}{rl} \texttt{(n < m) = (Suc n < Suc m)} & \textbf{NO} \\ \texttt{(n < m)} \Longrightarrow \texttt{(n < Suc m) = True} & \text{YES} \\ \texttt{Suc n < m} \Longrightarrow \texttt{(n < m) = True} & \textbf{NO} \end{array}$$

Assumptions and Simplification

Simplification of $[\![A_1, \ldots, A_n]\!] \Longrightarrow B$:

- Simplify A_1 to A_1'
- Simplify $[\![A_2, \ldots, A_n]\!] \Longrightarrow B$ using A_1'

Ignoring Assumptions

Sometimes need to ignore assumptions; can introduce non-termination.

How to exclude assumptions from `simp`:

```
apply (simp (no_asm_simp)...)
```

Simplify only the conclusion, but use assumptions

```
apply (simp (no_asm_use)...)
```

Simplify all, but do not use assumptions

```
apply (simp (no_asm)...)
```

Ignore assumptions completely

Rewriting with Definitions (`constdefs`)

Definitions do not have the `simp` attirbute.

They must be used explicitly:

```
apply (simp add: f_def ...)
```

Alternately, to just expand the definition:

```
apply (unfold f_def ...)
```

Ordered Rewriting

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

Solution: Permutative `simp`-rules are used only if the term becomes lexicographically smaller.

Example: $\mathtt{b} + \mathtt{a} \leadsto \mathtt{a} + \mathtt{b}$ but not $\mathtt{a} + \mathtt{b} \leadsto \mathtt{b} + \mathtt{a}$.

For types `nat`, `int`, etc., commutative, associative and distributive laws built in.

Example: `apply simp` yields:

$$((B + A) + ((2 :: nat) * C)) + (A + B) \quad \leadsto$$

$$\ldots \quad \leadsto \quad 2 * A + (2 * B + 2 * C)$$

Preprocessing

simp-rules are preprocessed (recursively) for maximal simplification power:

$$\begin{aligned} \neg A &\mapsto A = \texttt{False} \\ A \longrightarrow B &\mapsto A \Longrightarrow B \\ A \wedge B &\mapsto A, B \\ \forall x.A(x) &\mapsto A(?x) \\ A &\mapsto A = \texttt{True} \end{aligned}$$

Example:

$$(\texttt{p} \longrightarrow \texttt{q} \wedge \neg \texttt{r}) \wedge \texttt{s} \mapsto \texttt{p} \Longrightarrow \texttt{q} = \texttt{True}, \texttt{r} = \texttt{False}, \texttt{s} = \texttt{True}$$

Case Splitting with `simp`

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

Automatic by `apply (simp)`

Generalizing to `case`:

$$\texttt{P(case exp of 0} \Rightarrow \texttt{a|Suc n} \Rightarrow \texttt{b)} =$$
$$((\texttt{e} = 0 \longrightarrow \texttt{P(a)}) \wedge (\forall \texttt{n. exp} = \texttt{Suc n} \longrightarrow \texttt{P(b)}))$$

Needs more direction: `apply (simp split: nat.split)`

Similar for any datatype `t`: `t.split`

Demo: Simplification through Rewriting

Basic Induction Heuristics

- Theorems about recursive functions are proved by induction
- If f defined by induction on ith argument, proof is by induction of ith argument of f

Example: Tail Recursive Reverse

```
consts itrev ::  'a list ⇒ 'a list ⇒ 'a list

primrec
 itrev [ ]     ys = ys
 itrev (x#xs)  ys = itrev xs (x#ys)

lemma itrev xs [] = rev xs
```

This direction is easier to prove/use

lhs "more complex" than rhs

Demo: first attempt at itrev = rev

Generalization (first kind)

Replace constant arguments (`[ ]`) by variables:

```
lemma itrev xs ys = rev xs @ ys
```

Demo: second attempt at itrev = rev

Generalization (second kind)

Quantify all free variables by $\forall$,

except the induction variable

```
lemma ∀ ys.  itrev xs ys = rev xs @ ys
```

Proof Basics

- Isabelle uses *Natural Deduction* proofs
  - Uses *sequent* encoding
- Rule notation:

Rule	Sequent Encoding
$\dfrac{A_1 \ldots A_n}{A}$	$[\![A_1, \ldots, A_n]\!] \Longrightarrow A$
$\dfrac{A_1 \ldots \overset{\overset{B}{\vdots}}{\overline{A_i}} \ldots A_n}{A}$	$[\![A_1, \ldots, B \Longrightarrow A_i, \ldots, A_n]\!] \Longrightarrow A$

Natural Deduction

For each logical operator $\oplus$, have two kinds of rules:

Introduction: How can I prove $A \oplus B$?

$$\frac{?}{A \oplus B}$$

Elimination: What can I prove using $A \oplus B$?

$$\frac{\ldots A \oplus B \ldots}{?}$$

Operational Reading

$$\frac{A_1 \ldots A_n}{A}$$

Introduction rule:

To prove A it suffices to prove $A_1 \ldots A_n$.

Elimination rule:

If we know A_1 and we want to prove A

it suffices to prove $A_2 \ldots A_n$

Natural Deduction for Propositional Logic

$$\frac{A \quad B}{A \wedge B}\ \texttt{conjI}$$

$$\frac{A \wedge B \quad [\![A; B]\!] \Longrightarrow C}{C}\ \texttt{conjE}$$

$$\frac{A}{A \vee B}\ \frac{B}{A \vee B}\ \texttt{disjI1/2}$$

$$\frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{c}\ \texttt{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B}\ \texttt{impI}$$

$$\frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C}\ \texttt{impE}$$

Natural Deduction for Propositional Logic

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B}\ \texttt{iffI} \qquad \frac{A = B \quad A}{B}\ \texttt{iffD1}$$

$$\frac{A = B \quad B}{A}\ \texttt{iffD2}$$

$$\frac{A \Longrightarrow \texttt{False}}{\neg A}\ \texttt{notI} \qquad \frac{\neg A \quad A}{B}\ \texttt{notE}$$