CS576 Topics in Automated Deduction

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs576

Slides based in part on slides by Tobias Nipkow

February 11, 2015

Rewriting: More Formally

substitution = mapping of variables to terms

- $l = r$ is *applicable* to term $t[s]$ if there is a substitution σ such that $\sigma(l) = s$
  - s is an instance of l
- Result: $t[\sigma(r)]$
- Also have theorem: $t[s] = t[\sigma(r)]$

Example

- Equation: $0 + n = n$
- Term: $a + (0 + (b + c))$
- Substitution: $\sigma = \{n \mapsto b + c\}$
- Result: $a + (b + c)$
- Theorem: $a + (0 + (b + c)) = a + (b + c)$

Conditional Rewriting

Rewrite rules can be conditional:

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

is *applicable* to term $t[s]$ with substitution σ if:

- $\sigma(l) = s$ and
- $\sigma(P_1), \ldots, \sigma(P_n)$ are provable (possibly again by rewriting)

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. $[\![P_1;\ldots;P_m]\!] \Longrightarrow C$

`proof (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`, `fun` 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.

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

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`:

`proof (simp (no_asm_simp)...)`

Simplify only the conclusion, but use assumptions

`proof (simp (no_asm_use)...)`

Simplify all, but do not use assumptions

`proof (simp (no_asm)...)`

Ignore assumptions completely

Rewriting with Definitions (definition)

Definitions do not have the simp attirbute.

They must be used explicitly:

proof (simp add: 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: `proof simp` yields:

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

Preprocessing

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

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

Example:

$$
(\texttt{p} \longrightarrow \texttt{q} \wedge \neg \texttt{r}) \wedge \texttt{s} \quad \mapsto \quad \begin{array}{l} p \Longrightarrow q = \mathit{True}, \\ p \Longrightarrow r = \texttt{False}, \\ s = \mathit{True} \end{array}
$$

Demo: Simplification through Rewriting

General Isar Proof Format

```
proof (method)
    fix x
    assume A0: formula_0
    from A0
    have A1: formula_1
    by (method)
    from A0 and A1
      ...
      show formula_n
      proof (method)
        ...
      qed
qed
```

Proves $formula_0 \Longrightarrow formula_n$

Basic Isar Syntax

$$
\begin{array}{lcll}
\textit{proof} & = & \texttt{proof}\ [\textit{method}]\ \textit{statement}^{*}\ \texttt{qed} & \\
 & \mid & \texttt{by}\ \textit{method} & \\
 & & & \\
\textit{method} & = & (\texttt{simp}\ \ldots)\,|\,(\texttt{auto}\ \ldots)\,|\,(\texttt{blast}\ \ldots)\,|\,(\texttt{rule}\ \ldots)\,|\ \ldots & \\
 & & & \\
\textit{statement} & = & \texttt{fix}\ \textit{variable}^{+} & (\bigwedge) \\
 & \mid & \texttt{assume}\ \textit{proposition} & (\Longrightarrow) \\
 & \mid & [\texttt{from}\ \textit{name}^{+}]\ \textit{objective}\ \textit{proof} & \\
 & \mid & \texttt{next} & \text{(starts next subgoal)} \\
 & & & \\
\textit{objective} & = & \texttt{show}\ \textit{proposition} & \text{(next proof step)} \\
 & \mid & \texttt{have}\ \textit{proposition} & \text{(local claim)} \\
 & \mid & \texttt{obtain}\ \textit{variable}^{+}\ \texttt{where}\ \textit{proposition}^{+} & \\
 & & & \\
\textit{proposition} & = & [\textit{name}:]\ \textit{formula} &
\end{array}
$$

Proof Basics

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

Rule	Sequent Encoding
$\dfrac{A_1 \ldots A_n}{A}$	$[\![A_1, \ldots, A_n]\!] \Longrightarrow A$
$\dfrac{A_1 \ldots \quad \begin{matrix} B \\ \vdots \\ \hline A_i \end{matrix} \quad \ldots A_n}{A}$	$[\![A_1, \ldots, B \Longrightarrow A_i, \ldots, A_n]\!] \Longrightarrow A$

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

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

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

$$\frac{\mathtt{A}}{\mathtt{A} \vee \mathtt{B}} \qquad \frac{\mathtt{B}}{\mathtt{A} \vee \mathtt{B}}\ \mathtt{disjI1/2}$$

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

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

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

Natural Deduction for Propositional Logic

$$\frac{\mathtt{A} \Longrightarrow \mathtt{B} \quad \mathtt{B} \Longrightarrow \mathtt{A}}{\mathtt{A} = \mathtt{B}}\ \mathtt{iffI}$$

$$\frac{\mathtt{A} = \mathtt{B} \quad \mathtt{A}}{\mathtt{B}}\ \mathtt{iffD1}$$

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

$$\frac{\mathtt{A} \Longrightarrow \mathtt{False}}{\neg\mathtt{A}}\ \mathtt{notI}$$

$$\frac{\neg\mathtt{A} \quad \mathtt{A}}{\mathtt{B}}\ \mathtt{notE}$$