CS576 Topics in Automated Deduction

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

Slides based in part on slides by Tobias Nipkow

February 10, 2026

Term Rewriting

Term rewriting means . . .

Terminology: equation becomes *rewrite rule*

Using a set of equations $l = r$ from left to right

As long as possible (possibly forever!)

Example

Equations:

$$\begin{array}{rcll} 0 + n & = & n & (1)\\ (\mathtt{Suc}\ m) + n & = & \mathtt{Suc}(m + n) & (2)\\ (0 \leq m) & = & \mathtt{True} & (3)\\ (\mathtt{Suc}\ m \leq \mathtt{Suc}\ n) & = & (m \leq n) & (4) \end{array}$$

Rewriting:

$$\begin{array}{rcll} 0 + \mathtt{Suc}\ 0 & \leq & \mathtt{Suc}\ 0 + x & \underline{\underline{(1)}}\\ \mathtt{Suc}\ 0 & \leq & \mathtt{Suc}\ 0 + x & \underline{\underline{(2)}}\\ \mathtt{Suc}\ 0 & \leq & \mathtt{Suc}(0 + x) & \underline{\underline{(4)}}\\ 0 & \leq & 0 + x & \underline{\underline{(3)}}\\ \multicolumn{4}{c}{\mathtt{True}} \end{array}$$

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: $\forall x.\ x = x$
- free: $x = x$
- *schematic*: $?x = ?x$
  ("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 $\sigma = \{?xs \mapsto a\}$

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.

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

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: $\mathtt{b} + \mathtt{a} \rightsquigarrow \mathtt{a} + \mathtt{b}$ but not $\mathtt{a} + \mathtt{b} \rightsquigarrow \mathtt{b} + \mathtt{a}$.
For types `nat`, `int`, etc., commutative, associative and distributive laws built in.
Example: `proof simp` yields:

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

Preprocessing

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

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

Example:

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

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

proof = `proof` [*method*] *statement** `qed`
| `by` *method*

method = `(simp ...)|(auto ...)|(blast ...)|(rule ...)|` ...

statement =	`fix` *variable*$^+$	($\bigwedge$)
|	`assume` *proposition*	($\Longrightarrow$)
|	[`from` *name*$^+$] *objective proof*	
|	`next`	(starts next subgoal)
objective =	`show` *proposition*	(next proof step)
|	`have` *proposition*	(local claim)
|	`obtain` *variable*$^+$ `where` *proposition*$^+$	

proposition = [*name*:] *formula*

Proof Basics

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

Rule	Sequent Encoding
$\dfrac{A_1 \dots A_n}{A}$	$[\![A_1, \dots, A_n]\!] \Longrightarrow A$
$\dfrac{\begin{array}{c}B \\ \vdots \\ A_1 \dots\ A_i\ \dots A_n\end{array}}{A}$	$[\![A_1, \dots, B \Longrightarrow A_i, \dots, A_n]\!] \Longrightarrow A$

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 $A \oplus B$?

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

Operational Reading

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

Introduction rule:
To prove A it suffices to prove $A_1 \dots A_n$.

Elimination rule:
If we know A_1 and we want to prove A
it suffices to prove $A_2 \dots A_n$

Natural Deduction for Propositional Logic

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

$$\frac{A}{A \vee B}\qquad \frac{B}{A \vee B}\ \texttt{disjI1/2} \qquad \frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{c}\ \texttt{disjE}$$

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

Natural Deduction for Propositional Logic

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

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

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

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

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