DATA 61

COMP4161: Advanced Topics in Software Verification

$\longrightarrow$

June Andronick, Christine Rizkallah, Miki Tanaka, Johannes Åman Pohjola

T3/2019

data61.csiro.au

CSIRO

Content

- Intro & motivation, getting started [1]

- Foundations & Principles
  - Lambda Calculus, natural deduction [1,2]
  - Higher Order Logic, Isar (part 1) [3[a]]
  - Term rewriting [4]

- Proof & Specification Techniques
  - Inductively defined sets, rule induction [5]
  - Datatypes, recursion, induction, Isar (part 2) [6, 7[b]]
  - Hoare logic, proofs about programs, invariants [8]
  - C verification [9]
  - Practice, questions, exam prep [10[c]]

---

[a] a1 due; [b] a2 due; [c] a3 due

Last Time on HOL

➜ Defining HOL

Last Time on HOL

➜ Defining HOL
➜ Higher Order Abstract Syntax

Last Time on HOL

- ➜ Defining HOL
- ➜ Higher Order Abstract Syntax
- ➜ Deriving proof rules

Last Time on HOL

➜ Defining HOL
➜ Higher Order Abstract Syntax
➜ Deriving proof rules
➜ More automation

Term Rewriting

The Problem

Given a set of equations

$$\begin{gathered} l_1 = r_1 \\ l_2 = r_2 \\ \vdots \\ l_n = r_n \end{gathered}$$

The Problem

Given a set of equations

$$\begin{aligned} l_1 &= r_1 \\ l_2 &= r_2 \\ &\vdots \\ l_n &= r_n \end{aligned}$$

does equation $l = r$ hold?

The Problem

Given a set of equations

$$\begin{aligned} l_1 &= r_1 \\ l_2 &= r_2 \\ &\vdots \\ l_n &= r_n \end{aligned}$$

does equation $l = r$ hold?

Applications in:

➜ **Mathematics** (algebra, group theory, etc)

➜ **Functional Programming** (model of execution)

➜ **Theorem Proving** (dealing with equations, simplifying statements)

Term Rewriting: The Idea

use equations as reduction rules

$$\begin{array}{c} l_1 \longrightarrow r_1 \\ l_2 \longrightarrow r_2 \\ \vdots \\ l_n \longrightarrow r_n \end{array}$$

decide $l = r$ **by deciding** $l \stackrel{*}{\longleftrightarrow} r$

Arrow Cheat Sheet

$$\xrightarrow{0} \quad = \quad \{(x, y) | x = y\} \qquad \text{identity}$$

Arrow Cheat Sheet

$$\begin{array}{lcll} \xrightarrow{0} & = & \{(x,y) \mid x = y\} & \text{identity} \\ \xrightarrow{n+1} & = & \xrightarrow{n} \circ \longrightarrow & \text{n+1 fold composition} \end{array}$$

Arrow Cheat Sheet

$\xrightarrow{0}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\xrightarrow{n+1}$	$=$	$\xrightarrow{n} \circ \longrightarrow$	n+1 fold composition
$\xrightarrow{+}$	$=$	$\bigcup_{i>0} \xrightarrow{i}$	transitive closure

Arrow Cheat Sheet

$$
\begin{array}{lcll}
\xrightarrow{0} & = & \{(x,y)|x=y\} & \text{identity} \\
\xrightarrow{n+1} & = & \xrightarrow{n} \circ \longrightarrow & \text{n+1 fold composition} \\
\xrightarrow{+} & = & \bigcup_{i>0} \xrightarrow{i} & \text{transitive closure} \\
\xrightarrow{*} & = & \xrightarrow{+} \cup \xrightarrow{0} & \text{reflexive transitive closure}
\end{array}
$$

Arrow Cheat Sheet

$\xrightarrow{0}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\xrightarrow{n+1}$	$=$	$\xrightarrow{n} \circ \longrightarrow$	n+1 fold composition
$\xrightarrow{+}$	$=$	$\bigcup_{i>0} \xrightarrow{i}$	transitive closure
$\xrightarrow{*}$	$=$	$\xrightarrow{+} \cup \xrightarrow{0}$	reflexive transitive closure
$\xrightarrow{=}$	$=$	$\longrightarrow \cup \xrightarrow{0}$	reflexive closure

Arrow Cheat Sheet

$\xrightarrow{0}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\xrightarrow{n+1}$	$=$	$\xrightarrow{n} \circ \longrightarrow$	n+1 fold composition
$\xrightarrow{+}$	$=$	$\bigcup_{i>0} \xrightarrow{i}$	transitive closure
$\xrightarrow{*}$	$=$	$\xrightarrow{+} \cup \xrightarrow{0}$	reflexive transitive closure
$\xrightarrow{=}$	$=$	$\longrightarrow \cup \xrightarrow{0}$	reflexive closure
$\xrightarrow{-1}$	$=$	$\{(y,x) \mid x \longrightarrow y\}$	inverse

Arrow Cheat Sheet

$\xrightarrow{0}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\xrightarrow{n+1}$	$=$	$\xrightarrow{n} \circ \longrightarrow$	n+1 fold composition
$\xrightarrow{+}$	$=$	$\bigcup_{i>0} \xrightarrow{i}$	transitive closure
$\xrightarrow{*}$	$=$	$\xrightarrow{+} \cup \xrightarrow{0}$	reflexive transitive closure
$\xrightarrow{=}$	$=$	$\longrightarrow \cup \xrightarrow{0}$	reflexive closure
$\xrightarrow{-1}$	$=$	$\{(y,x) \mid x \longrightarrow y\}$	inverse
$\longleftarrow$	$=$	$\xrightarrow{-1}$	inverse

Arrow Cheat Sheet

$\xrightarrow{0}$	$=$	$\{(x,y) \mid x = y\}$	identity
$\xrightarrow{n+1}$	$=$	$\xrightarrow{n} \circ \longrightarrow$	n+1 fold composition
$\xrightarrow{+}$	$=$	$\bigcup_{i>0} \xrightarrow{i}$	transitive closure
$\xrightarrow{*}$	$=$	$\xrightarrow{+} \cup \xrightarrow{0}$	reflexive transitive closure
$\xrightarrow{=}$	$=$	$\longrightarrow \cup \xrightarrow{0}$	reflexive closure
$\xrightarrow{-1}$	$=$	$\{(y,x) \mid x \longrightarrow y\}$	inverse
$\longleftarrow$	$=$	$\xrightarrow{-1}$	inverse
$\longleftrightarrow$	$=$	$\longleftarrow \cup \longrightarrow$	symmetric closure

Arrow Cheat Sheet

$$
\begin{array}{lcll}
\xrightarrow{0} & = & \{(x,y) | x = y\} & \text{identity} \\
\xrightarrow{n+1} & = & \xrightarrow{n} \circ \longrightarrow & \text{n+1 fold composition} \\
\xrightarrow{+} & = & \bigcup_{i>0} \xrightarrow{i} & \text{transitive closure} \\
\xrightarrow{*} & = & \xrightarrow{+} \cup \xrightarrow{0} & \text{reflexive transitive closure} \\
\xrightarrow{=} & = & \longrightarrow \cup \xrightarrow{0} & \text{reflexive closure} \\
\xrightarrow{-1} & = & \{(y,x) | x \longrightarrow y\} & \text{inverse} \\
\longleftarrow & = & \xrightarrow{-1} & \text{inverse} \\
\longleftrightarrow & = & \longleftarrow \cup \longrightarrow & \text{symmetric closure} \\
\xleftrightarrow{+} & = & \bigcup_{i>0} \xleftrightarrow{i} & \text{transitive symmetric closure} \\
\xleftrightarrow{*} & = & \xleftrightarrow{+} \cup \xleftrightarrow{0} & \text{reflexive transitive symmetric closure}
\end{array}
$$

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β:

How to Decide $l \overset{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \overset{*}{\longrightarrow} n$ and $r \overset{*}{\longrightarrow} n$

Does this always work?

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

If $l \stackrel{*}{\longleftrightarrow} r$, will there always be a suitable n?

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

If $l \stackrel{*}{\longleftrightarrow} r$, will there always be a suitable n? **No!**

Example:

Rules: $f\ x \longrightarrow a, \quad g\ x \longrightarrow b, \quad f\ (g\ x) \longrightarrow b$

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

If $l \stackrel{*}{\longleftrightarrow} r$, will there always be a suitable n? **No!**

Example:

Rules: $f\ x \longrightarrow a, \quad g\ x \longrightarrow b, \quad f\ (g\ x) \longrightarrow b$

$f\ x \stackrel{*}{\longleftrightarrow} g\ x$ because $f\ x \longrightarrow a \longleftarrow f\ (g\ x) \longrightarrow b \longleftarrow g\ x$

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

If $l \stackrel{*}{\longleftrightarrow} r$, will there always be a suitable n? **No!**

Example:

Rules: $f\ x \longrightarrow a, \quad g\ x \longrightarrow b, \quad f\ (g\ x) \longrightarrow b$

$f\ x \stackrel{*}{\longleftrightarrow} g\ x$ because $f\ x \longrightarrow a \longleftarrow f\ (g\ x) \longrightarrow b \longleftarrow g\ x$

But: $f\ x \longrightarrow a$ and $g\ x \longrightarrow b$ and a, b in normal form

How to Decide $l \overset{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \overset{*}{\longrightarrow} n$ and $r \overset{*}{\longrightarrow} n$

Does this always work?

If $l \overset{*}{\longrightarrow} n$ and $r \overset{*}{\longrightarrow} n$ then $l \overset{*}{\longleftrightarrow} r$. Ok.

If $l \overset{*}{\longleftrightarrow} r$, will there always be a suitable n? **No!**

Example:

Rules: $f\ x \longrightarrow a$, $g\ x \longrightarrow b$, $f\ (g\ x) \longrightarrow b$

$f\ x \overset{*}{\longleftrightarrow} g\ x$ because $f\ x \longrightarrow a \longleftarrow f\ (g\ x) \longrightarrow b \longleftarrow g\ x$

But: $f\ x \longrightarrow a$ and $g\ x \longrightarrow b$ and a, b in normal form

Works only for systems with **Church-Rosser** property:

$$l \overset{*}{\longleftrightarrow} r \Longrightarrow \exists n.\ l \overset{*}{\longrightarrow} n \wedge r \overset{*}{\longrightarrow} n$$

How to Decide $l \stackrel{*}{\longleftrightarrow} r$

Same idea as for β: look for n such that $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$

Does this always work?

If $l \stackrel{*}{\longrightarrow} n$ and $r \stackrel{*}{\longrightarrow} n$ then $l \stackrel{*}{\longleftrightarrow} r$. Ok.

If $l \stackrel{*}{\longleftrightarrow} r$, will there always be a suitable n? **No!**

Example:

Rules: $f\ x \longrightarrow a$, $g\ x \longrightarrow b$, $f\ (g\ x) \longrightarrow b$

$f\ x \stackrel{*}{\longleftrightarrow} g\ x$ because $f\ x \longrightarrow a \longleftarrow f\ (g\ x) \longrightarrow b \longleftarrow g\ x$

But: $f\ x \longrightarrow a$ and $g\ x \longrightarrow b$ and a, b in normal form

Works only for systems with **Church-Rosser** property:

$$l \stackrel{*}{\longleftrightarrow} r \Longrightarrow \exists n.\ l \stackrel{*}{\longrightarrow} n \wedge r \stackrel{*}{\longrightarrow} n$$

Fact: $\longrightarrow$ is Church-Rosser iff it is confluent.

Confluence

$$\begin{tikzcd} & s \arrow[dl, "*"'] \arrow[dr, "*"] & \\ x & & y \end{tikzcd}$$

Problem:
is a given set of reduction rules confluent?

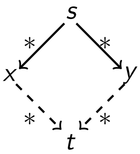
Confluence

$$\begin{tikzcd}[column sep=small]
 & s \arrow[dl, "*"'] \arrow[dr, "*"] & \\
x \arrow[dr, dashed, "*"'] & & y \arrow[dl, dashed, "*"] \\
 & t &
\end{tikzcd}$$

Problem:
is a given set of reduction rules confluent?

undecidable

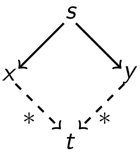
Confluence



Problem:
is a given set of reduction rules confluent?

undecidable

Local Confluence



Confluence

$$\begin{tikzcd}[column sep=small] & s \arrow[dl, "*"'] \arrow[dr, "*"] & \\ x \arrow[dr, dashed, "*"'] & & y \arrow[dl, dashed, "*"] \\ & t & \end{tikzcd}$$

Problem:
is a given set of reduction rules confluent?

undecidable

Local Confluence

$$\begin{tikzcd}[column sep=small] & s \arrow[dl] \arrow[dr] & \\ x \arrow[dr, dashed, "*"'] & & y \arrow[dl, dashed, "*"] \\ & t & \end{tikzcd}$$

Fact: local confluence and termination $\Longrightarrow$ confluence

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains
$\longrightarrow$ is **normalizing** if each element has a normal form
$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains
$\longrightarrow$ is **normalizing** if each element has a normal form
$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:
$\longrightarrow_\beta$ in λ is not terminating, but confluent

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains
$\longrightarrow$ is **normalizing** if each element has a normal form
$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:

$\longrightarrow_\beta$ in λ is not terminating, but confluent
$\longrightarrow_\beta$ in $\lambda^\rightarrow$ is terminating and confluent, i.e. convergent

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains
$\longrightarrow$ is **normalizing** if each element has a normal form
$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:
$\longrightarrow_\beta$ in λ is not terminating, but confluent
$\longrightarrow_\beta$ in $\lambda^\rightarrow$ is terminating and confluent, i.e. convergent

Problem: is a given set of reduction rules terminating?

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains
$\longrightarrow$ is **normalizing** if each element has a normal form
$\longrightarrow$ is **convergent** if it is terminating and confluent

Example:

$\longrightarrow_\beta$ in λ is not terminating, but confluent
$\longrightarrow_\beta$ in $\lambda^\rightarrow$ is terminating and confluent, i.e. convergent

Problem: is a given set of reduction rules terminating?

undecidable

When is $\longrightarrow$ Terminating?

Basic idea:

When is $\longrightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

When is $\longrightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

More formally: $\longrightarrow$ is terminating when there is a well founded order $<$ on terms for which $s < t$ whenever $t \longrightarrow s$ (well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Example:

When is $\longrightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

More formally: $\longrightarrow$ is terminating when there is a well founded order $<$ on terms for which $s < t$ whenever $t \longrightarrow s$ (well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Example: $f\ (g\ x) \longrightarrow g\ x$, $g\ (f\ x) \longrightarrow f\ x$

This system always terminates. Reduction order:

When is $\longrightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

More formally: $\longrightarrow$ is terminating when there is a well founded order $<$ on terms for which $s < t$ whenever $t \longrightarrow s$ (well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Example: $f\ (g\ x) \longrightarrow g\ x,\ g\ (f\ x) \longrightarrow f\ x$

This system always terminates. Reduction order:

$s <_r t$ iff $\mathit{size}(s) < \mathit{size}(t)$ with
$\mathit{size}(s) =$ number of function symbols in s

When is $\longrightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

More formally: $\longrightarrow$ is terminating when there is a well founded order $<$ on terms for which $s < t$ whenever $t \longrightarrow s$
(well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Example: $f\ (g\ x) \longrightarrow g\ x$, $g\ (f\ x) \longrightarrow f\ x$

This system always terminates. Reduction order:

$s <_r t$ iff $size(s) < size(t)$ with
$size(s)$ = number of function symbols in s

① Both rules always decrease *size* by 1 when applied to any term t

When is $\longrightarrow$ Terminating?

Basic idea: when each rule application makes terms simpler in some way.

More formally: $\longrightarrow$ is terminating when there is a well founded order $<$ on terms for which $s < t$ whenever $t \longrightarrow s$
(well founded = no infinite decreasing chains $a_1 > a_2 > \ldots$)

Example: $f\ (g\ x) \longrightarrow g\ x$, $g\ (f\ x) \longrightarrow f\ x$

This system always terminates. Reduction order:

$s <_r t$ iff $\mathit{size}(s) < \mathit{size}(t)$ with
$\mathit{size}(s)$ = number of function symbols in s

① Both rules always decrease *size* by 1 when applied to any term t

② $<_r$ is well founded, because $<$ is well founded on $\mathbb{N}$

Termination in Practice

In practice: often easier to consider just the rewrite rules by themselves, rather than their application to an arbitrary term t.

Show

Termination in Practice

In practice: often easier to consider just the rewrite rules by themselves, rather than their application to an arbitrary term t.

Show for each rule $l_i = r_i$, that $r_i < l_i$.

Termination in Practice

In practice: often easier to consider just the rewrite rules by themselves, rather than their application to an arbitrary term t.

Show for each rule $l_i = r_i$, that $r_i < l_i$.

Example:

$g\ x < f\ (g\ x)$ and $f\ x < g\ (f\ x)$

Requires

Termination in Practice

In practice: often easier to consider just the rewrite rules by themselves, rather than their application to an arbitrary term t.

Show for each rule $l_i = r_i$, that $r_i < l_i$.

Example:

$g\ x < f\ (g\ x)$ and $f\ x < g\ (f\ x)$

Requires

u to become smaller whenever any subterm of u is made smaller.

Formally:

Termination in Practice

In practice: often easier to consider just the rewrite rules by themselves, rather than their application to an arbitrary term t.

Show for each rule $l_i = r_i$, that $r_i < l_i$.

Example:

$g\ x < f\ (g\ x)$ and $f\ x < g\ (f\ x)$

Requires

u to become smaller whenever any subterm of u is made smaller.

Formally:

Requires $<$ to be **monotonic** with respect to the structure of terms:

$$s < t \longrightarrow u[s] < u[t].$$

Termination in Practice

In practice: often easier to consider just the rewrite rules by themselves, rather than their application to an arbitrary term t.

Show for each rule $l_i = r_i$, that $r_i < l_i$.

Example:

$g\ x < f\ (g\ x)$ and $f\ x < g\ (f\ x)$

Requires

u to become smaller whenever any subterm of u is made smaller.

Formally:

Requires $<$ to be **monotonic** with respect to the structure of terms:

$$s < t \longrightarrow u[s] < u[t].$$

True for most orders that don't treat certain parts of terms as special cases.

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

- ➜ Remove implications:

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

- Remove implications:

  imp: $(A \longrightarrow B) = (\neg A \vee B)$

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

- Remove implications:

  imp: $(A \longrightarrow B) = (\neg A \vee B)$

- Push $\neg$s down past other operators:

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

- Remove implications:

  imp: $(A \longrightarrow B) = (\neg A \vee B)$

- Push $\neg$s down past other operators:

  notnot: $(\neg\neg P) = P$

  notand: $(\neg(A \wedge B)) = (\neg A \vee \neg B)$

  notor: $(\neg(A \vee B)) = (\neg A \wedge \neg B)$

Example Termination Proof

Problem: Rewrite formulae containing $\neg$, $\wedge$, $\vee$ and $\longrightarrow$, so that they don't contain any implications and $\neg$ is applied only to variables and constants.

Rewrite Rules:

- Remove implications:

  imp: $(A \longrightarrow B) = (\neg A \vee B)$

- Push $\neg$s down past other operators:

  notnot: $(\neg\neg P) = P$

  notand: $(\neg(A \wedge B)) = (\neg A \vee \neg B)$

  notor: $(\neg(A \vee B)) = (\neg A \wedge \neg B)$

We show that the rewrite system defined by these rules is terminating.

Order on Terms

Each time one of our rules is applied, either:

- ➜ an implication is removed, or
- ➜ something that is not a $\neg$ is hoisted upwards in the term.

Order on Terms

Each time one of our rules is applied, either:

- an implication is removed, or
- something that is not a $\neg$ is hoisted upwards in the term.

This suggests a 2-part order, $<_r$: $s <_r t$ iff:

- num_imps $s <$ num_imps t, or
- num_imps $s =$ num_imps $t \wedge$ osize $s <$ osize t.

Order on Terms

Each time one of our rules is applied, either:

- an implication is removed, or
- something that is not a $\neg$ is hoisted upwards in the term.

This suggests a 2-part order, $<_r$: $s <_r t$ iff:

- num_imps $s <$ num_imps t, or
- num_imps $s =$ num_imps $t \wedge$ osize $s <$ osize t.

Let:

- $s <_i t \equiv$ num_imps $s <$ num_imps t and
- $s <_n t \equiv$ osize $s <$ osize t

Then $<_i$ and $<_n$ are both well-founded orders (since both return nats).

Order on Terms

Each time one of our rules is applied, either:

- ➜ an implication is removed, or
- ➜ something that is not a $\neg$ is hoisted upwards in the term.

This suggests a 2-part order, $<_r$: $s <_r t$ iff:

- ➜ num_imps $s <$ num_imps t, or
- ➜ num_imps $s =$ num_imps $t \wedge$ osize $s <$ osize t.

Let:

- ➜ $s <_i t \equiv$ num_imps $s <$ num_imps t and
- ➜ $s <_n t \equiv$ osize $s <$ osize t

Then $<_i$ and $<_n$ are both well-founded orders (since both return nats). $<_r$ is the lexicographic order over $<_i$ and $<_n$. $<_r$ is well-founded since $<_i$ and $<_n$ are both well-founded.

Order Decreasing

imp clearly decreases num_imps.

Order Decreasing

imp clearly decreases num_imps.

osize adds up all non-$\neg$ operators and variables/constants, weights each one according to its depth within the term.

Order Decreasing

imp clearly decreases num_imps.

osize adds up all non-$\neg$ operators and variables/constants, weights each one according to its depth within the term.

$$
\begin{array}{lll}
\text{osize}' \ c & x & = 2^x \\
\text{osize}' \ (\neg P) & x & = \text{osize}' \ P \ (x+1) \\
\text{osize}' \ (P \wedge Q) & x & = 2^x + (\text{osize}' \ P \ (x+1)) + (\text{osize}' \ Q \ (x+1)) \\
\text{osize}' \ (P \vee Q) & x & = 2^x + (\text{osize}' \ P \ (x+1)) + (\text{osize}' \ Q \ (x+1)) \\
\text{osize}' \ (P \longrightarrow Q) & x & = 2^x + (\text{osize}' \ P \ (x+1)) + (\text{osize}' \ Q \ (x+1)) \\
\text{osize} \ \ P & & = \text{osize}' \ P \ 0
\end{array}
$$

Order Decreasing

imp clearly decreases num_imps.
osize adds up all non-$\neg$ operators and variables/constants, weights each one according to its depth within the term.

$$
\begin{array}{lll}
\text{osize}'\ c & x & = 2^x \\
\text{osize}'\ (\neg P) & x & = \text{osize}'\ P\ (x+1) \\
\text{osize}'\ (P \wedge Q) & x & = 2^x + (\text{osize}'\ P\ (x+1)) + (\text{osize}'\ Q\ (x+1)) \\
\text{osize}'\ (P \vee Q) & x & = 2^x + (\text{osize}'\ P\ (x+1)) + (\text{osize}'\ Q\ (x+1)) \\
\text{osize}'\ (P \longrightarrow Q) & x & = 2^x + (\text{osize}'\ P\ (x+1)) + (\text{osize}'\ Q\ (x+1)) \\
\text{osize}\ \ P & & = \text{osize}'\ P\ 0
\end{array}
$$

The other rules decrease the depth of the things osize counts, so decrease osize.

Term Rewriting in Isabelle

Term rewriting engine in Isabelle is called **Simplifier**

Term Rewriting in Isabelle

Term rewriting engine in Isabelle is called **Simplifier**

apply simp

➜ uses simplification rules

Term Rewriting in Isabelle

Term rewriting engine in Isabelle is called **Simplifier**

apply simp

- ➜ uses simplification rules
- ➜ (almost) blindly from left to right

Term Rewriting in Isabelle

Term rewriting engine in Isabelle is called **Simplifier**

apply simp

- ➜ uses simplification rules
- ➜ (almost) blindly from left to right
- ➜ until no rule is applicable.

Term Rewriting in Isabelle

Term rewriting engine in Isabelle is called **Simplifier**

apply simp

- ➜ uses simplification rules
- ➜ (almost) blindly from left to right
- ➜ until no rule is applicable.

termination: not guaranteed
(may loop)

Term Rewriting in Isabelle

Term rewriting engine in Isabelle is called **Simplifier**

apply simp

- ➜ uses simplification rules
- ➜ (almost) blindly from left to right
- ➜ until no rule is applicable.

termination: not guaranteed
(may loop)

confluence: not guaranteed
(result may depend on which rule is used first)

Control

➜ Equations turned into simplification rules with **[simp]** attribute

Control

- Equations turned into simplification rules with **[simp]** attribute
- Adding/deleting equations locally:

  apply (simp add: <rules>)   and   **apply** (simp del: <rules>)

Control

- ➜ Equations turned into simplification rules with **[simp]** attribute
- ➜ Adding/deleting equations locally:

  apply (simp add: <rules>)   and   **apply** (simp del: <rules>)
- ➜ Using only the specified set of equations:

  apply (simp only: <rules>)

Demo

We have seen today...

- ➜ Equations and Term Rewriting

We have seen today...

- Equations and Term Rewriting

We have seen today...

- ➜ Equations and Term Rewriting
- ➜ Confluence and Termination of reduction systems

We have seen today...

- ➜ Equations and Term Rewriting
- ➜ Confluence and Termination of reduction systems
- ➜ Term Rewriting in Isabelle

Exercises

➜ Show, via a pen-and-paper proof, that the osize function is monotonic with respect to the structure of terms from that example.

