COMP4161
Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Miki Tanaka, Johannes Åman Pohjola

T3/2022

Content

➜ Foundations & Principles

- Intro, Lambda [illegible] atural deduction [1,2]
- Higher Order [illegible] part 1) [2,3[a]]
- Term rewriting [3,4]

➜ Proof & Specification Techniques

- Inductively defined sets, rule induction [4,5]
- Datatype induction, primitive recursion [5,7]
- General recursive functions, termination proofs [7[b]]
- Proof automation, Isar (part 2) [8]
- Hoare logic, proofs about programs, invariants [8,9]
- C verification [9,10]
- 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 Abstr

Last Time on HOL

- Defining HOL
- Higher Order Abstr
- Deriving proof rules

Last Time on HOL

- Defining HOL
- Higher Order Abstr
- Deriving proof rules
- More automation

Term Rewriting

The Problem

et of equations

$$l_1 = r_1$$
$$l_2 = r_2$$
$$\vdots$$
$$l_n = r_n$$

The Problem

et of equations

$$l_1 = r_1$$
$$l_2 = r_2$$
$$\vdots$$
$$l_n = r_n$$

does equation $l = r$ hold?

The Problem

et of equations

$$l_1 = r_1$$
$$l_2 = r_2$$
$$\vdots$$
$$l_n = r_n$$

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 $l_1 \longrightarrow r_1$, $l_2 \longrightarrow r_2$, ..., $l_n \longrightarrow r_n$ as reduction rules

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

Arrow Cheat Sheet

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

Arrow Cheat Sheet

$\xrightarrow{0} = \{(x, y) | x = v\}$ identity

$\xrightarrow{n+1} = \xrightarrow{n} \circ \longrightarrow$ n+1 fold composition

Arrow Cheat Sheet

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

$\xrightarrow{n+1} = \xrightarrow{n} \circ \longrightarrow$ $n+1$ fold composition

$\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i}$ transitive closure

Arrow Cheat Sheet

$\xrightarrow{0} = \{(x, y) | x = y\}$ 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

Arrow Cheat Sheet

$\xrightarrow{0} = \{(x, y) | x = y\}$ 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

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

$$\xrightarrow{n+1} = \xrightarrow{n} \circ \longrightarrow \quad \text{n+1 fold composition}$$

$$\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i} \quad \text{transitive closure}$$

$$\xrightarrow{*} = \xrightarrow{+} \cup \xrightarrow{0} \quad \text{reflexive transitive closure}$$

$$\xrightarrow{=} = \longrightarrow \cup \xrightarrow{0} \quad \text{reflexive closure}$$

$$\xrightarrow{-1} = \{(y,x) | x \longrightarrow y\} \quad \text{inverse}$$

Arrow Cheat Sheet

$\xrightarrow{0} = \{(x, y) | x = y\}$ 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

$\xrightarrow{-1} = \{(y, x) | x \longrightarrow y\}$ inverse

$\longleftarrow = \xrightarrow{-1}$ inverse

Arrow Cheat Sheet

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

$$\xrightarrow{n+1} = \xrightarrow{n} \circ \longrightarrow \quad \text{n+1 fold composition}$$

$$\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i} \quad \text{transitive closure}$$

$$\xrightarrow{*} = \xrightarrow{+} \cup \xrightarrow{0} \quad \text{reflexive transitive closure}$$

$$\xrightarrow{=} = \longrightarrow \cup \xrightarrow{0} \quad \text{reflexive closure}$$

$$\xrightarrow{-1} = \{(y,x) | x \longrightarrow y\} \quad \text{inverse}$$

$$\longleftarrow = \xrightarrow{-1} \quad \text{inverse}$$

$$\longleftrightarrow = \longleftarrow \cup \longrightarrow \quad \text{symmetric closure}$$

Arrow Cheat Sheet

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

$$\xrightarrow{n+1} = \xrightarrow{n} \circ \longrightarrow \quad \text{n+1 fold composition}$$

$$\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i} \quad \text{transitive closure}$$

$$\xrightarrow{*} = \xrightarrow{+} \cup \xrightarrow{0} \quad \text{reflexive transitive closure}$$

$$\xrightarrow{=} = \longrightarrow \cup \xrightarrow{0} \quad \text{reflexive closure}$$

$$\xrightarrow{-1} = \{(y,x) | x \longrightarrow y\} \quad \text{inverse}$$

$$\longleftarrow = \xrightarrow{-1} \quad \text{inverse}$$

$$\longleftrightarrow = \longleftarrow \cup \longrightarrow \quad \text{symmetric closure}$$

$$\xleftrightarrow{+} = \bigcup_{i>0} \xleftrightarrow{i} \quad \text{transitive symmetric closure}$$

$$\xleftrightarrow{*} = \xleftrightarrow{+} \cup \xleftrightarrow{0} \quad \text{reflexive transitive symmetric closure}$$

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

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 wor

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 wor

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

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 wor

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

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

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 wor

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

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

Example:

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

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

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

Does this always wor

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

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

Example:

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

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

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 wor

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

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

Example:

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

$f\ x \overset{*}{\longleftrightarrow} g\ x$ 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

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 wor

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

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

Example:

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

$f\ x \overset{*}{\longleftrightarrow} g\ x$ 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 $l \overset{*}{\longleftrightarrow} r$

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

Does this always wor

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

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

Example:

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

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

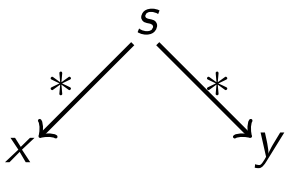
But: $f\ x \longrightarrow a$ and $g\ x \longrightarrow b$ and a, b in normal form

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

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

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

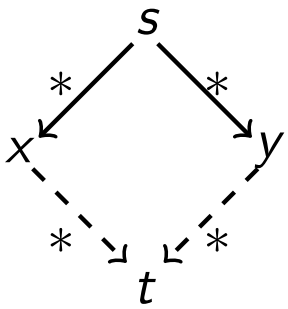
Confluence



n:

n set of reduction rules confluent?

Confluence

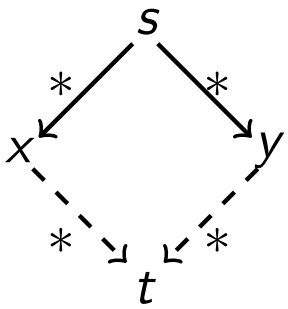


n:

n set of reduction rules confluent?

undecidable

Confluence

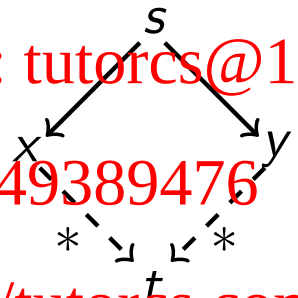


Local Confluence

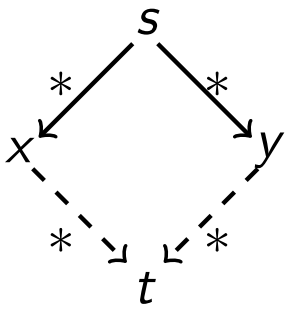
n:

n set of reduction rules confluent?

undecidable



Confluence

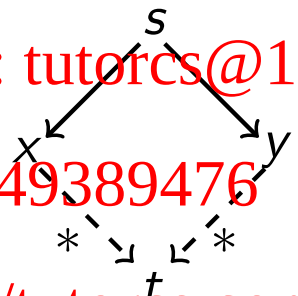


n:

n set of reduction rules confluent?

undecidable

Local Confluence



Fact: local confluence and termination $\Longrightarrow$ confluence

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains

$\longrightarrow$ is **normalizing** if every element has a normal form

$\longrightarrow$ is **convergent** if terminating and confluent

Example:

Termination

$\longrightarrow$ is **terminating** if there are no infinite reduction chains

$\longrightarrow$ is **normalizing** if every 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 every 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 every 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 every 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 $s < t$ 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 $t < t$ 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 $s < t$ 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

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 $s < t$ 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 $s < t$ 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
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 ap[illegible]o an arbitrary term t.

Show

Termination in Practice

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

Show for each rule $l_i =$ [illegible] $< l_i$.

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 = [illegible] $< l_i$.

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 that $t < l_i$.

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 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 impl[illegible]d $\neg$ 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 impl[illegible] 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 impl[illegible]d $\neg$ 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 impl[illegible]d $\neg$ 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 re
- ➜ something that is n isted upwards in the term.

Order on Terms

Each time one of our rules is applied, either:

- ➜ an implication is re
- ➜ something that is n isted upwards in the term.

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

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

Order on Terms

Each time one of our rules is applied, either:

- ➜ an implication is re
- ➜ something that is n isted upwards in the term.

This suggests a 2-part $s <_r t$ 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 re
- ➜ something that is n isted upwards in the term.

This suggests a 2-part $s <_r t$ 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-

and variables/constants, weights each one according to its de the term.

Order Decreasing

imp clearly decreases num_imps

osize adds up all non-  and variables/constants, weights each one according to its de the term.

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

Order Decreasing

imp clearly decreases num_imps

osize adds up all non-  and variables/constants, weights each one according to its de the term.

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

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

Term Rewriting in Isabelle

Term rewriti 

n Isabelle is called **Simplifier**

Term Rewriting in Isabelle

Term rewriting in Isabelle is called **Simplifier**

apply simp

➜ uses simplification rules

Term Rewriting in Isabelle

Term rewriting in Isabelle is called **Simplifier**

Simply simp

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

Term Rewriting in Isabelle

Term rewriting 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 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 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 in[illegible]ation rules with **[simp]** attribute

Control

- ➜ Equations turned in[illegible]ation rules with **[simp]** attribute
- ➜ Adding/deleting eq[illegible]lly:

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

Control

- ➜ Equations turned in[illegible]ation rules with **[simp]** attribute
- ➜ Adding/deleting eq[illegible]lly:

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

  apply (simp only: <rules>)

Demo

We have seen today...

- Equations and Tern

We have seen today...

- Equations and Tern

We have seen today...

- ➜ Equations and Tern
- ➜ Confluence and Ter

reduction systems

We have seen today...

- ➜ Equations and Tern
- ➜ Confluence and Ter
- ➜ Term Rewriting in I

reduction systems

Exercises

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