



# COMP4161

## Advanced Topics in Software Verification



Gerwin Klein, Miki Tanaka, Johannes Åman Pohjola, Robert Sison

T3/2023

# Content

## → Foundations & Principles

- Intro, Lambda calculus, natural deduction [1,2]
- Higher Order Logic, Isar (part 1) [2,3<sup>a</sup>]
- 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<sup>b</sup>]
- Proof automation, Isar (part 2) [8]
- Hoare logic, proofs about programs, invariants [8,9]
- C verification [9,10]
- Practice, questions, exam prep [10<sup>c</sup>]

---

<sup>a</sup>a1 due; <sup>b</sup>a2 due; <sup>c</sup>a3 due

## Last Time on HOL

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

# Term Rewriting

# The Problem

Given a set 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 equations as reduction rules

$$l_1 \longrightarrow r_1$$

$$l_2 \longrightarrow r_2$$

$\vdots$

$$l_n \longrightarrow r_n$$

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

## 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
$\longleftrightarrow$	$= \longleftarrow \cup \longrightarrow$	symmetric closure
$\xleftrightarrow{+}$	$= \bigcup_{i>0} \xleftrightarrow{i}$	transitive symmetric closure
$\xleftrightarrow{*}$	$= \xleftrightarrow{+} \cup \xleftrightarrow{0}$	reflexive transitive symmetric closure

## How to Decide $l \xleftrightarrow{*} r$

Same idea as for  $\beta$ : look for  $n$  such that  $l \xrightarrow{*} n$  and  $r \xrightarrow{*} n$

**Does this always work?**

If  $l \xrightarrow{*} n$  and  $r \xrightarrow{*} n$  then  $l \xleftrightarrow{*} r$ . Ok.

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

**Example:**

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

$f x \xleftrightarrow{*} g x$  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

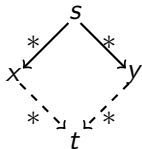
Works only for systems with **Church-Rosser** property:

$$l \xleftrightarrow{*} r \implies \exists n. l \xrightarrow{*} n \wedge r \xrightarrow{*} n$$

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



# Confluence

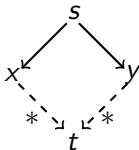


## Problem:

is a given set of reduction rules confluent?

**undecidable**

## Local Confluence



**Fact:** local confluence and termination  $\implies$  confluence

# Termination

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

## Example:

- <sub>β</sub> in  $\lambda$  is not terminating, but confluent
- <sub>β</sub> 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 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  $a_1 > a_2 > \dots$ )

**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 application to an arbitrary term  $t$ .

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

**Example:**

$$g\ x < f\ (g\ x) \text{ and } f\ x < g\ (f\ x)$$

**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.

### Rewrite Rules:

→ Remove implications:

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

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

$$\text{notnot: } (\neg\neg P) = P$$

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

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

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.

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

- $\text{num\_imps } s < \text{num\_imps } t$ , or
- $\text{num\_imps } s = \text{num\_imps } t \wedge \text{osize } s < \text{osize } t$ .

Let:

- $s <_i t \equiv \text{num\_imps } s < \text{num\_imps } t$  and
- $s <_n t \equiv \text{osize } s < \text{osize } t$

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 numimps.

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

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

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**

**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
- 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
- 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.