O • NICTA

COMP 4161 NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray



Slide 1

Oraclast	•
Content	NICTA
	Rough timeline
→ Intro & motivation, getting started	[1]
➔ Foundations & Principles	
Lambda Calculus, natural deduction	[2,3,4 ^a]
Higher Order Logic	[5,6 ^b ,7]
Term rewriting	[8,9,10 ^c]
Proof & Specification Techniques	
• Isar	[11,12 ^d]
 Inductively defined sets, rule induction 	[13 ^e ,15]
 Datatypes, recursion, induction 	[16,17 ^f ,18,19]
 Calculational reasoning, mathematics style proofs 	[20]
 Hoare logic, proofs about programs 	[21 ^g ,22,23]

^aa1 out; ^ba1 due; ^ca2 out; ^da2 due; ^esession break; ^fa3 out; ^ga3 due

Slide 2

Last Time



 \mathbf{O}

NICTA

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

Slide 3

Applying a Rewrite Rule $\rightarrow l \longrightarrow r$ applicable to term t[s]if there is substitution σ such that $\sigma l = s$

- → Result: $t[\sigma r]$
- → Equationally: $t[s] = t[\sigma r]$

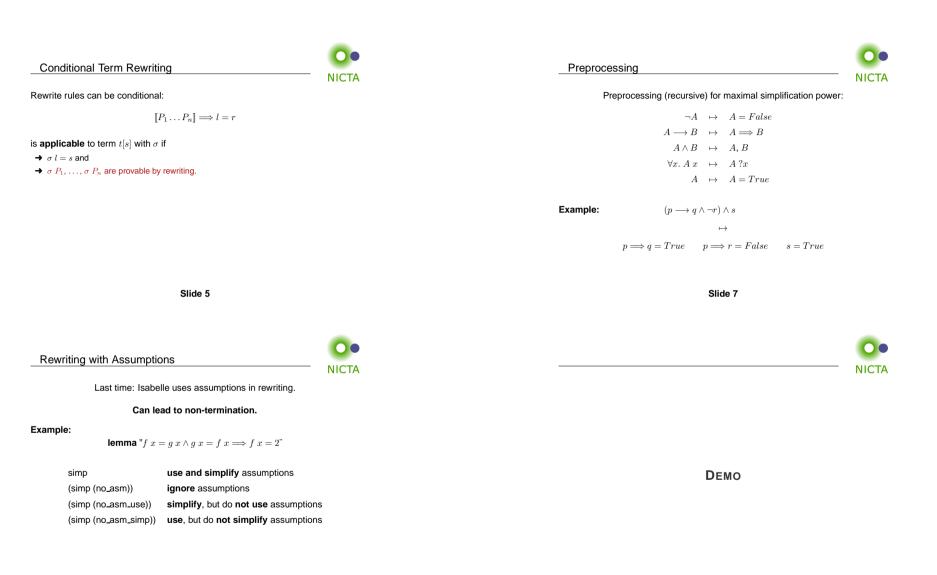
Example:

Rule: $0 + n \longrightarrow n$

Term: a + (0 + (b + c))

Substitution: $\sigma = \{n \mapsto b + c\}$

Result: a + (b + c)



Slide 6

Case splitting with simp



 $\begin{array}{c} P \ (\text{if} \ A \ \text{then} \ s \ \text{else} \ t) \\ = \\ (A \longrightarrow P \ s) \land (\neg A \longrightarrow P \ t) \end{array}$

Automatic

 $\begin{array}{l} P \; (\mathsf{case} \; e \; \mathsf{of} \; 0 \; \Rightarrow \; a \; | \; \mathsf{Suc} \; n \; \Rightarrow \; b) \\ = \\ (e = 0 \longrightarrow P \; a) \land (\forall n. \; e = \mathsf{Suc} \; n \longrightarrow P \; b) \end{array}$

Manually: apply (simp split: nat.split)

Similar for any data type t: t.split

Slide 9

Congruence Rules	O • NICTA
congruence rules are about using context	
Example : in $P \longrightarrow Q$ we could use P to simplify terms in Q	
For \Longrightarrow hardwired (assumptions used in rewriting)	
For other operators expressed with conditional rewriting.	
$\textbf{Example:} [\![P=P';P' \Longrightarrow Q=Q']\!] \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$	
Read : to simplify $P \longrightarrow Q$ \Rightarrow first simplify P to P'	

- → then simplify Q to Q' using P' as assumption
- \rightarrow the result is $P' \longrightarrow Q'$



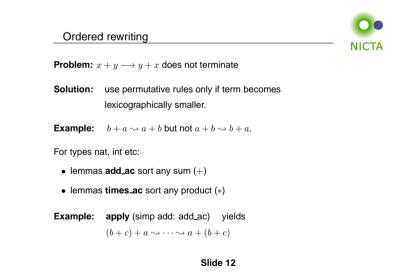
More Congruence

Sometimes useful, but not used automatically (slowdown): conj_cong: $[P = P'; P' \implies Q = Q'] \implies (P \land Q) = (P' \land Q')$ NICTA

Context for if-then-else: **if_cong**: $[b = c; c \Longrightarrow x = u; \neg c \Longrightarrow y = v] \Longrightarrow$ (if b then x else y) = (if c then u else v)

Prevent rewriting inside then-else (default): **if_weak_cong**: $b = c \implies$ (if b then x else y) = (if c then x else y)

- → declare own congruence rules with [cong] attribute
- → delete with [cong del]



AC Rules

NICTA

Example for associative-commutative rules: Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

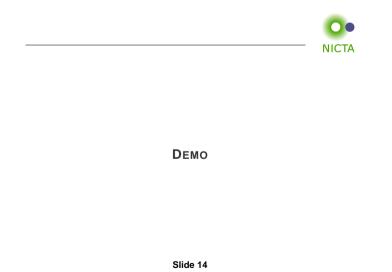
Commutative: $x \odot y = y \odot x$

These 2 rules alone get stuck too early (not confluent).

We need: AC rule $x \odot (y \odot z) = y \odot (x \odot z)$

If these 3 rules are present for an AC operator Isabelle will order terms correctly

Slide 13



Back to Confluence

Last time: confluence in general is undecidable. But: confluence for terminating systems is decidable! Problem: overlapping lhs of rules.

Definition:

Let $l_1 \longrightarrow r_1$ and $l_2 \longrightarrow r_2$ be two rules with disjoint variables. They form a **critical pair** if a non-variable subterm of l_1 unifies with l_2 .

Example:

(1)+(3)	$\{x \mapsto g \ z\}$	$a \stackrel{(1)}{\longleftarrow}$	f g t	$\xrightarrow{(3)} b$
(3)+(2)	$\{z \mapsto y\}$	$b \xleftarrow{(3)}$	f g t	$\stackrel{(2)}{\longrightarrow} b$

Slide 15

Completion



NICTA

(1) $f x \longrightarrow a$ (2) $g y \longrightarrow b$ (3) $f (g z) \longrightarrow b$ is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:

(1)+(3)
$$\{x \mapsto g z\}$$
 $a \stackrel{(1)}{\longleftarrow} f g t \stackrel{(3)}{\longrightarrow} b$
shows that $a = b$ (because $a \stackrel{\leftrightarrow}{\longleftrightarrow} b$), so we add $a \longrightarrow b$ as a rule

This is the main idea of the Knuth-Bendix completion algorithm.

