ļ	0	•
v	IC.	ТА

COMP 4161 NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray, Rafal Kolanski

 λ^{\rightarrow}

Slide 1

Last time...

- $\rightarrow \lambda$ calculus syntax
- → free variables, substitution
- → β reduction
- → α and η conversion
- → β reduction is confluent
- → λ calculus is expressive (turing complete)
- → λ calculus is inconsistent

0.	
NICTA	

	0.
Content	
	NICIA
→ Intro & motivation, getting started	[1]
➔ Foundations & Principles	
 Lambda Calculus, natural deduction 	[1,2]
Higher Order Logic	[3 ^a]
Term rewriting	[4]
➔ Proof & Specification Techniques	
• Isar	[5]
 Inductively defined sets, rule induction 	[6 ^b]
 Datatypes, recursion, induction 	[7 ^c , 8]
 Calculational reasoning, code generation 	[9]
Hoare logic, proofs about programs	[10 ^d ,11,12]
$^a{\rm a1}$ due; $^b{\rm a2}$ due; $^c{\rm session}$ break; $^d{\rm a3}$ due	

Slide 3

 λ calculus is inconsistent Can find term *R* such that $R R =_{\beta} \operatorname{not}(R R)$



There are more terms that do not make sense: $1\,2,\,\,{\rm true\,false},\,\,{\rm etc.}$

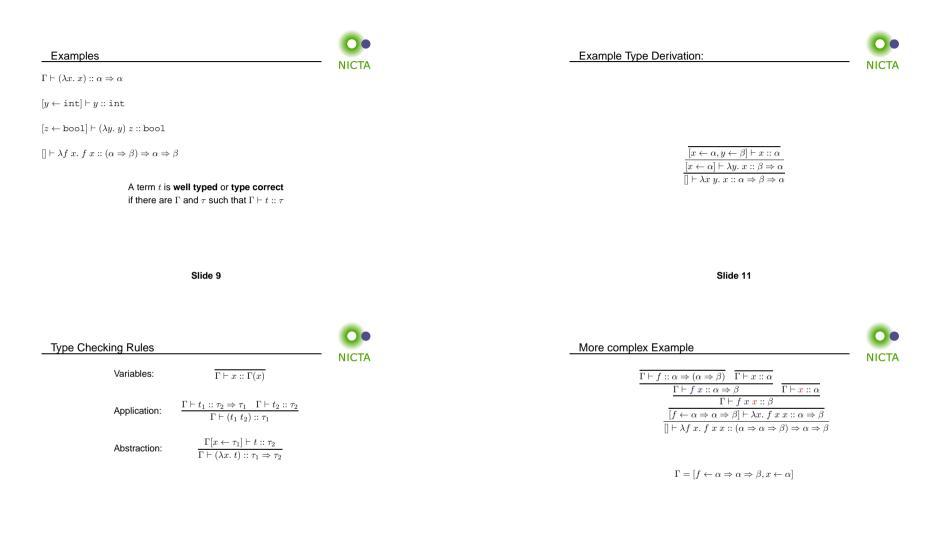
Solution: rule out ill-formed terms by using types. (Church 1940)

Slide 2

Slide 4

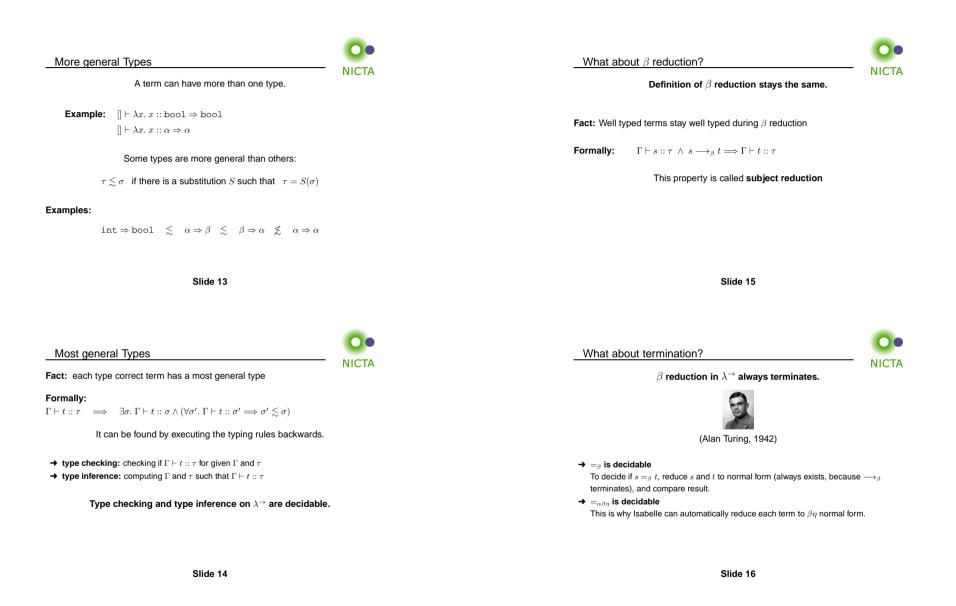
1

Introducing types	0•		
Idea: assign a type to each "sensible" λ term.		NIC	
Examples: \Rightarrow for term t has type α write $t :: \alpha$ \Rightarrow if x has type α then $\lambda x. x$ is a function from α to α Write: $(\lambda x. x) :: \alpha \Rightarrow \alpha$ \Rightarrow for $s t$ to be sensible: s must be function t must be right type for parameter If $s :: \alpha \Rightarrow \beta$ and $t :: \alpha$ then $(s t) :: \beta$		Now Formally Again	
Slide 5		Slide 7	
	NICTA	Syntax for λ^{\rightarrow}	
	Merk	Terms: $t ::= v \mid c \mid (t \ t) \mid (\lambda x. \ t)$ $v, x \in V, c \in C, V, C$ sets of names	MCIA
That's about it		Types: $\tau ::= b \mid \nu \mid \tau \Rightarrow \tau$ $b \in \{bool, int,\}$ base types $\nu \in \{\alpha, \beta,\}$ type variables	
		$\alpha \Rightarrow \beta \Rightarrow \gamma = \alpha \Rightarrow (\beta \Rightarrow \gamma)$	
		Context Γ : Γ : function from variable and constant names to types.	
		Term t has type $ au$ in context Γ : $\Gamma \vdash t :: au$	
Slide 6		Slide 8	





Slide 12



What does this mean for Expressiveness?

NICTA

Not all computable functions can be expressed in λ^{\rightarrow} !

How can typed functional languages then be turing complete?

Fact:

Each computable function can be encoded as closed, type correct λ^{\rightarrow} term using $Y :: (\tau \Rightarrow \tau) \Rightarrow \tau$ with $Y t \longrightarrow_{\beta} t (Y t)$ as only constant.

- \rightarrow Y is called fix point operator
- → used for recursion
- → lose decidability (what does $Y(\lambda x. x)$ reduce to?)

Slide 17

-			
Types	and Terms in Isabelle		
		NICT	
Types:	$\tau ::= \mathbf{b} \mid \mathbf{\nu} \mid \mathbf{\nu} :: C \mid \tau \Rightarrow \tau \mid (\tau, \dots, \tau) K$		
	$b \in \{bool, int, \ldots\}$ base types		
	$\nu \in \{\alpha, \beta, \ldots\}$ type variables		
	$K \in \{\texttt{set}, \texttt{list}, \ldots\}$ type constructors		
	$C \in \{\texttt{order}, \texttt{linord}, \ldots\}$ type classes		
Terms:	$t ::= v \mid c \mid ?v \mid (t t) \mid (\lambda x. t)$		
	$v, x \in V, c \in C, V, C \text{ sets of names}$		
→ type c	onstructors: construct a new type out of a parameter type.		
Examp	Dle: int list		
→ type c	lasses: restrict type variables to a class defined by axioms.		
Exam	ble: α :: order		
→ schematic variables: variables that can be instantiated.			

Type Classes

- → similar to Haskell's type classes, but with semantic properties
 class order =
 assumes order_refl: "x ≤ x"
 assumes order_trans: "[x ≤ y; y ≤ z] ⇒ x ≤ z"
 ...
- → theorems can be proved in the abstract lemma order.less_trans: " $\land x :: 'a :: order. [[x < y; y < z]] \implies x < z$ "
- → can be used for subtyping class linorder = order + assumes linorder_linear: "x < y ∨ y < x"</p>
- → can be instantiated instance nat :: "{order, linorder}" by ...

Slide 19

Schematic Variables Image: Nicta Nicta $\frac{X - Y}{X \wedge Y}$ Nicta

 \rightarrow X and Y must be **instantiated** to apply the rule

But: lemma "x + 0 = 0 + x"

- → x is free
- → convention: lemma must be true for all x
- \rightarrow during the proof, x must not be instantiated

Solution:

Isabelle has free (x), bound (x), and schematic (?X) variables.

Only schematic variables can be instantiated.

Free converted into schematic after proof is finished.

Slide 20

NICTA

Higher Order Unification

NICTA

Unification:

Find substitution σ on variables for terms s, t such that $\sigma(s) = \sigma(t)$

In Isabelle:

Find substitution σ on schematic variables such that $\sigma(s) =_{\alpha\beta\eta} \sigma(t)$

Examples:

$?X \land ?Y$	$=_{\alpha\beta\eta}$	$x \wedge x$	$[?X \leftarrow x, ?Y \leftarrow x]$
?P x	$=_{\alpha\beta\eta}$	$x \wedge x$	$[?P \leftarrow \lambda x. \ x \wedge x]$
P(?f x)	$=_{\alpha\beta\eta}$?Y x	$[?f \leftarrow \lambda x. \ x, ?Y \leftarrow P]$

Slide 21

Higher Order: schematic variables can be functions.

We have learned so far ..



- → Simply typed lambda calculus: λ^{\rightarrow}
- → Typing rules for λ^{\rightarrow} , type variables, type contexts
- → β -reduction in λ^{\rightarrow} satisfies subject reduction
- → β -reduction in λ^{\rightarrow} always terminates
- ➔ Types and terms in Isabelle

Slide 23

Higher Order Unification

- NICTA
- → Unification modulo $\alpha\beta$ (Higher Order Unification) is semi-decidable
- → Unification modulo $\alpha\beta\eta$ is undecidable
- → Higher Order Unification has possibly infinitely many solutions

But:

- ➔ Most cases are well-behaved
- → Important fragments (like Higher Order Patterns) are decidable

Higher Order Pattern:

- \twoheadrightarrow is a term in β normal form where
- \Rightarrow each occurrence of a schematic variable is of the form $?f t_1 \ \ldots \ t_n$
- → and the $t_1 \ldots t_n$ are η -convertible into n distinct bound variables

