Slide 1

NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications

$$\lambda^{\longrightarrow}$$

Slide 2

CONTENT

- ➜ Intro & motivation, getting started with Isabelle
- ➜ **Foundations & Principles**
  - **Lambda Calculus**
  - **Higher Order Logic, natural deduction**
  - Term rewriting
- ➜ Proof & Specification Techniques
  - Datatypes, recursion, induction
  - Inductively defined sets, rule induction
  - Calculational reasoning, mathematics style proofs
  - Hoare logic, proofs about programs

Slide 3

λ CALCULUS IS INCONSISTENT

From last lecture:
Can find term R such that $R\ R =_\beta \texttt{not}(R\ R)$

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

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

Slide 4

INTRODUCING TYPES

Idea: assign a type to each "sensible" λ term.

Examples:

- ➜ for *term t has type α* write $t :: \alpha$
- ➜ if x has type α then $\lambda x.\ x$ is a function from α to α
  Write: $(\lambda x.\ x) :: \alpha \Rightarrow a$
- ➜ 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 $(s\ t) :: \beta$

Slide 5

THAT'S ABOUT IT

Slide 6

NOW FORMALLY, AGAIN

Slide 7

SYNTAX FOR $\lambda^{\rightarrow}$

Terms: $t \quad ::= \quad v \mid c \mid (t\ t) \mid (\lambda x.\ t)$
$v, x \in V, \quad c \in C, \quad V, C$ sets of names

Types: $\tau \quad ::= \quad \mathsf{b} \mid \nu \mid \tau \Rightarrow \tau$
$\mathsf{b} \in \{\texttt{bool}, \texttt{int}, \ldots\}$ base types
$\nu \in \{\alpha, \beta, \ldots\}$ type variables

$$\alpha \Rightarrow \beta \Rightarrow \gamma \quad = \quad \alpha \Rightarrow (\beta \Rightarrow \gamma)$$

Contexts Γ:

Γ: function from variable and constant names to types.

Term t has type τ in context Γ: $\quad \Gamma \vdash t :: \tau$

Slide 8

EXAMPLES

$\Gamma \vdash (\lambda x.\ x) :: \alpha \Rightarrow \alpha$

$[y \leftarrow \texttt{int}] \vdash y :: \texttt{int}$

$[z \leftarrow \texttt{bool}] \vdash (\lambda y.\ y)\ z :: \texttt{bool}$

$[] \vdash \lambda f\ x.\ f\ x :: (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$

A term t is **well typed** or **type correct**
if there are Γ and τ such that $\Gamma \vdash t :: \tau$

Type Checking Rules

Slide 9

Variables: $$\frac{}{\Gamma \vdash x :: \Gamma(x)}$$

Application: $$\frac{\Gamma \vdash t_1 :: \tau_2 \Rightarrow \tau_1 \quad \Gamma \vdash t_2 :: \tau_2}{\Gamma \vdash (t_1\ t_2) :: \tau_1}$$

Abstraction: $$\frac{\Gamma[x \leftarrow \tau_1] \vdash t :: \tau_2}{\Gamma \vdash (\lambda x.\ t) :: \tau_1 \Rightarrow \tau_2}$$

Example Type Derivation:

Slide 10

$$\frac{\dfrac{\dfrac{}{[x \leftarrow \alpha, y \leftarrow \beta] \vdash x :: \alpha}}{[x \leftarrow \alpha] \vdash \lambda y.\ x :: \beta \Rightarrow \alpha}}{[] \vdash \lambda x\ y.\ x :: \alpha \Rightarrow \beta \Rightarrow \alpha}$$

More complex Example

Slide 11

$$\frac{\dfrac{\dfrac{\dfrac{\overline{\Gamma \vdash f :: \alpha \Rightarrow (\alpha \Rightarrow \beta)} \quad \overline{\Gamma \vdash x :: \alpha}}{\Gamma \vdash f\ x :: \alpha \Rightarrow \beta} \quad \overline{\Gamma \vdash x :: \alpha}}{\Gamma \vdash f\ x\ x :: \beta}}{[f \leftarrow \alpha \Rightarrow \alpha \Rightarrow \beta] \vdash \lambda x.\ f\ x\ x :: \alpha \Rightarrow \beta}}{[] \vdash \lambda f\ x.\ f\ x\ x :: (\alpha \Rightarrow \alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta}$$

$$\Gamma = [f \leftarrow \alpha \Rightarrow \alpha \Rightarrow \beta, x \leftarrow \alpha]$$

More general Types

Slide 12

A term can have more than one type.

Example: $[] \vdash \lambda x.\ x :: \texttt{bool} \Rightarrow \texttt{bool}$

$[] \vdash \lambda x.\ x :: \alpha \Rightarrow \alpha$

Some types are more general than others:

$\tau \lesssim \sigma$ if there is a substitution S such that $\tau = S(\sigma)$

Examples:

$$\texttt{int} \Rightarrow \texttt{bool} \quad \lesssim \quad \alpha \Rightarrow \beta \quad \lesssim \quad \beta \Rightarrow \alpha \quad \not\lesssim \quad \alpha \Rightarrow \alpha$$

MOST GENERAL TYPES

Slide 13

Fact: each type correct term has a most general type

Formally:
$\Gamma \vdash t :: \tau \quad \Longrightarrow \quad \exists \sigma.\, \Gamma \vdash t :: \sigma \wedge (\forall \sigma'.\, \Gamma \vdash t :: \sigma' \Longrightarrow \sigma' \lesssim \sigma)$

It can be found by executing the typing rules backwards.

- ➜ **type checking:** checking if $\Gamma \vdash t :: \tau$ for given Γ and τ
- ➜ **type inference:** computing Γ and τ such that $\Gamma \vdash t :: \tau$

Type checking and type inference on $\lambda^{\rightarrow}$ are decidable.

WHAT ABOUT β REDUCTION?

Slide 14

Definition of β reduction stays the same.

Fact: Well typed terms stay well typed during β reduction

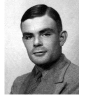
Formally: $\Gamma \vdash s :: \tau \ \wedge \ s \longrightarrow_\beta t \Longrightarrow \Gamma \vdash t :: \tau$

This property is called **subject reduction**

WHAT ABOUT TERMINATION?

Slide 15

β reduction in $\lambda^{\rightarrow}$ always terminates.



(Alan Turing, 1942)

- ➜ **$=_\beta$ is decidable**
  To decide if $s =_\beta t$, reduce s and t to normal form (always exists, because $\longrightarrow_\beta$ terminates), and compare result.
- ➜ **$=_{\alpha\beta\eta}$ is decidable**
  This is why Isabelle can automatically reduce each term to $\beta\eta$ normal form.

WHAT DOES THIS MEAN FOR EXPRESSIVENESS?

Slide 16

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

How can typed functional languages then be turing complete?

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

- ➜ Y is called fix point operator
- ➜ used for recursion

Types and Terms in Isabelle

Slide 17

Types: $\tau \;::=\; \mathsf{b} \mid {}'\nu \mid {}'\nu :: C \mid \tau \Rightarrow \tau \mid (\tau, \ldots, \tau)\, K$

b $\in$ {`bool`, `int`, ...} base types

$\nu \in \{\alpha, \beta, \ldots\}$ type variables

$K \in$ {`set`, `list`, ...} type constructors

$C \in$ {`order`, `linord`, ...} type classes

Terms: $t \;::=\; v \mid c \mid ?v \mid (t\; t) \mid (\lambda x.\, t)$

$v, x \in V, \quad c \in C, \quad V, C$ sets of names

- **type constructors**: construct a new type out of a parameter type.
  Example: `int list`
- **type classes**: restrict type variables to a class defined by axioms.
  Example: $\alpha :: order$
- **schematic variables**: variables that can be instantiated.

Type Classes

Slide 18

- similar to Haskell's type classes, but with semantic properties

  axclass order < ord
  order_refl: "$x \leq x$"
  order_trans: "$[\![x \leq y; y \leq z]\!] \Longrightarrow x \leq z$"
  ...
- theorems can be proved in the abstract

  lemma order_less_trans: "$\bigwedge x :: {}'a :: order.\ [\![x < y; y < z]\!] \Longrightarrow x < z$"
- can be used for subtyping

  axclass linorder < order
  linorder_linear: "$x \leq y \vee y \leq x$"
- can be instantiated

  instance nat :: "{order, linorder}" **by** ...

Schematic Variables

Slide 19

$$\frac{X \quad Y}{X \wedge Y}$$

- 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
- **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 fi nished.

Higher Order Unification

Slide 20

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:

$$\begin{array}{llll}
?X \wedge ?Y & =_{\alpha\beta\eta} & x \wedge x & [?X \leftarrow x, ?Y \leftarrow x] \\
?P\; x & =_{\alpha\beta\eta} & x \wedge x & [?P \leftarrow \lambda x.\, x \wedge x] \\
P\; (?f\; x) & =_{\alpha\beta\eta} & ?Y\; x & [?f \leftarrow \lambda x.\, x, ?Y \leftarrow P]
\end{array}$$

Higher Order: schematic variables can be functions.

Slide 21

Higher Order Unification

- ➜ 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:

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

Slide 22

We have learned so far...

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

Slide 23

Preview: Proofs in Isabelle

Slide 24

Proofs in Isabelle

General schema:

lemma name: "<goal>"
apply <method>
apply <method>
...
done

- ➜ Sequential application of methods until all **subgoals** are solved.

THE PROOF STATE

Slide 25

1. $\bigwedge x_1 \dots x_p. [\![A_1; \dots; A_n]\!] \Longrightarrow B$
2. $\bigwedge y_1 \dots y_q. [\![C_1; \dots; C_m]\!] \Longrightarrow D$

$x_1 \dots x_p$	Parameters
$A_1 \dots A_n$	Local assumptions
B	Actual (sub)goal

ISABELLE THEORIES

Slide 26

Syntax:

theory $MyTh$ = $ImpTh_1$ + ... + $ImpTh_n$:
(declarations, definitions, theorems, proofs, ...)*
end

➜ $MyTh$: name of theory. Must live in file $MyTh$.thy

➜ $ImpTh_i$: name of *imported* theories. Import transitive.

Unless you need something special:

theory $MyTh$ = Main:

NATURAL DEDUCTION RULES

Slide 27

$$\frac{A \quad B}{A \wedge B} \text{ conjI} \qquad \frac{A \wedge B \quad [\![A; B]\!] \Longrightarrow C}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2} \qquad \frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{ disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{ impI} \qquad \frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C} \text{ impE}$$

For each connective ($\wedge, \vee$, etc):
introduction and **elemination** rules

PROOF BY ASSUMPTION

Slide 28

apply assumption

proves

1. $[\![B_1; \dots; B_m]\!] \Longrightarrow C$

by unifying C with one of the B_i

There may be more than one matching B_i and multiple unifiers.

Backtracking!

Explicit backtracking command: **back**

INTRO RULES

Slide 29

Intro rules decompose formulae to the right of $\Longrightarrow$.

apply (rule <intro-rule>)

Intro rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ means

- ➜ To prove A it suffices to show $A_1 \ldots A_n$

Applying rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ to subgoal C:

- ➜ unify A and C
- ➜ replace C with n new subgoals $A_1 \ldots A_n$

ELIM RULES

Slide 30

Elim rules decompose formulae on the left of $\Longrightarrow$.

apply (erule <elim-rule>)

Elim rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ means

- ➜ If I know A_1 and want to prove A it suffices to show $A_2 \ldots A_n$

Applying rule $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ to subgoal C:
Like **rule** but also

- ➜ unifies first premise of rule with an assumption
- ➜ eliminates that assumption

Slide 31

DEMO

EXERCISES

Slide 32

- ➜ what are the types of $\lambda x\ y.\ y\ x$ and $\lambda x\ y\ z.\ x\ y\ (y\ z)$
- ➜ construct a type derivation tree on paper for $\lambda x\ y\ z.\ x\ y\ (y\ z)$
- ➜ find a unifier (substitution) such that $\lambda x\ y.\ ?F\ x = \lambda x\ y.\ c\ (?G\ y\ x)$
- ➜ prove $(A \longrightarrow B \longrightarrow C) = (A \wedge B \longrightarrow C)$ in Isabelle
- ➜ prove $\neg(A \wedge B) \Longrightarrow \neg A \vee \neg B$ in Isabelle (tricky!)