COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein
Formal Methods

λ

Enough Theory!
Getting started with Isabelle

System Architecture

Proof General – user interface

HOL, ZF – object-logics

Isabelle – generic, interactive theorem prover

Standard ML – logic implemented as ADT

User can access all layers!

System Requirements

- ➜ **Linux**, **FreeBSD**, **MacOS X** or **Solaris**
- ➜ **Standard ML**
  (PolyML fastest, SML/NJ supports more platforms)
- ➜ **XEmacs** or **Emacs**
  (for ProofGeneral)

If you have only Windows, try IsaMorph
`http://www.brucker.ch/projects/isamorph/` or
install Cygwin.

DOCUMENTATION

Available from `http://isabelle.in.tum.de`

➜ Learning Isabelle

- Tutorial on Isabelle/HOL (LNCS 2283)
- Tutorial on Isar
- Tutorial on Locales

➜ Reference Manuals

- Isabelle/Isar Reference Manual
- Isabelle Reference Manual
- Isabelle System Manual

➜ Reference Manuals for Object-Logics

PROOFGENERAL

➜ User interface for Isabelle
➜ Runs under XEmacs or Emacs
➜ Isabelle process in background

Interaction via

➜ Basic editing in XEmacs (with highlighting etc)
➜ Buttons (tool bar)
➜ Key bindings
➜ ProofGeneral Menu (lots of options, try them)

X-SYMBOL CHEAT SHEET

Input of funny symbols in ProofGeneral

➜ via menu (“X-Symbol”)
➜ via ASCII encoding (similar to LaTeX): \<and>, \<or>, ...
➜ via abbreviation: /\, \/, -->, ...
➜ via *rotate*: l C-. = λ (cycles through variations of letter)

<table>
<tr><th></th><th>∀</th><th>∃</th><th>λ</th><th>¬</th><th>∧</th><th>∨</th><th>⟶</th><th>⇒</th></tr>
<tr><td>①</td><td>\<forall></td><td>\<exists></td><td>\<lambda></td><td>\<not></td><td>/\</td><td>\/</td><td>--></td><td>=></td></tr>
<tr><td>②</td><td>ALL</td><td>EX</td><td>%</td><td>~</td><td>&</td><td>|</td><td></td><td></td></tr>
</table>

① converted to X-Symbol
② stays ASCII

DEMO

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

λ-Calculus

Alonzo Church

- lived 1903–1995
- supervised people like Alan Turing, Stephen Kleene
- famous for Church-Turing thesis, lambda calculus, first undecidability results
- invented λ calculus in 1930's

λ-calculus

- originally meant as foundation of mathematics
- important applications in theoretical computer science
- foundation of computability and functional programming

Untyped λ-Calculus

- turing complete model of computation
- a simple way of writing down functions

Basic intuition:

instead of $f(x) = x + 5$
write $f = \lambda x.\ x + 5$

$\lambda x.\ x + 5$

- a term
- a nameless function
- that adds 5 to its parameter

Function Application

For applying arguments to functions

instead of $f(x)$
write $f\ x$

Example: $(\lambda x.\ x + 5)\ a$

Evaluating: in $(\lambda x.\ t)\ a$ replace x by a in t
(computation!)

Example: $(\lambda x.\ x + 5)\ (a + b)$ evaluates to $(a + b) + 5$

NICTA

THAT'S IT!

Slide 13

NICTA

NOW FORMAL

Slide 14

NICTA

SYNTAX

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

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

- ➜ v, x variables
- ➜ c constants
- ➜ $(t\; t)$ application
- ➜ $(\lambda x.\; t)$ abstraction

Slide 15

NICTA

CONVENTIONS

- ➜ leave out parentheses where possible
- ➜ list variables instead of multiple λ

Example: instead of $(\lambda y.\; (\lambda x.\; (x\; y)))$ write $\lambda y\; x.\; x\; y$

Rules:

- ➜ list variables: $\lambda x.\; (\lambda y.\; t) = \lambda x\; y.\; t$
- ➜ application binds to the left: $x\; y\; z = (x\; y)\; z \neq x\; (y\; z)$
- ➜ abstraction binds to the right: $\lambda x.\; x\; y = \lambda x.\; (x\; y) \neq (\lambda x.\; x)\; y$
- ➜ leave out outermost parentheses

Slide 16

Getting used to the Syntax

Example:

$\lambda x\ y\ z.\ x\ z\ (y\ z) =$

$\lambda x\ y\ z.\ (x\ z)\ (y\ z) =$

$\lambda x\ y\ z.\ ((x\ z)\ (y\ z)) =$

$\lambda x.\ \lambda y.\ \lambda z.\ ((x\ z)\ (y\ z)) =$

$(\lambda x.\ (\lambda y.\ (\lambda z.\ ((x\ z)\ (y\ z)))))$

Slide 17

Computation

Intuition: replace parameter by argument
this is called β-reduction

Example

$(\lambda x\ y.\ f\ (y\ x))\ \ 5\ \ (\lambda x.\ x) \longrightarrow_\beta$
$(\lambda y.\ f\ (y\ 5))\ \ (\lambda x.\ x) \longrightarrow_\beta$
$f\ ((\lambda x.\ x)\ 5) \longrightarrow_\beta$
$f\ 5$

Slide 18

Defining Computation

β **reduction:**

$$\begin{array}{rclcrcl} & & & & (\lambda x.\ s)\ t & \longrightarrow_\beta & s[x \leftarrow t] \\ s & \longrightarrow_\beta & s' & \Longrightarrow & (s\ t) & \longrightarrow_\beta & (s'\ t) \\ t & \longrightarrow_\beta & t' & \Longrightarrow & (s\ t) & \longrightarrow_\beta & (s\ t') \\ s & \longrightarrow_\beta & s' & \Longrightarrow & (\lambda x.\ s) & \longrightarrow_\beta & (\lambda x.\ s') \end{array}$$

Still to do: define $s[x \leftarrow t]$

Slide 19

Defining Substitution

Easy concept. Small problem: variable capture.

Example: $(\lambda x.\ x\ z)[z \leftarrow x]$

We do **not** want: $(\lambda x.\ x\ x)$ as result.

What do we want?

In $(\lambda y.\ y\ z)\ [z \leftarrow x] = (\lambda y.\ y\ x)$ there would be no problem.

So, solution is: rename bound variables.

Slide 20

Free Variables

NICTA

Bound variables: in $(\lambda x.\ t)$, x is a bound variable.

Free variables FV of a term:

$$
\begin{aligned}
FV\ (x) &= \{x\} \\
FV\ (c) &= \{\} \\
FV\ (s\ t) &= FV(s) \cup FV(t) \\
FV\ (\lambda x.\ t) &= FV(t) \setminus \{x\}
\end{aligned}
$$

Example: $FV(\quad \lambda x.\ (\lambda y.\ (\lambda x.\ x)\ y)\ y\ x \quad) = \{y\}$

Term t is called **closed** if $FV(t) = \{\}$

Slide 21

Substitution

NICTA

$$
\begin{array}{lll}
x\ [x \leftarrow t] & = t & \\
y\ [x \leftarrow t] & = y & \text{if } x \neq y \\
c\ [x \leftarrow t] & = c &
\end{array}
$$

$$(s_1\ s_2)\ [x \leftarrow t] = (s_1[x \leftarrow t]\ s_2[x \leftarrow t])$$

$$
\begin{array}{lll}
(\lambda x.\ s)\ [x \leftarrow t] & = (\lambda x.\ s) & \\
(\lambda y.\ s)\ [x \leftarrow t] & = (\lambda y.\ s[x \leftarrow t]) & \text{if } x \neq y \text{ and } y \notin FV(t) \\
(\lambda y.\ s)\ [x \leftarrow t] & = (\lambda z.\ s[y \leftarrow z][x \leftarrow t]) & \text{if } x \neq y \\
 & & \text{and } z \notin FV(t) \cup FV(s)
\end{array}
$$

Slide 22

Substitution Example

NICTA

$$
\begin{aligned}
& (x\ \ (\lambda x.\ x)\ \ (\lambda y.\ z\ x))[x \leftarrow y] \\
= \ & (x[x \leftarrow y])\ \ ((\lambda x.\ x)[x \leftarrow y])\ \ ((\lambda y.\ z\ x)[x \leftarrow y]) \\
= \ & y\ (\lambda x.\ x)\ (\lambda y'.\ z\ y)
\end{aligned}
$$

Slide 23

α Conversion

NICTA

Bound names are irrelevant:
$\lambda x.\ x$ and $\lambda y.\ y$ denote the same function.

α conversion:
$s =_\alpha t$ means $s = t$ up to renaming of bound variables.

Formally:

$$
\begin{array}{cccccccl}
 & & & & (\lambda x.\ t) & \longrightarrow_\alpha & (\lambda y.\ t[x \leftarrow y]) & \text{if } y \notin FV(t) \\
s & \longrightarrow_\alpha & s' & \Longrightarrow & (s\ t) & \longrightarrow_\alpha & (s'\ t) & \\
t & \longrightarrow_\alpha & t' & \Longrightarrow & (s\ t) & \longrightarrow_\alpha & (s\ t') & \\
s & \longrightarrow_\alpha & s' & \Longrightarrow & (\lambda x.\ s) & \longrightarrow_\alpha & (\lambda x.\ s') &
\end{array}
$$

$s =_\alpha t$ iff $s \longrightarrow_\alpha^* t$

($\longrightarrow_\alpha^*$ = transitive, reflexive closure of $\longrightarrow_\alpha$ = multiple steps)

Slide 24

α Conversion

Equality in Isabelle is equality modulo α conversion:

if $s =_\alpha t$ then s and t are syntactically equal.

Examples:

$$
\begin{array}{ll}
 & x\ (\lambda x\ y.\ x\ y) \\
=_\alpha & x\ (\lambda y\ x.\ y\ x) \\
=_\alpha & x\ (\lambda z\ y.\ z\ y) \\
\neq_\alpha & z\ (\lambda z\ y.\ z\ y) \\
\neq_\alpha & x\ (\lambda x\ x.\ x\ x)
\end{array}
$$

Slide 25

Back to β

We have defined β reduction: $\longrightarrow_\beta$

Some notation and concepts:

- ➜ **β conversion:** $s =_\beta t$ iff $\exists n.\ s \longrightarrow_\beta^* n \wedge t \longrightarrow_\beta^* n$
- ➜ t is **reducible** if there is an s such that $t \longrightarrow_\beta s$
- ➜ $(\lambda x.\ s)\ t$ is called a **redex** (reducible expression)
- ➜ t is reducible iff it contains a redex
- ➜ if it is not reducible, t is in **normal form**

Slide 26

Does every λ term have a normal form?

No!

Example:

$$
\begin{array}{l}
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \\
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \\
(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ \longrightarrow_\beta \ldots
\end{array}
$$

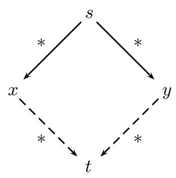
(but: $(\lambda x\ y.\ y)\ ((\lambda x.\ x\ x)\ (\lambda x.\ x\ x))\ \longrightarrow_\beta\ \lambda y.\ y$)

λ calculus is not terminating

Slide 27

β reduction is confluent

Confluence: $s \longrightarrow_\beta^* x \wedge s \longrightarrow_\beta^* y \Longrightarrow \exists t.\ x \longrightarrow_\beta^* t \wedge y \longrightarrow_\beta^* t$



Order of reduction does not matter for result
Normal forms in λ calculus are unique

Slide 28

β REDUCTION IS CONFLUENT

NICTA

Example:

$(\lambda x\ y.\ y)\ ((\lambda x.\ x\ x)\ a) \longrightarrow_\beta (\lambda x\ y.\ y)\ (a\ a) \longrightarrow_\beta \lambda y.\ y$

$(\lambda x\ y.\ y)\ ((\lambda x.\ x\ x)\ a) \longrightarrow_\beta \lambda y.\ y$

η CONVERSION

NICTA

Another case of trivially equal functions: $t = (\lambda x.\ t\ x)$

Definition:

$$\begin{array}{lllcrll} & & & & (\lambda x.\ t\ x) & \longrightarrow_\eta & t \quad \text{if } x \notin FV(t) \\ s & \longrightarrow_\eta & s' & \Longrightarrow & (s\ t) & \longrightarrow_\eta & (s'\ t) \\ t & \longrightarrow_\eta & t' & \Longrightarrow & (s\ t) & \longrightarrow_\eta & (s\ t') \\ s & \longrightarrow_\eta & s' & \Longrightarrow & (\lambda x.\ s) & \longrightarrow_\eta & (\lambda x.\ s') \end{array}$$

$$s =_\eta t \text{ iff } \exists n.\ s \longrightarrow_\eta^* n \wedge t \longrightarrow_\eta^* n$$

Example: $(\lambda x.\ f\ x)\ (\lambda y.\ g\ y) \longrightarrow_\eta (\lambda x.\ f\ x)\ g \longrightarrow_\eta f\ g$

- ➜ η reduction is confluent and terminating.
- ➜ $\longrightarrow_{\beta\eta}$ is confluent.
  $\longrightarrow_{\beta\eta}$ means $\longrightarrow_\beta$ and $\longrightarrow_\eta$ steps are both allowed.
- ➜ **Equality in Isabelle is also modulo η conversion.**

IN FACT ...

NICTA

Equality in Isabelle is modulo α, β, and η conversion.

We will see next lecture why that is possible.

EXERCISES

NICTA

- ➜ Download and install Isabelle from
  `http://isabelle.in.tum.de` or
  `http://mirror.cse.unsw.edu.au/pub/isabelle/`
- ➜ Switch on X-Symbol in ProofGeneral
- ➜ Step through the demo files from the lecture web page
- ➜ Write an own theory file, look at some theorems in the library, try 'find theorem'