NICTA

COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein
Formal Methods

$\lambda^{\rightarrow}$ and **HOL**

Slide 1

NICTA

More complex Example

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

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

Slide 2

NICTA

More general Types

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:

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

Slide 3

NICTA

Most general Types

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.

Slide 4

What about β reduction?

Definition of β reduction stays the same.

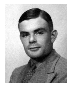
Fact: Well typed terms stay well typed during β reduction

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

This property is called **subject reduction**

What about termination?

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?

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

Types: $\tau \ ::= \ \mathsf{b} \mid {}'\nu \mid {}'\nu :: C \mid \tau \Rightarrow \tau \mid (\tau, \ldots, \tau)\ K$
$\mathsf{b} \in \{\texttt{bool}, \texttt{int}, \ldots\}$ 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: $t \ ::= \ v \mid c \mid ?v \mid (t\ t) \mid (\lambda x.\ t)$
$v, x \in V, \quad c \in C, \quad V, C$ 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.

NICTA

Type Classes

- ➜ 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** ...

NICTA

Schematic Variables

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

- ➜ 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 finished.

NICTA

Higher Order Unification

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:

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

Higher Order: schematic variables can be functions.

NICTA

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

WE HAVE LEARNED SO FAR...

NICTA

- ➜ 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 13

NICTA

PREVIEW: PROOFS IN ISABELLE

Slide 14

PROOFS IN ISABELLE

NICTA

General schema:

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

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

Slide 15

THE PROOF STATE

NICTA

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

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

Slide 16

Isabelle Theories

NICTA

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:

Slide 17

Natural Deduction Rules

NICTA

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

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

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

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

Slide 18

Proof by Assumption

NICTA

apply assumption

proves

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

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**

Slide 19

Intro Rules

NICTA

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 $A_1 \ldots A_n$

Slide 20

ELIM RULES

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

NICTA

DEMO