Getting Started With Isabelle

Lecture II: Theory Files

Lawrence C. Paulson
Computer Laboratory

UNIVERSITY OF CAMBRIDGE

Syntax Fundamentals

sorts to classify types for overloading*

types to classify terms (including polymorphism)

terms and formulas (which are just Boolean terms)

inference rules as assertions of the meta-logic

theory files to declare types, constants, etc.

proof files containing `Goal`, `by`, `qed` commands

new-style theories by Markus Wenzel (Isar)*

*not in this course

Types in Isabelle/HOL

σ `=>` τ	function types
`'a`, `'b`, ...	type variables (like in ML)
`bool`, `nat`, ...	base types
`'a list`, ...	type constructors
`(bool*nat)list`	instance of a type constructor

x `::` τ means "x has type τ"

Type `bool`*: Formulas of Higher-Order Logic*

`~`P	negation of P
P `&` Q	conjunction of P and Q
P `|` Q	disjunction of P and Q
P `-->` Q	implication between P and Q
`(`P`) = (`Q`)`	logical equivalence of P and Q
`ALL` x`.` P or `!` x`.` P	for all (universal quantifier)
`EX` x`.` P or `?` x`.` P	for some (existential quantifier)

Also conditional expressions: `if` P `then` t `else` u

Numeric Types `nat`, `int`, `real`, ...

`-`x	unary minus of x	all numerics
`+ - *`	sum, difference, product	all numerics
$\#ddd$	binary numerals	all numerics
`div mod`	quotient, remainder	types `nat`, `int`
`Suc` n	successor $n+1$	type `nat`
`0 1 2`	unary numerals	type `nat`
`< <=`	orderings	overloaded
`= ~=`	equality, non-equality	overloaded

Automatic simplification, including linear arithmetic

Lists: the Type Constructor `'a list`

`Nil`	the empty list
`Cons` x l	list with head x, tail l
xs `@` ys	append of xs, ys
`hd tl rev` ...	common list functions
`map filter` ...	common list functionals
`[`x_1`, ... ,` x_n`]`	list notation
`[`x`:`l`. P]`	nice syntax for `filter`

Sets: the Type Constructor `'a set`

`x : A`	membership, $x \in A$
`x ~: A`	non-membership, $x \notin A$
`A <= B`	subset, $A \subseteq B$
`-A`	complement of A
`A Un B`	union of A and B
`A Int B`	intersection of A and B
`ALL x:A. P`	bounded quantifier (also `EX`)
`UN x:A. P`	union of a family of sets (also `INT`)

Tupled and Curried Functions

$[\sigma_1, \ \ldots \ , \ \sigma_n]$ => τ	curried function type
$\%x_1 \ \ldots \ x_n . \ t$	curried λ-abstraction
$f \ t_1 \ldots t_n$	curried function application
σ_1 * $\cdots$ * σ_n => τ	tupled function type
$\%(x_1, \ \ldots \ , x_n). \ t$	tupled λ-abstraction
$f \ (t_1, \ldots ,t_n)$	tupled function application

Tupled abstraction allowed elsewhere:

```
ALL (x,y):edges. x ~= y
```

Constants and Variables

Name spaces resolve duplicate constant declarations

Identifiers not declared as constants can be variables

Unknowns are instantiated automatically

$T.c$	constant c declared in theory T
c	constant declared most recently
x	free variable (if not declared as a constant)
$?x$	schematic variable (unknown)

Format of a Theory File

$T = T_1 + \cdots + T_n +$

```
consts uList :: "'a => 'a list"

defs   uList_def  "uList x == [x]"
                                   (*note the == symbol!*)

rules   f_axiom   "f(f n) < f (Suc n)"

record ...

inductive ...

end
```

Extend theories $T_1, \ldots, T_n$ with constants, axioms, record declarations, etc., etc.

Further Material Provided by Isabelle/HOL

Relations — their properties and operations on them

Equivalence classes — quotients and congruences

Well-foundedness of many orderings including multisets

Cardinality including binomials and powersets

Non-standard analysis (thanks to Jacques Fleuriot)

Prime numbers — GCDs, unique factorization

Browse the Isabelle theory library on the WWW