Topics in Automated Deduction (CS 576)

Elsa L. Gunter
2112 Siebel Center
egunter@cs.uiuc.edu
http://www.cs.uiuc.edu/class/sp06/cs576/
Currying

- **Curried:** \( f :: \tau_1 \Rightarrow \tau_2 \Rightarrow \tau \)

- **Tupled:** \( f :: \tau_1 \times \tau_2 \Rightarrow \tau \)

Advantage: partial application \( f \ a_1 \) with \( a_1 :: \tau \)

**Moral:** Thou shalt curry your functions (most of the time :-))

Terms: Syntactic Sugar

Some predefined syntactic sugar:

- Infix: \(+\), \(-\), \#, \@, \ldots\)
- Mixfix: \texttt{if\_then\_else\_}, \texttt{case\_of\_}, \ldots\)
- Binders: \(\forall x. P x \) means \( (\forall)(\lambda x. P x) \)

Prefix binds more strongly than infix:

\[
\! f \ x + y \equiv (f \ x) + y \not\equiv f \ (x + y) \ !
\]
Type bool

Formulae = terms of type bool

True::bool
False::bool
¬ :: bool ⇒ bool
∧, ∨, . . . :: bool ⇒ bool

if-and-only-if: =
Type nat

0 :: nat
Suc :: nat ⇒ nat
+ , * , . . . :: nat ⇒ nat ⇒ nat
::
Overloading

! Numbers and arithmetic operations are overloaded:

0, 1, 2, . . . :: nat or real (or others)

+ :: nat ⇒ nat ⇒ nat and

+ :: real ⇒ real ⇒ real (and others)

You need type annotations: 1 :: nat, x + (y :: nat)

... unless the context is unambiguous: Suc 0
Type list

- [ ]: empty list
- \( x \# xs \): list with first element \( x \) ("head") and rest \( xs \) ("tail")
- Syntactic sugar: \([x_1, \ldots, x_n] \equiv x_1\#\ldots\#x_n\#[\]

Large library:
hd, tl, map, size, filter, set, nth, take, drop, distinct, ...

Don’t reinvent, reuse!

\[\sim HOL/List.thy\]
Theory = Module

Syntax:

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

- $MyTh$: name of theory being built. Must live in file $MyTh.thy$.
- $ImpTh_i$: name of imported theories. Importing is transitive.
Proof General

An Isabelle Interface
by David Aspinall
ProofGeneral

Customized version of (x)emacs:

- All of emacs (info: Ctrl-h i)
- Isabelle aware when editing .thy files
- (Optional) Can use mathematical symbols ("x-symbols")

Interaction:

- via mouse / buttons / pull-down menus
- or keyboard (for key bindings, see Ctrl-h m)
Input of math symbols in ProofGeneral

- via menu ("X-Symbol")
- via ascii encoding (similar to \LaTeX):
  \texttt{\textbackslash<and>, \textbackslash<or>, \ldots}

- via "standard" ascii name: \&, |, \texttt{-->, \ldots}
Symbol Translations

<table>
<thead>
<tr>
<th>x-symbol</th>
<th>∀</th>
<th>∃</th>
<th>λ</th>
<th>¬</th>
<th>∧</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii (1)</td>
<td>&lt;forall&gt;</td>
<td>&lt;exists&gt;</td>
<td>&lt;lambda&gt;</td>
<td>&lt;not&gt;</td>
<td>&lt;and&gt;</td>
</tr>
<tr>
<td>ascii (2)</td>
<td>ALL</td>
<td>EX</td>
<td>%</td>
<td>~</td>
<td>&amp;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x-symbol</th>
<th>∨</th>
<th>→</th>
<th>⇒</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii (1)</td>
<td>&lt;or&gt;</td>
<td>&lt;longrightarrow&gt;</td>
<td>&lt;Rightarrow&gt;</td>
</tr>
<tr>
<td>ascii (2)</td>
<td></td>
<td>--&gt;</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>

(1) is converted to x-symbol, (2) remains as ascii
See Appendix A of text for more complete list
Time for a demo of types and terms
A Recursive datatype

\[
\text{datatype } \ 'a\ \text{list} = \text{Nil} \mid \text{Cons } 'a \ 'a\ \text{list}
\]

\text{Nil: empty list}

\text{Cons x xs: list with head x::'a, tail xs::'a list}

A toy list: Cons False (Cons True Nil)

Syntactic sugar: \ [False, True]
When writing terms and types in `.thy` files (or an Isabelle shell):

**Types and terms need to be enclosed in "..."**

Except for single identifiers, e.g. `a`

"..." won’t always be shown on slides
Structural Induction on Lists

P xs holds for all lists xs if

- P Nil
- and for arbitrary y and ys, P ys implies P (Cons y ys)

\[
P \, ys \\
\vdash \\
P \, (\text{Cons} \, y \, ys) \\
\hline \\
P \, xs
\]
A Recursive Function: List Append

Declaration:

consts app :: "'a list ⇒ 'a list ⇒ 'a list

and definition by primitive recursion:

primrec

app Nil ys = ____

app (Cons x xs) ys = ____app xs ...____

One rule per constructor

Recursive calls only applied to constructor arguments

Guarantees termination (total function)
Demo: Append and Reverse
Proofs

General schema:

\texttt{lemma } name: " ..."

\texttt{apply ( ...)}

\texttt{::}

\texttt{done}

If the lemma is suitable as a simplification rule:

\texttt{lemma } name[simp]: " ..."

Adds lemma \texttt{name} to future simplificaitons
Top-down Proofs

**sorry**

“completes” any proof (by giving up, and accepting it)

Suitable for top-down development of theories:
Assume lemmas first, prove them later.

**Only allowed for interactive proof!**