The Language HOL

BNF-converter

January 19, 2009

This document was automatically generated by the *BNF-Converter*. It was generated together with the lexer, the parser, and the abstract syntax module, which guarantees that the document matches with the implementation of the language (provided no hand-hacking has taken place).

The lexical structure of HOL

Literals

Idents literals are recognized by the regular expression $\langle letter \rangle + \langle digit \rangle * \mid \langle digit \rangle +$

PredId literals are recognized by the regular expression $\langle upper \rangle \langle letter \rangle * \text{`('}$

FunctId literals are recognized by the regular expression $\langle lower \rangle \langle letter \rangle * \text{`('}$

Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in HOL are the following:

```
and      else    exists
false    forall  if
implies  in      let
not      or      then
true
```

The symbols used in HOL are the following:

```
)    =   .
)(   ,   ;
(
```

Comments

Single-line comments begin with $--$.
There are no multiple-line comments in the grammar.

The syntactic structure of HOL

Non-terminals are enclosed between $\langle$ and $\rangle$. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

$\langle PROGRAM \rangle$ ::= $\langle ListSENT \rangle$

$\langle SENT6 \rangle$ ::= $\langle PredId \rangle$ $\langle ListTERMS \rangle$ `)`
| `true`
| `false`
| $\langle Idents \rangle$
| $\langle TERM \rangle = \langle TERM \rangle$
| `(` $\langle SENT \rangle$ `)`

$\langle SENT5 \rangle$ ::= `not` $\langle SENT5 \rangle$
| $\langle SENT6 \rangle$

$\langle SENT4 \rangle$ ::= `if` $\langle SENT3 \rangle$ `then` $\langle SENT3 \rangle$ `else` $\langle SENT3 \rangle$
| `let` $\langle ListDEF \rangle$ `in` $\langle SENT2 \rangle$
| $\langle SENT5 \rangle$

$\langle SENT3 \rangle$ ::= $\langle SENT3 \rangle$ `and` $\langle SENT4 \rangle$
| $\langle SENT3 \rangle$ `or` $\langle SENT4 \rangle$
| $\langle SENT4 \rangle$

$\langle SENT2 \rangle$ ::= $\langle SENT2 \rangle$ `implies` $\langle SENT3 \rangle$
| $\langle SENT3 \rangle$

$\langle DEF \rangle$ ::= $\langle Idents \rangle = \langle SENT \rangle$

$\langle SENT \rangle$ ::= `forall` $\langle ListTERM \rangle$ `.` $\langle SENT \rangle$
| `exists` $\langle ListTERM \rangle$ `.` $\langle SENT \rangle$
| $\langle SENT1 \rangle$

⟨*TERM*⟩ ::= ⟨*FunctId*⟩ ⟨*ListTERMS*⟩)
| ⟨*TERM1*⟩

⟨*TERM1*⟩ ::= ⟨*Idents*⟩
| (⟨*TERM*⟩)

⟨*TERMS*⟩ ::= ⟨*ListTERM*⟩

⟨*ListTERMS*⟩ ::= ϵ
| ⟨*TERMS*⟩
| ⟨*TERMS*⟩)(⟨*ListTERMS*⟩

⟨*ListDEF*⟩ ::= ϵ
| ⟨*DEF*⟩
| ⟨*DEF*⟩ , ⟨*ListDEF*⟩

⟨*ListSENT*⟩ ::= ϵ
| ⟨*SENT*⟩ ; ⟨*ListSENT*⟩

⟨*SENT1*⟩ ::= ⟨*SENT2*⟩

⟨*ListTERM*⟩ ::= ϵ
| ⟨*TERM*⟩
| ⟨*TERM*⟩ , ⟨*ListTERM*⟩