

The Language hardtyped

BNF-converter

May 26, 2022

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 hardtyped

Identifiers

Identifiers $\langle Ident \rangle$ are unquoted strings beginning with a letter, followed by any combination of letters, digits, and the characters `_` `'`, reserved words excluded.

Literals

Double-precision float literals $\langle Double \rangle$ have the structure indicated by the regular expression $\langle digit \rangle + \cdot \langle digit \rangle + (\text{'e'-'?'}\langle digit \rangle +)?$ i.e. two sequences of digits separated by a decimal point, optionally followed by an unsigned or negative exponent.

Integer literals $\langle Int \rangle$ are nonempty sequences of digits.

String literals $\langle String \rangle$ have the form `"x"`, where *x* is any sequence of any characters except `"` unless preceded by `\`.

Bool literals are recognized by the regular expression `{"true"} | {"false"}`

Unit literals are recognized by the regular expression `{"unit"}`

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 hardtyped are the following:

```
Any    Bool    Int
Real   String  Unit
and    as      in
let    letrec  lettype
not    or
```

The symbols used in hardtyped are the following:

```
;      (      )
-|     =      :
/\     {      }
->     .      ,
|      |:     +
-      *      /
>      >=     ==
!=     <=     <
```

Comments

Single-line comments begin with `//`.

Multiple-line comments are enclosed with `/*` and `*/`.

The syntactic structure of hardtyped

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.

```
 $\langle ListExpr \rangle ::= \langle Expr \rangle ;$ 
                |  $\langle Expr \rangle ; \langle ListExpr \rangle$ 

 $\langle Expr \rangle ::= \langle Expr1 \rangle$ 
            |  $\langle String \rangle$  as  $\langle Ident \rangle$ 
            |  $\langle String \rangle$ 
            | let  $\langle VarDec \rangle = \langle Expr1 \rangle$ 
            | let  $\langle VarDec \rangle = \langle Expr1 \rangle$  in  $\langle Expr1 \rangle$ 
            | letrec  $\langle VarDec \rangle = \langle Expr1 \rangle$ 
            | letrec  $\langle VarDec \rangle = \langle Expr1 \rangle$  in  $\langle Expr1 \rangle$ 
            | lettype  $\langle VarDec \rangle = \langle Type \rangle$ 
            | lettype  $\langle VarDec \rangle = \langle Type \rangle$  in  $\langle Expr1 \rangle$ 
            |  $\langle ListIfExpr \rangle \langle ElseExpr \rangle$ 
```

$$\begin{aligned}
\langle \text{Expr1} \rangle & ::= \langle \text{Expr2} \rangle \\
& \quad | \quad /\backslash \langle \text{ListFuncArg} \rangle \{ \langle \text{ListExpr} \rangle \} \\
\langle \text{Expr2} \rangle & ::= \langle \text{Expr3} \rangle \\
& \quad | \quad /\backslash \langle \text{ListFuncArg} \rangle \{ \langle \text{ListExpr} \rangle \} (\langle \text{ListExprSequence} \rangle) \\
& \quad | \quad /\backslash \langle \text{ListFuncArg} \rangle \{ \langle \text{ListExpr} \rangle \} \rightarrow \langle \text{Type} \rangle \\
& \quad | \quad \langle \text{Op} \rangle \\
\langle \text{Expr3} \rangle & ::= \langle \text{Expr4} \rangle \\
& \quad | \quad \langle \text{Expr4} \rangle (\langle \text{ListExprSequence} \rangle) \\
& \quad | \quad \langle \text{Integer} \rangle \\
& \quad | \quad \langle \text{Double} \rangle \\
& \quad | \quad \langle \text{String} \rangle \\
& \quad | \quad \langle \text{Bool} \rangle \\
& \quad | \quad \langle \text{Unit} \rangle \\
& \quad | \quad \{ \langle \text{ListRecordElem} \rangle \} \\
\langle \text{Expr4} \rangle & ::= \langle \text{Expr5} \rangle \\
& \quad | \quad \langle \text{Ident} \rangle \\
& \quad | \quad \langle \text{Ident} \rangle \rightarrow \langle \text{Expr4} \rangle \\
& \quad | \quad \langle \text{Expr4} \rangle . \langle \text{Ident} \rangle \\
\langle \text{Expr5} \rangle & ::= (\langle \text{Expr} \rangle) \\
& \quad | \quad (\langle \text{ListExpr} \rangle) \\
\langle \text{VarDec} \rangle & ::= \langle \text{Ident} \rangle : \langle \text{Type} \rangle \\
& \quad | \quad \langle \text{Ident} \rangle \\
\langle \text{ExprSequence} \rangle & ::= \langle \text{Expr} \rangle \\
\langle \text{ListExprSequence} \rangle & ::= \epsilon \\
& \quad | \quad \langle \text{ExprSequence} \rangle \\
& \quad | \quad \langle \text{ExprSequence} \rangle , \langle \text{ListExprSequence} \rangle \\
\langle \text{FuncArg} \rangle & ::= \langle \text{Ident} \rangle : \langle \text{Type} \rangle \\
\langle \text{ListFuncArg} \rangle & ::= \langle \text{FuncArg} \rangle . \\
& \quad | \quad \langle \text{FuncArg} \rangle . \langle \text{ListFuncArg} \rangle \\
\langle \text{IfExpr} \rangle & ::= | (\langle \text{Expr2} \rangle) : \langle \text{Expr} \rangle \\
\langle \text{ListIfExpr} \rangle & ::= \langle \text{IfExpr} \rangle \\
& \quad | \quad \langle \text{IfExpr} \rangle \langle \text{ListIfExpr} \rangle \\
\langle \text{ElseExpr} \rangle & ::= | : \langle \text{Expr} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle Op \rangle & ::= \langle Op1 \rangle \\
& | \quad \langle Expr3 \rangle \text{ or } \langle Expr3 \rangle \\
\langle Op1 \rangle & ::= \langle Op2 \rangle \\
& | \quad \langle Expr3 \rangle \text{ and } \langle Expr3 \rangle \\
\langle Op2 \rangle & ::= \langle Op3 \rangle \\
& | \quad \text{not } \langle Expr3 \rangle \\
\langle Op3 \rangle & ::= \langle Op4 \rangle \\
& | \quad \langle Expr3 \rangle > \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle \geq \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle == \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle != \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle \leq \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle < \langle Expr3 \rangle \\
\langle Op4 \rangle & ::= \langle Op5 \rangle \\
& | \quad \langle Expr3 \rangle + \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle - \langle Expr3 \rangle \\
\langle Op5 \rangle & ::= \langle Op6 \rangle \\
& | \quad \langle Expr3 \rangle * \langle Expr3 \rangle \\
& | \quad \langle Expr3 \rangle / \langle Expr3 \rangle \\
\langle Op6 \rangle & ::= (\langle Op \rangle) \\
& | \quad + \langle Expr3 \rangle \\
& | \quad - \langle Expr3 \rangle \\
\langle RecordElem \rangle & ::= \langle Ident \rangle = \langle Expr3 \rangle \\
\langle ListRecordElem \rangle & ::= \epsilon \\
& | \quad \langle RecordElem \rangle \\
& | \quad \langle RecordElem \rangle , \langle ListRecordElem \rangle \\
\langle Type \rangle & ::= \langle Type1 \rangle \\
\langle Type1 \rangle & ::= \langle Type2 \rangle \\
& | \quad \langle Type1 \rangle \rightarrow \langle Type2 \rangle
\end{aligned}$$

$$\begin{aligned}
\langle Type2 \rangle &::= \langle Type3 \rangle \\
&| \quad \langle Ident \rangle \\
&| \quad Int \\
&| \quad Real \\
&| \quad Bool \\
&| \quad String \\
&| \quad Unit \\
&| \quad Any \\
&| \quad \{ \langle ListRecordElemType \rangle \} \\
\langle Type3 \rangle &::= (\langle Type \rangle) \\
\langle RecordElemType \rangle &::= \langle Ident \rangle : \langle Type \rangle \\
\langle ListRecordElemType \rangle &::= \epsilon \\
&| \quad \langle RecordElemType \rangle \\
&| \quad \langle RecordElemType \rangle , \langle ListRecordElemType \rangle
\end{aligned}$$