

The Language hardtyped

BNF-converter

May 24, 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"}`

Print literals are recognized by the regular expression `{"print"}`

ReadReal literals are recognized by the regular expression `{"readReal"}`

ReadInt literals are recognized by the regular expression `{"readInt"}`

ReadString literals are recognized by the regular expression `{"readString"}`

ReadBool literals are recognized by the regular expression `{"readBool"}`

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:

<code>Any</code>	<code>Bool</code>	<code>Int</code>
<code>Real</code>	<code>String</code>	<code>Unit</code>
<code>and</code>	<code>as</code>	<code>in</code>
<code>let</code>	<code>letrec</code>	<code>lettype</code>
<code>not</code>	<code>or</code>	

The symbols used in `hardtyped` are the following:

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

Comments

Single-line comments begin with `//`.

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

The syntactic structure of `hardtyped`

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

$$\begin{aligned} \langle ListExpr \rangle &::= \langle Expr \rangle ; \\ &| \langle Expr \rangle ; \langle ListExpr \rangle \end{aligned}$$

$$\begin{aligned}
\langle \text{Expr} \rangle &::= \langle \text{Expr1} \rangle \\
&| \quad - | \langle \text{String} \rangle \text{ as } \langle \text{Ident} \rangle \\
&| \quad - | \langle \text{String} \rangle \\
&| \quad \text{let } \langle \text{VarDec} \rangle = \langle \text{Expr1} \rangle \\
&| \quad \text{let } \langle \text{VarDec} \rangle = \langle \text{Expr1} \rangle \text{ in } \langle \text{Expr1} \rangle \\
&| \quad \text{letrec } \langle \text{VarDec} \rangle = \langle \text{Expr1} \rangle \\
&| \quad \text{lettype } \langle \text{VarDec} \rangle = \langle \text{Type} \rangle \\
&| \quad \langle \text{ListIfExpr} \rangle \langle \text{ElseExpr} \rangle \\
\langle \text{Expr1} \rangle &::= \langle \text{Expr2} \rangle \\
&| \quad / \backslash \langle \text{ListFuncArg} \rangle \{ \langle \text{ListExpr} \rangle \} \\
&| \quad \langle \text{Print} \rangle (\langle \text{Expr} \rangle) \\
&| \quad \langle \text{ReadReal} \rangle (\langle \text{ListExprSequence} \rangle) \\
&| \quad \langle \text{ReadInt} \rangle (\langle \text{ListExprSequence} \rangle) \\
&| \quad \langle \text{ReadString} \rangle (\langle \text{ListExprSequence} \rangle) \\
&| \quad \langle \text{ReadBool} \rangle (\langle \text{ListExprSequence} \rangle) \\
\langle \text{Expr2} \rangle &::= \langle \text{Expr3} \rangle \\
&| \quad / \backslash \langle \text{ListFuncArg} \rangle \{ \langle \text{ListExpr} \rangle \} (\langle \text{ListExprSequence} \rangle) \\
&| \quad / \backslash \langle \text{ListFuncArg} \rangle \{ \langle \text{ListExpr} \rangle \} \rightarrow \langle \text{Type} \rangle \\
&| \quad \langle \text{Op} \rangle \\
\langle \text{Expr3} \rangle &::= \langle \text{Expr4} \rangle \\
&| \quad \langle \text{Ident} \rangle \\
&| \quad \langle \text{VarName} \rangle (\langle \text{ListExprSequence} \rangle) \\
&| \quad \langle \text{Integer} \rangle \\
&| \quad \langle \text{Double} \rangle \\
&| \quad \langle \text{String} \rangle \\
&| \quad \langle \text{Bool} \rangle \\
&| \quad \langle \text{Unit} \rangle \\
&| \quad \{ \langle \text{ListRecord} \rangle \} \\
\langle \text{Expr4} \rangle &::= (\langle \text{Expr} \rangle) \\
&| \quad (\langle \text{ListExpr} \rangle) \\
\langle \text{VarDec} \rangle &::= \langle \text{Ident} \rangle : \langle \text{Type} \rangle \\
&| \quad \langle \text{Ident} \rangle \\
\langle \text{VarName} \rangle &::= \langle \text{ListVarPath} \rangle \langle \text{Ident} \rangle \\
\langle \text{VarPath} \rangle &::= \langle \text{Ident} \rangle \\
\langle \text{ListVarPath} \rangle &::= \epsilon \\
&| \quad \langle \text{VarPath} \rangle . \langle \text{ListVarPath} \rangle \\
\langle \text{ExprSequence} \rangle &::= \langle \text{Expr} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle \text{ListExprSequence} \rangle &::= \epsilon \\
&| \langle \text{ExprSequence} \rangle \\
&| \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 . \\
&| \langle \text{FuncArg} \rangle . \langle \text{ListFuncArg} \rangle \\
\langle \text{IfExpr} \rangle &::= | (\langle \text{Expr2} \rangle) : \langle \text{Expr} \rangle \\
\langle \text{ListIfExpr} \rangle &::= \epsilon \\
&| \langle \text{IfExpr} \rangle \langle \text{ListIfExpr} \rangle \\
\langle \text{ElseExpr} \rangle &::= | : \langle \text{Expr} \rangle \\
\langle \text{Op} \rangle &::= \langle \text{Op1} \rangle \\
&| \langle \text{Expr3} \rangle \text{ or } \langle \text{Expr3} \rangle \\
\langle \text{Op1} \rangle &::= \langle \text{Op2} \rangle \\
&| \langle \text{Expr3} \rangle \text{ and } \langle \text{Expr3} \rangle \\
\langle \text{Op2} \rangle &::= \langle \text{Op3} \rangle \\
&| \text{not } \langle \text{Expr3} \rangle \\
\langle \text{Op3} \rangle &::= \langle \text{Op4} \rangle \\
&| \langle \text{Expr3} \rangle > \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle \geq \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle == \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle != \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle \leq \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle < \langle \text{Expr3} \rangle \\
\langle \text{Op4} \rangle &::= \langle \text{Op5} \rangle \\
&| \langle \text{Expr3} \rangle + \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle - \langle \text{Expr3} \rangle \\
\langle \text{Op5} \rangle &::= \langle \text{Op6} \rangle \\
&| \langle \text{Expr3} \rangle * \langle \text{Expr3} \rangle \\
&| \langle \text{Expr3} \rangle / \langle \text{Expr3} \rangle \\
\langle \text{Op6} \rangle &::= (\langle \text{Op} \rangle) \\
&| + \langle \text{Expr3} \rangle \\
&| - \langle \text{Expr3} \rangle \\
\langle \text{Type} \rangle &::= \langle \text{Type1} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle Type1 \rangle & ::= \langle Type2 \rangle \\
& | \quad \langle Type1 \rangle -> \langle Type2 \rangle \\
\langle Type2 \rangle & ::= \langle Type3 \rangle \\
& | \quad \text{Int} \\
& | \quad \text{Real} \\
& | \quad \text{Bool} \\
& | \quad \text{String} \\
& | \quad \text{Unit} \\
& | \quad \text{Any} \\
& | \quad \{ \langle ListRecord \rangle \} \\
\langle Type3 \rangle & ::= (\langle Type \rangle) \\
\langle Record \rangle & ::= \langle VarDec \rangle = \langle Expr3 \rangle \\
& | \quad \langle VarDec \rangle \\
\langle ListRecord \rangle & ::= \epsilon \\
& | \quad \langle Record \rangle \\
& | \quad \langle Record \rangle , \langle ListRecord \rangle
\end{aligned}$$