The Language hardtyped

BNF-converter

May 23, 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 ⟨*Ident*⟩ 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 ⟨*Double*⟩ have the structure indicated by the regular expression ⟨*digit*⟩ + '.'⟨*digit*⟩ + ('e'-'?⟨*digit*⟩+)? i.e. two sequences of digits separated by a decimal point, optionally followed by an unsigned or negative exponent.

Integer literals ⟨*Int*⟩ are nonempty sequences of digits.

String literals ⟨*String*⟩ 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"}

Read literals are recognized by the regular expression {"read"}

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 sym-

bols, 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:

```
Boolean   Integer   Real
String    Unit      and
as        in        let
letrec    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 ⟨ and ⟩. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

$$
\begin{array}{lll}
\langle ListExpr\rangle & ::= & \epsilon \\
& | & \langle Expr\rangle \\
& | & \langle Expr\rangle \; ; \; \langle ListExpr\rangle
\end{array}
$$

$$\langle Expr\rangle \quad ::= \quad \text{/\textbackslash} \ \langle FuncArg\rangle \ \texttt{\{} \ \langle InExpr\rangle \ \texttt{\}}$$

$\langle Expr\rangle$	$::=$	$\text{/\textbackslash}\ \langle FuncArg\rangle\ \texttt{\{}\ \langle InExpr\rangle\ \texttt{\}}$
	$\mid$	$\langle Expr\rangle\ \texttt{(}\ \langle Expr\rangle\ \texttt{)}$
	$\mid$	$\langle Print\rangle\ \texttt{(}\ \langle Expr\rangle\ \texttt{)}$
	$\mid$	$\langle Read\rangle\ \texttt{(}\ \langle Expr\rangle\ \texttt{)}$
	$\mid$	$\langle Print\rangle\ \texttt{(}\ \texttt{)}$
	$\mid$	$\langle Read\rangle\ \texttt{(}\ \texttt{)}$
	$\mid$	$\langle Expr\rangle\ \texttt{,}\ \langle Expr\rangle$
	$\mid$	$\texttt{|}\ \texttt{(}\ \langle Expr\rangle\ \texttt{)}\ \texttt{:}\ \langle Expr\rangle$
	$\mid$	$\texttt{|:}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{+}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{-}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{*}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{/}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{and}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{or}\ \langle Expr\rangle$
	$\mid$	$\texttt{not}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{>}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{>=}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{==}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{<=}\ \langle Expr\rangle$
	$\mid$	$\langle Expr\rangle\ \texttt{<}\ \langle Expr\rangle$
	$\mid$	$\texttt{+}\ \langle Expr\rangle$
	$\mid$	$\texttt{-}\ \langle Expr\rangle$
	$\mid$	$\texttt{let}\ \langle VarDec\rangle\ \texttt{=}\ \langle Expr\rangle$
	$\mid$	$\texttt{let}\ \langle VarDec\rangle\ \texttt{=}\ \langle Expr\rangle\ \texttt{in}\ \langle Expr\rangle$
	$\mid$	$\texttt{let}\ \langle VarDec\rangle\ \texttt{=}\ \langle Expr\rangle\ \texttt{as}\ \langle Type\rangle$
	$\mid$	$\texttt{letrec}\ \langle VarDec\rangle\ \texttt{=}\ \langle Expr\rangle$
	$\mid$	$\langle Integer\rangle$
	$\mid$	$\langle Double\rangle$
	$\mid$	$\langle String\rangle$
	$\mid$	$\langle Bool\rangle$
	$\mid$	$\langle Unit\rangle$
	$\mid$	$\langle Ident\rangle$
	$\mid$	$\texttt{(}\ \langle Expr\rangle\ \texttt{)}$
	$\mid$	$\langle Ident\rangle\ \texttt{.}\ \langle Ident\rangle$
$\langle InExpr\rangle$	$::=$	$\langle InExpr\rangle\ \texttt{;}\ \langle InExpr\rangle$
	$\mid$	$\langle InExpr\rangle\ \texttt{;}$
	$\mid$	$\langle Expr\rangle$
$\langle FuncArg\rangle$	$::=$	$\langle VarDec\rangle\ \texttt{.}\ \langle FuncArg\rangle$
	$\mid$	$\langle VarDec\rangle\ \texttt{.}$
$\langle VarDec\rangle$	$::=$	$\langle Ident\rangle\ \texttt{:}\ \langle Type\rangle$
	$\mid$	$\langle Ident\rangle$

```
⟨Type⟩   ::=   Integer
          |     Real
          |     Boolean
          |     String
          |     Unit
```