The Language SW

BNF-converter

June 7, 2021

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 SW

Literals

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 `\`.

Numvar literals are recognized by the regular expression '%'$\langle letter \rangle$($\langle letter \rangle$ | $\langle digit \rangle$)*

Stringvar literals are recognized by the regular expression '\$'$\langle letter \rangle$($\langle letter \rangle$ | $\langle digit \rangle$)*

Envar literals are recognized by the regular expression '\$'`'_'`$\langle letter \rangle$($\langle letter \rangle$ | $\langle digit \rangle$)*

Symvar literals are recognized by the regular expression '&'$\langle letter \rangle$($\langle letter \rangle$ | $\langle digit \rangle$ | '_')*

SubId literals are recognized by the regular expression 'ˆ'$\langle letter \rangle$($\langle letter \rangle$ | $\langle digit \rangle$ | '_')*

Id literals are recognized by the regular expression $\langle letter \rangle$($\langle letter \rangle$ | $\langle digit \rangle$ | '_')*

ValidImport literals are recognized by the regular expression '{'($\langle letter \rangle$ | $\langle digit \rangle$ | '_' | '.' | '/') * '}'

Date literals are recognized by the regular expression $\langle digit \rangle\langle digit \rangle\langle digit \rangle\langle digit \rangle$('−'$\langle digit \rangle\langle digit \rangle$)* 'T'(':' | $\langle digit \rangle$ | '−')*

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 SW are the following:

There are no reserved words in SW.

The symbols used in SW are the following:

```
;               {       }
<               −       >
_               (       )
.               /       =
StreamWork:   − − −    :
```

Comments

Single-line comments begin with `#`.
Multiple-line comments are enclosed with `{#` and `#}`.

The syntactic structure of SW

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.

$$
\begin{array}{lll}
\langle \textit{Valide} \rangle & ::= & \langle \textit{ValidConfig} \rangle \\
& | & \langle \textit{ValidSW} \rangle \\
\\
\langle \textit{ValidSW} \rangle & ::= & \langle \textit{ListStm} \rangle \\
\\
\langle \textit{Stm} \rangle & ::= & \langle \textit{DataFlow} \rangle \\
& | & \langle \textit{Numassgn} \rangle \\
& | & \langle \textit{Strassgn} \rangle \\
& | & \langle \textit{SymAssgn} \rangle \\
& | & \langle \textit{Hermt} \rangle \\
& | & \langle \textit{Subdef} \rangle
\end{array}
$$

$\langle ListStm \rangle$ ::= ϵ
        | $\langle Stm \rangle$; $\langle ListStm \rangle$

$\langle Subdef \rangle$ ::= $\langle SubId \rangle$ { $\langle ListSubnet \rangle$ }

$\langle Subnet \rangle$ ::= $\langle Hermt \rangle$
        | $\langle DataFlow \rangle$
        | $\langle ExtPortIn \rangle$
        | $\langle ExtPortOut \rangle$

$\langle ListSubnet \rangle$ ::= ϵ
        | $\langle Subnet \rangle$; $\langle ListSubnet \rangle$

$\langle ExtPortIn \rangle$ ::= $\langle Proc \rangle$ $\langle Prt \rangle$ $\langle Larrow \rangle$ $\langle Tab \rangle$
        | $\langle Tab \rangle$ $\langle Rarrow \rangle$ $\langle Prt \rangle$ $\langle Proc \rangle$

$\langle ExtPortOut \rangle$ ::= $\langle Tab \rangle$ $\langle Larrow \rangle$ $\langle Prt \rangle$ $\langle Proc \rangle$
        | $\langle Proc \rangle$ $\langle Prt \rangle$ $\langle Rarrow \rangle$ $\langle Tab \rangle$

$\langle Tab \rangle$ ::= $\langle Numval \rangle$
        | $\langle Symval \rangle$

$\langle DataFlow \rangle$ ::= $\langle Proc \rangle$ $\langle Prt \rangle$ $\langle Larrow \rangle$ $\langle Prt \rangle$ $\langle Proc \rangle$
        | $\langle Proc \rangle$ $\langle Prt \rangle$ $\langle Rarrow \rangle$ $\langle Prt \rangle$ $\langle Proc \rangle$
        | $\langle DataFlow \rangle$ $\langle Prt \rangle$ $\langle Larrow \rangle$ $\langle Prt \rangle$ $\langle Proc \rangle$
        | $\langle DataFlow \rangle$ $\langle Prt \rangle$ $\langle Rarrow \rangle$ $\langle Prt \rangle$ $\langle Proc \rangle$

$\langle Larrow \rangle$ ::= < $\langle TypeDef \rangle$ $\langle Buffsize \rangle$ −

$\langle Rarrow \rangle$ ::= − $\langle TypeDef \rangle$ $\langle Buffsize \rangle$ >

$\langle TypeDef \rangle$ ::= $\langle Symvalu \rangle$
        | ϵ

$\langle Buffsize \rangle$ ::= $\langle Numval \rangle$
        | ϵ

$\langle Hermt \rangle$ ::= $\langle Symvalu \rangle$ $\langle Comp \rangle$ $\langle ListArgument \rangle$
        | $\langle Symvalu \rangle$ $\langle ListArgument \rangle$

$\langle Symvalu \rangle$ ::= $\langle Symval \rangle$
        | _

$\langle Proc \rangle$ ::= ($\langle Symvalu \rangle$ $\langle Comp \rangle$ $\langle ListArgument \rangle$)
        | ($\langle Symvalu \rangle$ $\langle ListArgument \rangle$)

$\langle Prt \rangle$ ::= $\langle Numval \rangle$
  | $\langle Numval \rangle$. $\langle Symval \rangle$
  | $\langle Symval \rangle$. $\langle Numval \rangle$
  | $\langle Symval \rangle$
  | ϵ

$\langle Comp \rangle$ ::= $\langle Symval \rangle$
  | $\langle SubId \rangle$
  | $\langle ModPath \rangle \, \langle Symval \rangle$
  | $\langle RemPath \rangle$

$\langle ModPath \rangle$ ::= / $\langle Symval \rangle$ /
  | $\langle Symval \rangle$ /
  | $\langle ModPath \rangle \, \langle Symval \rangle$ /

$\langle RemPath \rangle$ ::= $\langle ValidImport \rangle \, \langle Symval \rangle$

$\langle Argument \rangle$ ::= $\langle Stringval \rangle$

$\langle ListArgument \rangle$ ::= ϵ
  | $\langle Argument \rangle \, \langle ListArgument \rangle$

$\langle Numassgn \rangle$ ::= $\langle Numvar \rangle = \langle Numval \rangle$

$\langle Strassgn \rangle$ ::= $\langle Stringvar \rangle = \langle Symval \rangle$

$\langle SymAssgn \rangle$ ::= $\langle Symvar \rangle = \langle Symval \rangle$

$\langle Numval \rangle$ ::= $\langle Integer \rangle$
  | $\langle Numvar \rangle$

$\langle Stringval \rangle$ ::= $\langle String \rangle$
  | $\langle Stringvar \rangle$
  | $\langle Envar \rangle$

$\langle Symval \rangle$ ::= $\langle Symvar \rangle$
  | $\langle Id \rangle$
  | $\langle Envar \rangle$

$\langle ValidConfig \rangle$ ::= `StreamWork:` $\langle ListEntry \rangle$
  | $---$ $\langle ListEntry \rangle$

$\langle Entry \rangle$ ::= $\langle KeyVal \rangle$
  | $\langle KeyName \rangle$

$\langle ListEntry \rangle$ ::= ϵ
  | $\langle Entry \rangle \, \langle ListEntry \rangle$

$$\begin{array}{rll} \langle KeyVal\rangle & ::= & \langle KeyName\rangle\ \langle Integer\rangle \\ & | & \langle KeyName\rangle\ \langle String\rangle \\ & | & \langle KeyName\rangle\ \langle Date\rangle \\[4pt] \langle KeyName\rangle & ::= & \langle Symval\rangle\ : \\ & | & \langle ModPath\rangle\ \langle Symval\rangle\ : \end{array}$$