The Language chi

BNF-converter

November 22, 2008

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 chi

Literals

UIdent literals are recognized by the regular expression ⟨*upper*⟩(⟨*letter*⟩ | ⟨*digit*⟩ | '−')∗

LIdent literals are recognized by the regular expression ⟨*lower*⟩(⟨*letter*⟩ | ⟨*digit*⟩ | '−')∗

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 chi are the following:

```
case  in   let
of    rec
```

The symbols used in chi are the following:

```
\   .   {
}   ;   =
(   )   ->
```

Comments

Single-line comments begin with $--$.
Multiple-line comments are enclosed with $\{-$ and $-\}$.

The syntactic structure of chi

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.

$\langle Exp \rangle$ ::= \ $\langle ListLIdent \rangle$. $\langle Exp \rangle$
| `case` $\langle Exp \rangle$ `of` { $\langle ListBranch \rangle$ }
| `rec` $\langle LIdent \rangle$ = $\langle Exp \rangle$
| `let` { $\langle ListDef \rangle$ } `in` $\langle Exp \rangle$
| $\langle Exp1 \rangle$

$\langle ListLIdent \rangle$ ::= ϵ
| $\langle LIdent \rangle$ $\langle ListLIdent \rangle$

$\langle Exp1 \rangle$ ::= $\langle Exp \rangle$ $\langle ListExp3 \rangle$
| $\langle Exp2 \rangle$

$\langle ListExp3 \rangle$ ::= $\langle Exp3 \rangle$
| $\langle Exp3 \rangle$ $\langle ListExp3 \rangle$

$\langle ListBranch \rangle$ ::= $\langle Branch \rangle$
| $\langle Branch \rangle$; $\langle ListBranch \rangle$

$\langle ListDef \rangle$ ::= $\langle Def \rangle$
| $\langle Def \rangle$; $\langle ListDef \rangle$

$\langle Exp3 \rangle$ ::= $\langle LIdent \rangle$
| $\langle UIdent \rangle$
| ($\langle Exp \rangle$)

$\langle Exp2 \rangle$ ::= $\langle Exp3 \rangle$

$\langle Branch \rangle$ ::= $\langle UIdent \rangle$ -> $\langle Exp \rangle$

$\langle Def \rangle$ ::= $\langle LIdent \rangle$ = $\langle Exp \rangle$