The Language GFC

BNF-converter

December 2, 2005

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 GFC

Identifiers

Identifiers ⟨*Ident*⟩ are unquoted strings beginning with a letter, followed by any combination of letters, digits, and the characters `_ '`, reserved words excluded.

Literals

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

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

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.

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 GFC are the following:

```
Ints      Str    Type
abstract  cat    concrete
data      flags  fun
grammar   in     lin
lincat    of     open
oper      param  pre
resource  table  transfer
variants
```

The symbols used in GFC are the following:

```
;    =   {
}    :   ->
**   [   ]
\    .   (
)    _   <
>    $   ?
=>   !   ++
/    @   +
|    ,
```

Comments

There are no single-line comments in the grammar.
There are no multiple-line comments in the grammar.

The syntactic structure of GFC

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

⟨*Canon*⟩ ::= `grammar` ⟨*ListIdent*⟩ `of` ⟨*Ident*⟩ `;` ⟨*ListModule*⟩
| ⟨*ListModule*⟩

⟨*Line*⟩ ::= `grammar` ⟨*ListIdent*⟩ `of` ⟨*Ident*⟩ `;`
| ⟨*ModType*⟩ `=` ⟨*Extend*⟩ ⟨*Open*⟩ `{`
| ⟨*Flag*⟩ `;`
| ⟨*Def*⟩ `;`
| `}`

⟨*Module*⟩ ::= ⟨*ModType*⟩ `=` ⟨*Extend*⟩ ⟨*Open*⟩ `{` ⟨*ListFlag*⟩ ⟨*ListDef*⟩ `}`

⟨*ModType*⟩ ::= `abstract` ⟨*Ident*⟩
| `concrete` ⟨*Ident*⟩ `of` ⟨*Ident*⟩
| `resource` ⟨*Ident*⟩
| `transfer` ⟨*Ident*⟩ `:` ⟨*Ident*⟩ `->` ⟨*Ident*⟩

⟨*ListModule*⟩ ::= ϵ
| ⟨*Module*⟩ ⟨*ListModule*⟩

⟨*Extend*⟩ ::= ⟨*ListIdent*⟩ `**`
| ϵ

⟨*Open*⟩ ::= `open` ⟨*ListIdent*⟩ `in`
| ϵ

⟨*Flag*⟩ ::= `flags` ⟨*Ident*⟩ `=` ⟨*Ident*⟩

⟨*Def*⟩ ::= `cat` ⟨*Ident*⟩ `[` ⟨*ListDecl*⟩ `]` `=` ⟨*ListCIdent*⟩
| `fun` ⟨*Ident*⟩ `:` ⟨*Exp*⟩ `=` ⟨*Exp*⟩
| `transfer` ⟨*Ident*⟩ `=` ⟨*Exp*⟩
| `param` ⟨*Ident*⟩ `=` ⟨*ListParDef*⟩
| `oper` ⟨*Ident*⟩ `:` ⟨*CType*⟩ `=` ⟨*Term*⟩
| `lincat` ⟨*Ident*⟩ `=` ⟨*CType*⟩ `=` ⟨*Term*⟩ `;` ⟨*Term*⟩
| `lin` ⟨*Ident*⟩ `:` ⟨*CIdent*⟩ `=` `\` ⟨*ListArgVar*⟩ `->` ⟨*Term*⟩ `;` ⟨*Term*⟩
| ⟨*Ident*⟩ ⟨*Status*⟩ `in` ⟨*Ident*⟩

⟨*ParDef*⟩ ::= ⟨*Ident*⟩ ⟨*ListCType*⟩

⟨*Status*⟩ ::= `data`
| ϵ

⟨*CIdent*⟩ ::= ⟨*Ident*⟩ `.` ⟨*Ident*⟩

⟨*Exp1*⟩ ::= ⟨*Exp1*⟩ ⟨*Exp2*⟩
| ⟨*Exp2*⟩

⟨*Exp*⟩ ::= `(` ⟨*Ident*⟩ `:` ⟨*Exp*⟩ `)` `->` ⟨*Exp*⟩
| `\` ⟨*Ident*⟩ `->` ⟨*Exp*⟩
| `{` ⟨*ListEquation*⟩ `}`
| ⟨*Exp1*⟩

⟨*Exp2*⟩ ::= ⟨*Atom*⟩
| `data`
| `(` ⟨*Exp*⟩ `)`

⟨*Sort*⟩ ::= `Type`

⟨*Equation*⟩ ::= ⟨*ListAPatt*⟩ `->` ⟨*Exp*⟩

⟨*APatt*⟩ ::= `(` ⟨*CIdent*⟩ ⟨*ListAPatt*⟩ `)`
| ⟨*Ident*⟩
| ⟨*String*⟩
| ⟨*Integer*⟩
| ⟨*Double*⟩
| `_`

⟨*ListDecl*⟩ ::= ϵ
| ⟨*Decl*⟩
| ⟨*Decl*⟩ `;` ⟨*ListDecl*⟩

⟨*ListAPatt*⟩ ::= ϵ
| ⟨*APatt*⟩ ⟨*ListAPatt*⟩

⟨*ListEquation*⟩ ::= ϵ
| ⟨*Equation*⟩ `;` ⟨*ListEquation*⟩

⟨*Atom*⟩ ::= ⟨*CIdent*⟩
| `<` ⟨*CIdent*⟩ `>`
| `$` ⟨*Ident*⟩
| `?` ⟨*Integer*⟩
| ⟨*String*⟩
| ⟨*Integer*⟩
| ⟨*Sort*⟩

⟨*Decl*⟩ ::= ⟨*Ident*⟩ `:` ⟨*Exp*⟩

⟨*CType*⟩ ::= `{` ⟨*ListLabelling*⟩ `}`
| `(` ⟨*CType*⟩ `=>` ⟨*CType*⟩ `)`
| ⟨*CIdent*⟩
| `Str`
| `Ints` ⟨*Integer*⟩

⟨*Labelling*⟩ ::= ⟨*Label*⟩ `:` ⟨*CType*⟩

⟨*Term2*⟩ ::= ⟨*ArgVar*⟩
| ⟨*CIdent*⟩
| `<` ⟨*CIdent*⟩ ⟨*ListTerm2*⟩ `>`
| `$` ⟨*Ident*⟩
| `{` ⟨*ListAssign*⟩ `}`
| ⟨*Integer*⟩
| ⟨*Double*⟩
| ⟨*Tokn*⟩
| `[]`
| `(` ⟨*Term*⟩ `)`

⟨*Term1*⟩ ::= ⟨*Term2*⟩ `.` ⟨*Label*⟩
| `table` ⟨*CType*⟩ `{` ⟨*ListCase*⟩ `}`
| `table` ⟨*CType*⟩ `[` ⟨*ListTerm2*⟩ `]`
| ⟨*Term1*⟩ `!` ⟨*Term2*⟩
| `variants` `{` ⟨*ListTerm2*⟩ `}`
| ⟨*Term2*⟩

⟨*Term*⟩ ::= ⟨*Term*⟩ `++` ⟨*Term1*⟩
| ⟨*Term1*⟩

⟨*Tokn*⟩ ::= ⟨*String*⟩
| `[ pre` ⟨*ListString*⟩ `{` ⟨*ListVariant*⟩ `} ]`

⟨*Assign*⟩ ::= ⟨*Label*⟩ `=` ⟨*Term*⟩

⟨*Case*⟩ ::= ⟨*ListPatt*⟩ `=>` ⟨*Term*⟩

⟨*Variant*⟩ ::= ⟨*ListString*⟩ `/` ⟨*ListString*⟩

⟨*Label*⟩ ::= ⟨*Ident*⟩
| `$` ⟨*Integer*⟩

⟨*ArgVar*⟩ ::= ⟨*Ident*⟩ `@` ⟨*Integer*⟩
| ⟨*Ident*⟩ `+` ⟨*Integer*⟩ `@` ⟨*Integer*⟩

⟨*Patt*⟩ ::= `(` ⟨*CIdent*⟩ ⟨*ListPatt*⟩ `)`
| ⟨*Ident*⟩
| `_`
| `{` ⟨*ListPattAssign*⟩ `}`
| ⟨*Integer*⟩
| ⟨*Double*⟩

⟨*PattAssign*⟩ ::= ⟨*Label*⟩ `=` ⟨*Patt*⟩

⟨*ListFlag*⟩ ::= ϵ
| ⟨*Flag*⟩ `;` ⟨*ListFlag*⟩

⟨*ListDef*⟩ ::= ϵ
| ⟨*Def*⟩ `;` ⟨*ListDef*⟩

⟨*ListParDef*⟩ ::= ϵ
| ⟨*ParDef*⟩
| ⟨*ParDef*⟩ `|` ⟨*ListParDef*⟩

⟨*ListCType*⟩ ::= ϵ
| ⟨*CType*⟩ ⟨*ListCType*⟩

⟨*ListCIdent*⟩ ::= ϵ
| ⟨*CIdent*⟩ ⟨*ListCIdent*⟩

⟨*ListAssign*⟩ ::= ϵ
| ⟨*Assign*⟩
| ⟨*Assign*⟩ ; ⟨*ListAssign*⟩

⟨*ListArgVar*⟩ ::= ϵ
| ⟨*ArgVar*⟩
| ⟨*ArgVar*⟩ , ⟨*ListArgVar*⟩

⟨*ListLabelling*⟩ ::= ϵ
| ⟨*Labelling*⟩
| ⟨*Labelling*⟩ ; ⟨*ListLabelling*⟩

⟨*ListCase*⟩ ::= ϵ
| ⟨*Case*⟩
| ⟨*Case*⟩ ; ⟨*ListCase*⟩

⟨*ListTerm2*⟩ ::= ϵ
| ⟨*Term2*⟩ ⟨*ListTerm2*⟩

⟨*ListString*⟩ ::= ϵ
| ⟨*String*⟩ ⟨*ListString*⟩

⟨*ListVariant*⟩ ::= ϵ
| ⟨*Variant*⟩
| ⟨*Variant*⟩ ; ⟨*ListVariant*⟩

⟨*ListPattAssign*⟩ ::= ϵ
| ⟨*PattAssign*⟩
| ⟨*PattAssign*⟩ ; ⟨*ListPattAssign*⟩

⟨*ListPatt*⟩ ::= ϵ
| ⟨*Patt*⟩ ⟨*ListPatt*⟩

⟨*ListIdent*⟩ ::= ϵ
| ⟨*Ident*⟩
| ⟨*Ident*⟩ , ⟨*ListIdent*⟩