The Language GF

BNF-converter

May 26, 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 GF

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

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 \.

LString literals are recognized by the regular expression $‘”(\langle anychar \rangle - ‘”)*‘”$

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 GF are the following:

```
Lin         PType      Str
Strs        Tok        Type
abstract    case       cat
concrete    data       def
flags       fn         fun
grammar     in         include
incomplete  instance   interface
let         lin        lincat
lindef      lintype    of
open        oper       out
package     param      pattern
pre         printname  resource
reuse       strs       table
tokenizer   transfer   union
var         variants   where
with
```

The symbols used in GF are the following:

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

Comments

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

The syntactic structure of GF

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

⟨*Grammar*⟩   ::=   ⟨*ListModDef*⟩

$\langle ListModDef \rangle$ ::= ϵ
       | $\langle ModDef \rangle$ $\langle ListModDef \rangle$

$\langle ModDef \rangle$ ::= $\langle ModDef \rangle$;
       | `grammar` $\langle Ident \rangle$ = { `abstract` = $\langle Ident \rangle$; $\langle ListConcSpec \rangle$ }
       | $\langle ComplMod \rangle$ $\langle ModType \rangle$ = $\langle ModBody \rangle$

$\langle ConcSpec \rangle$ ::= $\langle Ident \rangle$ = $\langle ConcExp \rangle$

$\langle ListConcSpec \rangle$ ::= ϵ
       | $\langle ConcSpec \rangle$
       | $\langle ConcSpec \rangle$; $\langle ListConcSpec \rangle$

$\langle ConcExp \rangle$ ::= $\langle Ident \rangle$ $\langle ListTransfer \rangle$

$\langle ListTransfer \rangle$ ::= ϵ
       | $\langle Transfer \rangle$ $\langle ListTransfer \rangle$

$\langle Transfer \rangle$ ::=   (`transfer in` $\langle Open \rangle$)
       | (`transfer out` $\langle Open \rangle$)

$\langle ModType \rangle$ ::=   `abstract` $\langle Ident \rangle$
       | `resource` $\langle Ident \rangle$
       | `interface` $\langle Ident \rangle$
       | `concrete` $\langle Ident \rangle$ `of` $\langle Ident \rangle$
       | `instance` $\langle Ident \rangle$ `of` $\langle Ident \rangle$
       | `transfer` $\langle Ident \rangle$: $\langle Open \rangle$ $->$ $\langle Open \rangle$

$\langle ModBody \rangle$ ::= $\langle Extend \rangle$ $\langle Opens \rangle$ { $\langle ListTopDef \rangle$ }
       | $\langle Ident \rangle$ `with` $\langle ListOpen \rangle$
       | $\langle ListIncluded \rangle$ `**` $\langle Ident \rangle$ `with` $\langle ListOpen \rangle$
       | `reuse` $\langle Ident \rangle$
       | `union` $\langle ListIncluded \rangle$

$\langle ListTopDef \rangle$ ::= ϵ
       | $\langle TopDef \rangle$ $\langle ListTopDef \rangle$

$\langle Extend \rangle$ ::= $\langle ListIncluded \rangle$ `**`
       | ϵ

$\langle ListOpen \rangle$ ::= ϵ
       | $\langle Open \rangle$
       | $\langle Open \rangle$, $\langle ListOpen \rangle$

$\langle Opens \rangle$ ::= ϵ
       | `open` $\langle ListOpen \rangle$ `in`

⟨Open⟩   ::=   ⟨Ident⟩
         | (⟨QualOpen⟩ ⟨Ident⟩)
         | (⟨QualOpen⟩ ⟨Ident⟩ = ⟨Ident⟩)

⟨ComplMod⟩   ::=   ε
             | incomplete

⟨QualOpen⟩   ::=   ε
             | incomplete
             | interface

⟨ListIncluded⟩   ::=   ε
                 | ⟨Included⟩
                 | ⟨Included⟩ , ⟨ListIncluded⟩

⟨Included⟩   ::=   ⟨Ident⟩
             | ⟨Ident⟩ [⟨ListIdent⟩]
             | ⟨Ident⟩ − [⟨ListIdent⟩]

⟨Def⟩   ::=   ⟨ListName⟩ : ⟨Exp⟩
        | ⟨ListName⟩ = ⟨Exp⟩
        | ⟨Name⟩ ⟨ListPatt⟩ = ⟨Exp⟩
        | ⟨ListName⟩ : ⟨Exp⟩ = ⟨Exp⟩

⟨TopDef⟩   ::=   cat ⟨ListCatDef⟩
           | fun ⟨ListFunDef⟩
           | data ⟨ListFunDef⟩
           | def ⟨ListDef⟩
           | data ⟨ListDataDef⟩
           | transfer ⟨ListDef⟩
           | param ⟨ListParDef⟩
           | oper ⟨ListDef⟩
           | lincat ⟨ListPrintDef⟩
           | lindef ⟨ListDef⟩
           | lin ⟨ListDef⟩
           | printname cat ⟨ListPrintDef⟩
           | printname fun ⟨ListPrintDef⟩
           | flags ⟨ListFlagDef⟩
           | printname ⟨ListPrintDef⟩
           | lintype ⟨ListDef⟩
           | pattern ⟨ListDef⟩
           | package ⟨Ident⟩ = { ⟨ListTopDef⟩ } ;
           | var ⟨ListDef⟩
           | tokenizer ⟨Ident⟩ ;

⟨CatDef⟩   ::=   ⟨Ident⟩ ⟨ListDDecl⟩
           | [⟨Ident⟩ ⟨ListDDecl⟩]
           | [⟨Ident⟩ ⟨ListDDecl⟩] { ⟨Integer⟩ }

$\langle\mathit{FunDef}\rangle$ ::= $\langle\mathit{ListIdent}\rangle$: $\langle\mathit{Exp}\rangle$

$\langle\mathit{DataDef}\rangle$ ::= $\langle\mathit{Ident}\rangle$ = $\langle\mathit{ListDataConstr}\rangle$

$\langle\mathit{DataConstr}\rangle$ ::= $\langle\mathit{Ident}\rangle$
         | $\langle\mathit{Ident}\rangle$. $\langle\mathit{Ident}\rangle$

$\langle\mathit{ListDataConstr}\rangle$ ::= ϵ
         | $\langle\mathit{DataConstr}\rangle$
         | $\langle\mathit{DataConstr}\rangle$ | $\langle\mathit{ListDataConstr}\rangle$

$\langle\mathit{ParDef}\rangle$ ::= $\langle\mathit{Ident}\rangle$ = $\langle\mathit{ListParConstr}\rangle$
         | $\langle\mathit{Ident}\rangle$ = (in $\langle\mathit{Ident}\rangle$)
         | $\langle\mathit{Ident}\rangle$

$\langle\mathit{ParConstr}\rangle$ ::= $\langle\mathit{Ident}\rangle$ $\langle\mathit{ListDDecl}\rangle$

$\langle\mathit{PrintDef}\rangle$ ::= $\langle\mathit{ListName}\rangle$ = $\langle\mathit{Exp}\rangle$

$\langle\mathit{FlagDef}\rangle$ ::= $\langle\mathit{Ident}\rangle$ = $\langle\mathit{Ident}\rangle$

$\langle\mathit{ListDef}\rangle$ ::= $\langle\mathit{Def}\rangle$;
         | $\langle\mathit{Def}\rangle$; $\langle\mathit{ListDef}\rangle$

$\langle\mathit{ListCatDef}\rangle$ ::= $\langle\mathit{CatDef}\rangle$;
         | $\langle\mathit{CatDef}\rangle$; $\langle\mathit{ListCatDef}\rangle$

$\langle\mathit{ListFunDef}\rangle$ ::= $\langle\mathit{FunDef}\rangle$;
         | $\langle\mathit{FunDef}\rangle$; $\langle\mathit{ListFunDef}\rangle$

$\langle\mathit{ListDataDef}\rangle$ ::= $\langle\mathit{DataDef}\rangle$;
         | $\langle\mathit{DataDef}\rangle$; $\langle\mathit{ListDataDef}\rangle$

$\langle\mathit{ListParDef}\rangle$ ::= $\langle\mathit{ParDef}\rangle$;
         | $\langle\mathit{ParDef}\rangle$; $\langle\mathit{ListParDef}\rangle$

$\langle\mathit{ListPrintDef}\rangle$ ::= $\langle\mathit{PrintDef}\rangle$;
         | $\langle\mathit{PrintDef}\rangle$; $\langle\mathit{ListPrintDef}\rangle$

$\langle\mathit{ListFlagDef}\rangle$ ::= $\langle\mathit{FlagDef}\rangle$;
         | $\langle\mathit{FlagDef}\rangle$; $\langle\mathit{ListFlagDef}\rangle$

$\langle\mathit{ListParConstr}\rangle$ ::= ϵ
         | $\langle\mathit{ParConstr}\rangle$
         | $\langle\mathit{ParConstr}\rangle$ | $\langle\mathit{ListParConstr}\rangle$

$\langle\mathit{ListIdent}\rangle$ ::= $\langle\mathit{Ident}\rangle$
         | $\langle\mathit{Ident}\rangle$, $\langle\mathit{ListIdent}\rangle$

$$\langle \text{Name} \rangle \quad ::= \quad \langle \text{Ident} \rangle$$
$$\quad\quad | \quad \texttt{[} \langle \text{Ident} \rangle \texttt{]}$$

$$\langle \text{ListName} \rangle \quad ::= \quad \langle \text{Name} \rangle$$
$$\quad\quad | \quad \langle \text{Name} \rangle \texttt{,} \langle \text{ListName} \rangle$$

$$\langle \text{LocDef} \rangle \quad ::= \quad \langle \text{ListIdent} \rangle \texttt{:} \langle \text{Exp} \rangle$$
$$\quad\quad | \quad \langle \text{ListIdent} \rangle \texttt{=} \langle \text{Exp} \rangle$$
$$\quad\quad | \quad \langle \text{ListIdent} \rangle \texttt{:} \langle \text{Exp} \rangle \texttt{=} \langle \text{Exp} \rangle$$

$$\langle \text{ListLocDef} \rangle \quad ::= \quad \epsilon$$
$$\quad\quad | \quad \langle \text{LocDef} \rangle$$
$$\quad\quad | \quad \langle \text{LocDef} \rangle \texttt{;} \langle \text{ListLocDef} \rangle$$

$$\langle \text{Exp4} \rangle \quad ::= \quad \langle \text{Ident} \rangle$$
$$\quad\quad | \quad \texttt{\{} \langle \text{Ident} \rangle \texttt{\}}$$
$$\quad\quad | \quad \texttt{\%} \langle \text{Ident} \rangle \texttt{\%}$$
$$\quad\quad | \quad \langle \text{Sort} \rangle$$
$$\quad\quad | \quad \langle \text{String} \rangle$$
$$\quad\quad | \quad \langle \text{Integer} \rangle$$
$$\quad\quad | \quad \texttt{?}$$
$$\quad\quad | \quad \texttt{[]}$$
$$\quad\quad | \quad \texttt{data}$$
$$\quad\quad | \quad \texttt{[} \langle \text{Ident} \rangle \langle \text{Exps} \rangle \texttt{]}$$
$$\quad\quad | \quad \texttt{[} \langle \text{String} \rangle \texttt{]}$$
$$\quad\quad | \quad \texttt{\{} \langle \text{ListLocDef} \rangle \texttt{\}}$$
$$\quad\quad | \quad \texttt{<} \langle \text{ListTupleComp} \rangle \texttt{>}$$
$$\quad\quad | \quad \texttt{(} \texttt{in} \langle \text{Ident} \rangle \texttt{)}$$
$$\quad\quad | \quad \texttt{<} \langle \text{Exp} \rangle \texttt{:} \langle \text{Exp} \rangle \texttt{>}$$
$$\quad\quad | \quad \texttt{(} \langle \text{Exp} \rangle \texttt{)}$$
$$\quad\quad | \quad \langle \text{LString} \rangle$$

$$\langle \text{Exp3} \rangle \quad ::= \quad \langle \text{Exp3} \rangle \texttt{.} \langle \text{Label} \rangle$$
$$\quad\quad | \quad \texttt{\{} \langle \text{Ident} \rangle \texttt{.} \langle \text{Ident} \rangle \texttt{\}}$$
$$\quad\quad | \quad \texttt{\%} \langle \text{Ident} \rangle \texttt{.} \langle \text{Ident} \rangle$$
$$\quad\quad | \quad \langle \text{Exp4} \rangle$$

$\langle Exp2 \rangle$ ::= $\langle Exp2 \rangle$ $\langle Exp3 \rangle$
| table { $\langle ListCase \rangle$ }
| table $\langle Exp4 \rangle$ { $\langle ListCase \rangle$ }
| table $\langle Exp4 \rangle$ [$\langle ListExp \rangle$]
| case $\langle Exp \rangle$ of { $\langle ListCase \rangle$ }
| variants { $\langle ListExp \rangle$ }
| pre { $\langle Exp \rangle$; $\langle ListAltern \rangle$ }
| strs { $\langle ListExp \rangle$ }
| $\langle Ident \rangle$ @ $\langle Exp4 \rangle$
| $\langle Exp3 \rangle$
| Lin $\langle Ident \rangle$

$\langle Exp1 \rangle$ ::= $\langle Exp1 \rangle$! $\langle Exp2 \rangle$
| $\langle Exp1 \rangle$ * $\langle Exp2 \rangle$
| $\langle Exp1 \rangle$ ** $\langle Exp2 \rangle$
| $\langle Exp2 \rangle$

$\langle Exp \rangle$ ::= \ $\langle ListBind \rangle$ $->$ $\langle Exp \rangle$
| \ \ $\langle ListBind \rangle$ $=>$ $\langle Exp \rangle$
| $\langle Decl \rangle$ $->$ $\langle Exp \rangle$
| $\langle Exp1 \rangle$ $=>$ $\langle Exp \rangle$
| $\langle Exp1 \rangle$ ++ $\langle Exp \rangle$
| $\langle Exp1 \rangle$ + $\langle Exp \rangle$
| let { $\langle ListLocDef \rangle$ } in $\langle Exp \rangle$
| let $\langle ListLocDef \rangle$ in $\langle Exp \rangle$
| $\langle Exp1 \rangle$ where { $\langle ListLocDef \rangle$ }
| fn { $\langle ListEquation \rangle$ }
| $\langle Exp1 \rangle$

$\langle ListExp \rangle$ ::= ϵ
| $\langle Exp \rangle$
| $\langle Exp \rangle$; $\langle ListExp \rangle$

$\langle Exps \rangle$ ::= ϵ
| $\langle Exp4 \rangle$ $\langle Exps \rangle$

$\langle Patt1 \rangle$ ::= _
| $\langle Ident \rangle$
| { $\langle Ident \rangle$ }
| $\langle Ident \rangle$. $\langle Ident \rangle$
| $\langle Integer \rangle$
| $\langle String \rangle$
| { $\langle ListPattAss \rangle$ }
| < $\langle ListPattTupleComp \rangle$ >
| ($\langle Patt \rangle$)

$\langle Patt \rangle$ ::= $\langle Ident \rangle \, \langle ListPatt \rangle$
| $\langle Ident \rangle$. $\langle Ident \rangle \, \langle ListPatt \rangle$
| $\langle Patt1 \rangle$

$\langle PattAss \rangle$ ::= $\langle ListIdent \rangle = \langle Patt \rangle$

$\langle Label \rangle$ ::= $\langle Ident \rangle$
| $ $\langle Integer \rangle$

$\langle Sort \rangle$ ::= Type
| PType
| Tok
| Str
| Strs

$\langle ListPattAss \rangle$ ::= ϵ
| $\langle PattAss \rangle$
| $\langle PattAss \rangle$; $\langle ListPattAss \rangle$

$\langle PattAlt \rangle$ ::= $\langle Patt \rangle$

$\langle ListPatt \rangle$ ::= $\langle Patt1 \rangle$
| $\langle Patt1 \rangle \, \langle ListPatt \rangle$

$\langle ListPattAlt \rangle$ ::= $\langle PattAlt \rangle$
| $\langle PattAlt \rangle$ | $\langle ListPattAlt \rangle$

$\langle Bind \rangle$ ::= $\langle Ident \rangle$
| _

$\langle ListBind \rangle$ ::= ϵ
| $\langle Bind \rangle$
| $\langle Bind \rangle$, $\langle ListBind \rangle$

$\langle Decl \rangle$ ::= ($\langle ListBind \rangle$: $\langle Exp \rangle$)
| $\langle Exp2 \rangle$

$\langle TupleComp \rangle$ ::= $\langle Exp \rangle$

$\langle PattTupleComp \rangle$ ::= $\langle Patt \rangle$

$\langle ListTupleComp \rangle$ ::= ϵ
| $\langle TupleComp \rangle$
| $\langle TupleComp \rangle$, $\langle ListTupleComp \rangle$

$\langle ListPattTupleComp \rangle$ ::= ϵ
| $\langle PattTupleComp \rangle$
| $\langle PattTupleComp \rangle$, $\langle ListPattTupleComp \rangle$

$\langle Case \rangle \quad ::= \quad \langle ListPattAlt \rangle \Rightarrow \langle Exp \rangle$

$\langle ListCase \rangle \quad ::= \quad \langle Case \rangle$
$\qquad\qquad\qquad | \quad \langle Case \rangle \; ; \; \langle ListCase \rangle$

$\langle Equation \rangle \quad ::= \quad \langle ListPatt \rangle \; -> \; \langle Exp \rangle$

$\langle ListEquation \rangle \quad ::= \quad \epsilon$
$\qquad\qquad\qquad\quad | \quad \langle Equation \rangle$
$\qquad\qquad\qquad\quad | \quad \langle Equation \rangle \; ; \; \langle ListEquation \rangle$

$\langle Altern \rangle \quad ::= \quad \langle Exp \rangle \; / \; \langle Exp \rangle$

$\langle ListAltern \rangle \quad ::= \quad \epsilon$
$\qquad\qquad\qquad\quad | \quad \langle Altern \rangle$
$\qquad\qquad\qquad\quad | \quad \langle Altern \rangle \; ; \; \langle ListAltern \rangle$

$\langle DDecl \rangle \quad ::= \quad (\; \langle ListBind \rangle : \langle Exp \rangle \;)$
$\qquad\qquad\qquad | \quad \langle Exp4 \rangle$

$\langle ListDDecl \rangle \quad ::= \quad \epsilon$
$\qquad\qquad\qquad\quad | \quad \langle DDecl \rangle \, \langle ListDDecl \rangle$

$\langle OldGrammar \rangle \quad ::= \quad \langle Include \rangle \, \langle ListTopDef \rangle$

$\langle Include \rangle \quad ::= \quad \epsilon$
$\qquad\qquad\qquad | \quad$ `include` $\langle ListFileName \rangle$

$\langle FileName \rangle \quad ::= \quad \langle String \rangle$
$\qquad\qquad\qquad\quad | \quad \langle Ident \rangle$
$\qquad\qquad\qquad\quad | \quad / \; \langle FileName \rangle$
$\qquad\qquad\qquad\quad | \quad . \; \langle FileName \rangle$
$\qquad\qquad\qquad\quad | \quad - \; \langle FileName \rangle$
$\qquad\qquad\qquad\quad | \quad \langle Ident \rangle \, \langle FileName \rangle$

$\langle ListFileName \rangle \quad ::= \quad \langle FileName \rangle \; ;$
$\qquad\qquad\qquad\quad | \quad \langle FileName \rangle \; ; \; \langle ListFileName \rangle$