The Language Stella

BNF-converter

March 18, 2023

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 Stella

Literals

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

StellaIdent literals are recognized by the regular expression (‘_’ | ⟨*letter*⟩)([“:_”] | ⟨*digit*⟩ | ⟨*letter*⟩)∗

ExtensionName literals are recognized by the regular expression ‘”([“ −_”] | ⟨*digit*⟩ | ⟨*letter*⟩) ∗ ‘”

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 Stella are the following:

`Bool`	`Nat`	`Unit`
`and`	`as`	`cons`
`core`	`else`	`extend`
`false`	`fix`	`fn`
`fold`	`if`	`in`
`inline`	`language`	`let`
`match`	`not`	`or`
`record`	`return`	`struct`
`succ`	`then`	`throws`
`true`	`type`	`unfold`
`variant`	`with`	μ

The symbols used in Stella are the following:

`;`	`,`	`(`
`)`	`{`	`}`
`=`	`:`	`->`
`=>`	`<`	`>`
`[`	`]`	`<=`
`>=`	`==`	`!=`
`+`	`*`	`List::head`
`List::isempty`	`List::tail`	`Nat::pred`
`Nat::iszero`	`Nat::rec`	`.`

Comments

Single-line comments begin with `//`.
There are no multiple-line comments in the grammar.

The syntactic structure of Stella

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.

$\langle Program \rangle$ ::= $\langle LanguageDecl \rangle$ $\langle ListExtension \rangle$ $\langle ListDecl \rangle$

$\langle LanguageDecl \rangle$ ::= `language` `core` `;`

$\langle Extension \rangle$ ::= `extend` `with` $\langle ListExtensionName \rangle$

⟨*ListExtensionName*⟩ ::= ϵ
| ⟨*ExtensionName*⟩
| ⟨*ExtensionName*⟩ `,` ⟨*ListExtensionName*⟩

⟨*ListExtension*⟩ ::= ϵ
| ⟨*Extension*⟩ `;` ⟨*ListExtension*⟩

⟨*Decl*⟩ ::= ⟨*ListAnnotation*⟩ `fn` ⟨*StellaIdent*⟩ `(` ⟨*ListParamDecl*⟩ `)` ⟨*ReturnType*⟩ ⟨*ThrowType*
| `type` ⟨*StellaIdent*⟩ `=` ⟨*Type*⟩

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

⟨*LocalDecl*⟩ ::= ⟨*Decl*⟩

⟨*ListLocalDecl*⟩ ::= ϵ
| ⟨*LocalDecl*⟩ `;` ⟨*ListLocalDecl*⟩

⟨*Annotation*⟩ ::= `inline`

⟨*ListAnnotation*⟩ ::= ϵ
| ⟨*Annotation*⟩ ⟨*ListAnnotation*⟩

⟨*ParamDecl*⟩ ::= ⟨*StellaIdent*⟩ `:` ⟨*Type*⟩

⟨*ListParamDecl*⟩ ::= ϵ
| ⟨*ParamDecl*⟩
| ⟨*ParamDecl*⟩ `,` ⟨*ListParamDecl*⟩

⟨*ReturnType*⟩ ::= ϵ
| `->` ⟨*Type*⟩

⟨*ThrowType*⟩ ::= ϵ
| `throws` ⟨*ListType*⟩

⟨*Expr*⟩ ::= `if` ⟨*Expr*⟩ `then` ⟨*Expr*⟩ `else` ⟨*Expr*⟩
| `let` ⟨*StellaIdent*⟩ `=` ⟨*Expr*⟩ `in` ⟨*Expr*⟩
| ⟨*Expr1*⟩

⟨*ListExpr*⟩ ::= ϵ
| ⟨*Expr*⟩
| ⟨*Expr*⟩ `,` ⟨*ListExpr*⟩

⟨*MatchCase*⟩ ::= ⟨*Pattern*⟩ `=>` ⟨*Expr*⟩

⟨*ListMatchCase*⟩ ::= ϵ
| ⟨*MatchCase*⟩
| ⟨*MatchCase*⟩ `;` ⟨*ListMatchCase*⟩

```
⟨Pattern⟩  ::=  < ⟨StellaIdent⟩ = ⟨Pattern⟩ >
             |  { ⟨ListPattern⟩ }
             |  record { ⟨ListLabelledPattern⟩ }
             |  [ ⟨ListPattern⟩ ]
             |  cons ( ⟨Pattern⟩ , ⟨Pattern⟩ )
             |  false
             |  true
             |  ⟨Integer⟩
             |  succ ( ⟨Pattern⟩ )
             |  ⟨StellaIdent⟩
             |  ( ⟨Pattern⟩ )

⟨ListPattern⟩  ::=  ϵ
                 |  ⟨Pattern⟩
                 |  ⟨Pattern⟩ , ⟨ListPattern⟩

⟨LabelledPattern⟩  ::=  ⟨StellaIdent⟩ = ⟨Pattern⟩

⟨ListLabelledPattern⟩  ::=  ϵ
                         |  ⟨LabelledPattern⟩
                         |  ⟨LabelledPattern⟩ , ⟨ListLabelledPattern⟩

⟨Binding⟩  ::=  ⟨StellaIdent⟩ = ⟨Expr⟩

⟨ListBinding⟩  ::=  ϵ
                 |  ⟨Binding⟩
                 |  ⟨Binding⟩ , ⟨ListBinding⟩

⟨Expr0⟩  ::=  ⟨Expr1⟩ < ⟨Expr1⟩
           |  ⟨Expr1⟩ <= ⟨Expr1⟩
           |  ⟨Expr1⟩ > ⟨Expr1⟩
           |  ⟨Expr1⟩ >= ⟨Expr1⟩
           |  ⟨Expr1⟩ == ⟨Expr1⟩
           |  ⟨Expr1⟩ != ⟨Expr1⟩

⟨Expr1⟩  ::=  ⟨Expr1⟩ as ⟨Type⟩
           |  fn ( ⟨ListParamDecl⟩ ) { return ⟨Expr⟩ ; }
           |  { ⟨ListExpr⟩ }
           |  record { ⟨ListBinding⟩ }
           |  < ⟨StellaIdent⟩ = ⟨Expr⟩ >
           |  match ⟨Expr1⟩ { ⟨ListMatchCase⟩ }
           |  [ ⟨ListExpr⟩ ]
           |  ⟨Expr1⟩ + ⟨Expr2⟩
           |  ⟨Expr1⟩ or ⟨Expr2⟩
           |  ⟨Expr2⟩
```

```
⟨Expr2⟩    ::=  ⟨Expr2⟩ * ⟨Expr3⟩
            |   ⟨Expr2⟩ and ⟨Expr3⟩
            |   ⟨Expr3⟩

⟨Expr3⟩    ::=  ⟨Expr3⟩ ( ⟨ListExpr⟩ )
            |   ⟨Expr4⟩

⟨Expr4⟩    ::=  cons ( ⟨Expr⟩ , ⟨Expr⟩ )
            |   List::head ( ⟨Expr⟩ )
            |   List::isempty ( ⟨Expr⟩ )
            |   List::tail ( ⟨Expr⟩ )
            |   succ ( ⟨Expr⟩ )
            |   not ( ⟨Expr⟩ )
            |   Nat::pred ( ⟨Expr⟩ )
            |   Nat::iszero ( ⟨Expr⟩ )
            |   fix ( ⟨Expr⟩ )
            |   Nat::rec ( ⟨Expr⟩ , ⟨Expr⟩ , ⟨Expr⟩ )
            |   fold [ ⟨Type⟩ ] ⟨Expr5⟩
            |   unfold [ ⟨Type⟩ ] ⟨Expr5⟩
            |   ⟨Expr5⟩

⟨Expr5⟩    ::=  ⟨Expr5⟩ . ⟨StellaIdent⟩
            |   ⟨Expr5⟩ . ⟨Integer⟩
            |   true
            |   false
            |   ⟨Integer⟩
            |   ⟨StellaIdent⟩
            |   ( ⟨Expr⟩ )

⟨Type⟩    ::=  fn ( ⟨ListType⟩ ) −> ⟨Type⟩
           |   μ ⟨StellaIdent⟩ . ⟨Type⟩
           |   ⟨Type1⟩

⟨Type1⟩    ::=  { ⟨ListType⟩ }
            |   struct { ⟨ListFieldType⟩ }
            |   variant < ⟨ListFieldType⟩ >
            |   [ ⟨Type⟩ ]
            |   ⟨Type2⟩

⟨Type2⟩    ::=  Bool
            |   Nat
            |   Unit
            |   ⟨StellaIdent⟩
            |   ( ⟨Type⟩ )

⟨ListType⟩    ::=  ϵ
               |   ⟨Type⟩
               |   ⟨Type⟩ , ⟨ListType⟩
```

$$\begin{array}{lll} \langle \mathit{FieldType} \rangle & ::= & \langle \mathit{StellaIdent} \rangle \texttt{ : } \langle \mathit{Type} \rangle \end{array}$$

$$\begin{array}{lll} \langle \mathit{ListFieldType} \rangle & ::= & \epsilon \\ & | & \langle \mathit{FieldType} \rangle \\ & | & \langle \mathit{FieldType} \rangle \texttt{ , } \langle \mathit{ListFieldType} \rangle \end{array}$$

$$\begin{array}{lll} \langle \mathit{Typing} \rangle & ::= & \langle \mathit{Expr} \rangle \texttt{ : } \langle \mathit{Type} \rangle \end{array}$$