BNF for the Gryph Programming Language

Vitor Greati Artur Curinga Carlos Vieira Vinícius Campos

May 15, 2018

Contents

1 General structure

1.1 Program

$$\begin{aligned} \langle \text{program} \rangle &\models \langle \text{program-unit} \rangle \mid \langle \text{program-unit} \rangle \langle \text{program} \rangle \\ \langle \text{program-unit} \rangle &\models \langle \text{stmt} \rangle \; ; \mid \langle \text{subprog-decl} \rangle \end{aligned}$$

1.2 Statements

$$\begin{aligned} \langle \text{stmt-list} \rangle &\models \langle \text{stmt} \rangle \; ; \mid \langle \text{stmt} \rangle \; ; \langle \text{stmt-list} \rangle \\ \langle \text{stmt} \rangle &\models \langle \text{read-stmt} \rangle \mid \langle \text{print-stmt} \rangle \mid \langle \text{var-decl-stmt} \rangle \end{aligned}$$

1.2.1 IO

$$\begin{aligned} \langle \text{read-stmt} \rangle &\models \text{read} \; \langle \text{ident} \rangle \\ \langle \text{write-stmt} \rangle &\models \text{print} \; \langle \text{ident} \rangle \mid \text{print} \; \langle \text{string-lit} \rangle \end{aligned}$$

1.2.2 Variables

⟨ident-begin-stmt⟩ ⊨ ⟨ident-list⟩⟨ident-list-post⟩
⟨ident-list-post⟩ ⊨ :⟨type⟩⟨var-decl-stmt⟩ | ⟨var-attr-stmt⟩
⟨var-decl-stmt⟩ ⊨ λ | ⟨var-attr-stmt⟩
⟨var-attr-stmt⟩ ⊨ =⟨expr-list⟩

1.3 Subprograms

1.3.1 Declaration

1.3.2 Call

⟨subprog-call⟩ ⊨ ⟨ident⟩(⟨expr-list⟩)

2 Control Structures

2.1 Ifelse statements

⟨if-stmt⟩ ⊨ (⟨b-expr⟩)⟨block-or-stmt⟩

3 Types

⟨type-list⟩ ⊨ ⟨type⟩,⟨type-list⟩ | ⟨type⟩
⟨type⟩ ⊨ ⟨native-type⟩ | ⟨user-type⟩
⟨native-type⟩ ⊨ ⟨primitive-type⟩ | ⟨composite-type⟩
⟨primitive-type⟩ ⊨ int | float | char | string
⟨composite-type⟩ ⊨ [⟨type⟩] | |⟨type⟩| | (⟨type⟩,⟨type-list⟩) | ⟨graph-type⟩
⟨graph-type⟩ ⊨ <⟨type⟩> | <⟨type⟩,⟨type⟩>
⟨user-type⟩ ⊨ ⟨upper-letter⟩⟨alpha-num-list⟩

Observations

- The maximum size of tuples depends on the language implementation, though, in the BNF description above, it may assume any value.

4 Expressions

4.1 Any expression

⟨any-expr⟩ ⊨ ⟨rel-expr⟩ | ⟨bool-expr⟩ | ⟨expr⟩

4.2 Relational expressions

⟨rel-expr⟩ ⊨ ⟨rel-term⟩⟨rel-expr-aux⟩
⟨rel-expr-aux⟩ ⊨ ⟨rel-op⟩⟨rel-term⟩ | ⟨rel-op⟩⟨rel-term⟩⟨rel-expr-aux⟩
⟨rel-op⟩ ⊨ > | < | <= | >= | == | !=
⟨rel-term⟩ ⊨ ⟨any-expr⟩

4.3 Boolean expressions

⟨b-expr⟩ ⊨ ⟨b-term⟩ | ⟨b-term⟩⟨b-expr-aux⟩
⟨b-expr-aux⟩ ⊨ ⟨b-bin-op-p0⟩⟨b-term⟩ | ⟨b-bin-op-p0⟩⟨b-term⟩⟨b-expr-aux⟩
⟨b-term⟩ ⊨ ⟨b-un-op⟩⟨b-term⟩ | ⟨b-un-op⟩⟨b-term⟩⟨b-term-aux⟩ | ⟨b-factor⟩ | ⟨b-factor⟩⟨b-term-aux⟩
⟨b-term-aux⟩ ⊨ ⟨b-bin-op-p1⟩⟨b-factor⟩ | ⟨b-bin-op-p1⟩⟨b-factor⟩⟨b-term-aux⟩
⟨b-un-op⟩ ⊨ not
⟨b-bin-op-p0⟩ ⊨ or | xor
⟨b-bin-op-p1⟩ ⊨ and
⟨b-factor⟩ ⊨ (⟨b-expr⟩) | true | false | ⟨rel-expr⟩

4.4 Expressions with numbers, lists and strings

⟨expr⟩ ⊨ ⟨term⟩ | ⟨term⟩⟨expr-aux⟩
⟨expr-aux⟩ ⊨ ⟨bin-op-p0⟩⟨term⟩ | ⟨bin-op-p0⟩⟨term⟩⟨expr-aux⟩
⟨term⟩ ⊨ ⟨factor⟩ | ⟨factor⟩⟨term-aux⟩
⟨term-aux⟩ ⊨ ⟨bin-op-p1⟩⟨factor⟩ | ⟨bin-op-p1⟩⟨factor⟩⟨term-aux⟩
⟨factor⟩ ⊨ ⟨literal⟩^⟨factor⟩ | ⟨literal⟩
⟨literal⟩ ⊨ ⟨basis⟩ | +⟨basis⟩ | -⟨basis⟩
⟨basis⟩ ⊨ (⟨expr⟩) | ⟨ident⟩ | ⟨int-lit⟩ | ⟨float-lit⟩ | ⟨list-lit⟩ | ⟨subprog-call⟩ | ⟨string-lit⟩
⟨un-op⟩ ⊨ + | -
⟨bin-op-p0⟩ ⊨ + | -
⟨bin-op-p1⟩ ⊨ * | / | % | ++ | **