Specification for the Teko Programming Language

Conor Stuart Roe

December 26, 2018

1 Overview

Teko is a high-level, statically typed scripting language that is primarily interpreted. It follows an object-oriented and imperative programming paradigm. It is intended to offer a full type system with user-defined classes, class hierarchies, and generics, much like Java, and can be rigorously statically type checked. Its high-level architecture offers a variety of advanced data types, such as complex numbers, bytestrings, maps, and structs as primary types, and other features like asynchronous loops easily available. It follows a typical C family syntax; at a glance, Teko looks most like Java, with some syntactic features inspired by Javascript, C++, or Python.

Teko was created in 2018 by Conor Stuart Roe.

2 Lexical and Syntactic Structure

Teko lexical and syntactic structure are defined by the BNF descriptions below. := indicates definition of a nonterminal, pipe || indicates options, parentheses () indicate optional elements, and asterisk * indicates occurrence zero or more times. Angle brackets <> may be used to write English descriptions. Literal characters are written with `code formatting`. Teko is not whitespace-sensitive, except with regard to parsing strings and line comments.

2.1 Lexical Structure

Teko token capture is greedy, so if two adjacent tokens could be mistaken as comprising a single token, they must be separated by whitespace.

Label := Alphabetical (Alphanumeral)*

- Labels are any sequence of alphanumeric characters beginning with an alphabetical character, excluding the following reserved words:

`true`, `false`, `if`, `else`, `for`, `while`, `each`, `begin`, `let`, `class`, `in`, `return`, `public`, `protected`, `private`, `readonly`.

Int := Numeral (Numeral)*

Real := Numeral (Numeral)* `.` (Numeral)*

- Reals may end with a decimal point `.`, which implies a fractional part equal to zero.

Bool := `true` ‖ `false`

Char := `'` CharacterOrEscape `'` ‖ `'"'` ‖ `'\''`

String := `"` (CharacterOrEscape ‖ `'` ‖ `\"`)* `"`

Alphanumeral := Alphabetical ‖ Numeral

CharacterOrEscape := Alphabetical ‖ Numeral ‖ Punct ‖ <space> ‖ `\\` ‖ `\n` ‖ `\t`

- Teko characters and strings may not contain literal tab or newline characters.

Alphabetical := (one of) `ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_`

Numeral := (one of) `0123456789`

Punct := (one of) `!#$%&()*+,-./:;<=>?@]^{|}~`

2.2 Comment Structure

Teko support C-style line comments, which begin with a double slash `//` and end at the next newline.

2.3 Syntactic Structure

Each Teko file consists of a single CodeBlock.

CodeBlock := (Line `;`)*

Line := VariableDeclaration ‖ Assignment ‖ Expression ‖ IfStatement ‖ WhileBlock ‖ ForBlock

VariableDeclaration := `let` Declared Setter Expression (`,` Declared Setter Expression)* || Type Declared (Setter Expression) (`,` Declared (Setter Expression))*

> • Teko permits the declaration of variables using the declarator `let` in place of a type, but doing so requires that all variables declared in this way be set to a value on the same line, so that type can be inferred.

Type := NamedType || Type {} || Type `<>` || Type `[]` || { Type `:` Type } || `(` (Type, (`,` Type)*) `)` || Struct

NamedType := `int` || `real` || `bool` || `char` || `str` || `enum` || `comp` || `bits` || <any label assigned to a type>

Struct := `(` (Type Label (`,` Type Label)*) `)`

Assignment := Declared Setter Expression

Declared := Label (Struct)

Setter := `=` || `+=` || `-=` || `*=` || `/=` || `^=` || `%=`

Expression := Primitive || Label || ||

IfStatement := `if (` Expression `) {` CodeBlock `}` (`else if (` Expression `) {` CodeBlock `}`)* (`else {` CodeBlock `}`)

WhileBlock := `while (` Expression `) {` CodeBlock `}`

ForBlock := `for (` Type Label `in` Expression `) {` CodeBlock `}`