César Liesens - 39161400
Nima Farnoodian - 68372000

LINGI2132
Languages & Translators
Project - Part II

02/04/2021

1 The Rime Programming Language 2.0

Numerous changes have been brought upon Rime since the last part of the project. First (and most significant) of all, the language is now statically typed! Variables and functions (for their return type and arguments) now both require types to be provided in their definitions. As an example, a Rime program's entry point is now defined as `proc main([string]: args) {}`, i.e. a function named `main` that takes a list of strings named `args` as argument and returns no value.

For collections, arrays are now called lists, and maps are now called dicts; these changes were made simply to be more in line with the language's actual implementation and design. Lists, sets and dicts types can be declared respectively as `[type]`, `{type}` and `{keyType: valueType}`, and their corresponding empty initializations can be assigned using the following functions `list(type)` (or `[type]`), `set(type)` and `dict(keyType, valueType)` (where `type` can be any of `bool`, `int` or `string`). As an example, if you wanted to define a single assignment variable named `wordCounts` holding a dictionary mapping strings to integers, you could do it like so : `val {string: int}: wordCounts = dict(string, int)`.

Most other features of the language have remained the same, though some had to be adapted or modified because of the language's new static typing. Functions can still be declared as `procs` or `funcs`, but only `funcs` require a return type to be provided in their definition. Finally, variables must now be assigned when they are declared, and `strings` can now be concatenated using the + operator.

About the implementation itself, many things in the codebase were moved, renamed or changed to give it a more meaningful (or at least more approachable) structure, and make it more convenient to work with.

2 Semantic Analysis

As we know, semantic analysis deals mainly with two things : name (or identifier) resolution and scopes. In the case of a statically typed language such as Rime, type checking also becomes a part of semantic analysis. Type checking for Rime consists of checking that both side of an assignment have matching types ($value \longrightarrow variable$, $argument \longrightarrow parameter$), checking that `if` and `while` conditions are `bool` values, checking that array indices are `int` values, and finally that `dict` keys are a value of a primitive type (excluding `void`). All statements (`if`, `while`, `func`, `proc`) introduce a new scope, and any part of a Rime program can access the root scope and everything it contains. The root scope holds declarations for specific values (`TRUE`, `FALSE` and `NULL`), various predefined functions and all primitive types.

The parts of the implementation which differ the most from the Sigh example are located mainly in `semantic.scope.RootScope` and `semantic.SemanticAnalysis`. For the `semantic.SemanticAnalysis` class, some of the methods with the most changes are `indexedCollectionAccess` and `emptyList/Set/Dict`, along with some of the methods in the `UTILS` region (at the bottom of the class).

Due to an unfortunate lack of time, this part of the project still has quite a few rough edges and requires more work & polish, which will be done prior to the third part on interpretation, along with the addition of more thorough tests for the AST generation.

3 Additional Features

Many of the features we had envisioned, or even started implementing for Rime had to be left out or ignored for the time being, as this second part of the project proved much more challenging than expected. Not all is lost, though! We currently support single and multiple assignment variables (though it still requires an interpreter to be fully enforced), provided with the `val` and `var` keywords; Rime also has native support for sets.

Additionally, we hope to be able to add some other features during the third part of the project, or for the final submission, though we would rather not disclose any of those at the time as they will likely change and/or not be implemented at all (maybe we could expect some functional-like features?).