Relatório INF2102

Victor Nogueira

22 de Junho de 2021

1 Introdução

O projeto escolhido para a disciplina de INF2101 envolve a implementação de módulos na linguagem *Pallene* [1], uma linguagem desenvolvida no *LabLua* com o intuito de ser uma *"Companion Language"* [2] de *Lua* [3]. O objetivo do projeto é permitir com que programas escritos em *Pallene* possam importar outros módulos escritos na mesma linguagem e, assim, fazer uso de funções e variáveis destes.

2 Especificação

O projeto tem como escopo a implementação de módulos na linguagem Pallene, isto é, permitir com que um módulo Pallene possa importar outros módulos Pallene e usar suas funções e variáveis. Até o presente momento, é possível chamar qualquer função de outro módulo, exceto as que tem como argumento, ou retornam, valores do tipo *Record* [4].

3 Projeto

Para descrever o que foi feito no projeto, é desejável introduzir a arquitetura básica de Pallene

- Análise Léxica: Nessa etapa o compilador transforma o código fonte em tokens que farão parte da gramática da linguagem[5]. Por exemplo, a palavra *for* se transforma em um token *for*, o qual será utilzado na regra *forstat* da gramática. Essa etapa tem como entrada o código fonte do arquivo e ocorre em paralelo com a etapa de Análise Sintática
- Análise Sintática: Nessa etapa o compilador analisa os tokens gerados pela etapa anterior e verifica se eles estão de acordo com a gramática da linguagem. A saída dela é a *AST* (Abstract Syntax Tree).
- Análise Semântica: A análise semântica verifica os nomes das variáveis e a tipagem. Erros de tipo ou escopo são gerados nessa etapa.
- Geração da IR: Após a análise semântica, o compilador gera um código em uma representação intermediária, em inglês *intermediate representation* (IR), a qual é usada para representar o código de uma forma que seja mais fácil de se otimizar.
- Geração de Código: Com a IR em mãos, o compilador gera um código C, o qual será, em seguida, compilado pelo GCC [6].

Dividiremos a explicação do projeto em duas partes: A primeira descreverá as melhorias que foram realizadas no Front-end do compilador, antes de se implementar a importação dos módulos em si e a segunda descreverá a importação dos módulos.

```
export function add(x1, x2)
    return x1 + x2
end

export function sub(x1, x2)
    return x1 - x2
end

export x = 2
```

Figura 1: Exemplo de uso de export

3.1 Melhorias no front-end

Antes de começar a de fato permitir a importação de módulos na linguagem Pallene, foi necessário realizar algumas mudanças em três das etapas supracitadas: Análise Léxica, Análise Sintática e Análise Semântica, as quais configuram o *Front-end* do compilador. No início do ano, um programa Pallene dizia quais funções e variáveis deveriam ser exportadas usando a palavra reservada *export*, como exemplificado na figura 1 . Essa abordagem possui alguns problemas, um deles reside na diferença em relação à abordagem de Lua, a qual faz um módulo Lua retornar uma tabela contendo como campos todas as variáveis e funções que o módulo deseja exportar. Como um dos objetivos de pallene é aproximar-se ao máximo da sintaxe de Lua, foram feitas as seguintes modificações:

- Análise Léxica:
  - Remoção de *export* da lista palavras reservadas.
- Análise Sintática:
  - Modificação na gramática, obrigando a definição da variável do tipo *module* no início do programa e o seu retorno ao final do programa.
- Análise Semântica:
  - Introdução do tipo *module*.
  - Variáveis do tipo *module* são especiais, só podem ser declaradas na primeira linha do módulo e só são usadas para representar o módulo.
  - Todas as variáveis e funções exportadas devem ser campos da variável *module*, a qual deve ser a única desse tipo no arquivo.

. Para ilustrar melhor, o exemplo da figura 1, após essas modificações se escreve de acordo com a figura 2. As próximas etapas seguiram inalteradas, pois o código gerado para $m.x = 2$ é o mesmo do gerado anteriormente para *export* $x = 2$, o mesmo se aplica a declaração de funções exportadas.

3.2 Importação de módulos

Na segunda parte do projeto foi implementada a importação de módulos Pallene. Para isso, usou-se a mesma abordagem de Lua, isto é, para importar um módulo Pallene, deve-se chamar a função *require*, passando como argumento uma *string*, a qual se refere ao nome do módulo que se deseja importar. Um exemplo dessa abordagem encontra-se na figura 3. Assim como em Lua, *require* não é uma palavra reservada e é tratada pela gramática como o nome de uma função qualquer.

As modificações nas etapas do compilador foram as seguintes:

- Análise Semântica:
  - Quando algum módulo é importado, o compilador é chamado recursivamente e a AST do módulo importado é gerada. Com essa AST em mãos, o compilador tem o nome e o tipo de todas as funções e variáveis do outro módulo e, assim, consegue fazer a análise semântica de qualquer acesso a elas. Essa abordagem é temporária e vai ser substituída pela utilização de arquivos de interface, os quais ditaram os tipos das funções e variáveis exportadas.
  - Se o módulo importado não for encontrado, é nessa etapa que ocorre o erro.
- Geração da IR:
  - Adicionou-se os tipos de valores *ImportedVar* e *ImportedFunc* na IR
- Geração de código:
  - Quando algum módulo é importado, o código C que chama a função *require* de Lua, via *lua_call* [7], é gerado e será executada assim que o módulo for carregado. Caso o *require* falhe ao carregar o módulo, um erro Lua será gerado e o programa será abortado.
  - Todas as variáveis e funções do módulo usadas serão carregadas em uma tabela de globais, assim que o módulo for carregado. Se alguma variável ou função não for encontrada, o programa será abortado.
  - Toda chamada de função ou acesso a variáveis de outro módulo será feito usando essa tabela de globais citada no item anterior.
  - Toda chamada a funções de outro módulo é feita via *lua_call*.

```
local m: module = {}
function m.add(x1, x2)
    return x1 + x2
end

function m.sub(x1, x2)
    return x1 - x2
end

m.x = 2
return m
```

Figura 2: Exemplo da figura 1 após as modificações

```
local m: module = {}
local fibonacci = require('fibonacci')
local fibnum = fibonacci.calculate(5)
...
return m
```

Figura 3: Exemplo require

4 Testes

Os testes de Pallene encontram-se na pasta *spec*. Para executar todos os testes, basta rodar o programa 'test-project' que está no diretório raiz do projeto. Vale destacar que é importante baixar as dependências de pallene antes de compilar ou rodar os testes. Todos os passos necessários para isso estão disponíveis no repositório de Pallene [8].

Segue os objetivos de teste para cada etapa. O nome, entre parênteses, representa o nome do arquivo onde a etapa é testada.

- Análise Sintática (parser_spec.lua)
  - Verificar que o arquivo pallene começa com a inicialização de uma variável *module* e termina com seu retorno
  - Verificar que a variável *module* é inicializada com uma declaração vazia ("{}").
  - Verificar que só uma variável do tipo *module* foi declarada.
- Análise Semântica (checker_spec.lua)
  - Verificar que a variável do tipo *module* está sendo usada corretamente, isto é, só é usada para declarar as funções e variáveis exportadas.
  - Verificar que não se exporta variáveis, ou funções com o mesmo nome mais de uma vez.
  - Verificar que nenhuma outra variável tem o mesmo nome da variável do tipo *module*.
  - Verificar que funções exportada suportam qualquer tipo de parâmetro, com exceção de *Records*.
  - Verificar que funções exportada suportam retornar qualquer tipo de valor, com exceção de *Records*.
  - Verificar que módulos suportam exportar variáveis de qualquer tipo, com exceção de *Records*.
  - Verificar que a importação de módulos inexistentes gera erros.
  - Verificar que o uso de variáveis inexistentes de outros módulos gera erros.
  - Verificar que o uso de funções inexistentes de outros módulos gera erros.
- Testes Funcionais (execution_tests.lua)
  - Verificar que *require* é uma função qualquer, ou seja, garantir que é possível declarar uma outra função com esse mesmo nome.
  - Verificar que é possível chamar funções de outros módulos com parâmetros de todos os tipos, menos *Record*.
  - Verificar que é possível chamar funções de outros módulos com valores de retorno de todos os tipos, menos *Record*.
  - Verificar que é possível acessar todos os tipos de variáveis de outros módulos, menos *Record*.
  - Verifica que o uso de módulos não existentes gera erros.

5 Dificuldades

Como o trabalho consistiu em adicionar uma funcionalidade nova a uma base de código já existente, a qual eu nunca havia usado, uma das maiores dificuldades foi entender o código já existente. Embora o código estivesse bem escrito e documentado, a base de código era relativamente grande, portanto era preciso analisar e entender bastante código. Depois de um período

de adpatação, tornou-se mais fácil entender o que cada componente fazia e, consequentemente, alterar o código existente.

O arcabouço de testes utlizado (busted) também me era desconhecido, portanto tive que entender, do zero, como a ferramenta funcionava. Felizmente, a ferramenta era intuitiva e, portanto, não demorou tanto para aprender o básico que era necessário para a realização do projeto.

Referências

[1] H. M. Gualandi, *The Pallene Programming Language*. PhD thesis, PUC-Rio, 2020.

[2] H. M. Gualandi and R. Ierusalimschy, "Pallene: A companion language for lua," *Science of Computer Programming*, vol. 189, p. 102393, 2020.

[3] R. Ierusalimschy, *Programming in Lua*. Lua.org, Fourth ed., 2016.

[4] Gualandi, Hugo Musso, "Documentação de pallene relativa a records." Disponível em: https://github.com/pallene-lang/pallene/blob/master/doc/manual.md#records Acesso em: 22 de Junho. 2021.

[5] Gualandi, Hugo Musso, "Gramática de pallene." Disponível em: https://github.com/pallene-lang/pallene/blob/master/doc/manual.md#the-complete-syntax-of-pallene. Acesso em: 22 de Junho. 2021.

[6] B. Gough and R. M. Stallman, "An introduction to gcc for the gnu compilers gcc and g++," *Network Theory Ltd*, vol. 258, 2004.

[7] Ierusalimschy, Roberto, "Documentação de lua_call." Disponível em: https://www.lua.org/manual/5.4/manual.html#lua_call Acesso em: 22 de Junho. 2021.

[8] Gualandi, Hugo Musso, "Repositório de pallene." Disponível em: https://github.com/pallene-lang/pallene Acesso em: 22 de Junho. 2021.