Название

Автор

29 июля 2025 г.

Язык и платформа

- Язык — OCaml
- Хостинг кода — GitHub
- CI/CD — GitHub Actions

Цель

Создание AOT-компилятора вымышленного императивного языка, способного компилировать относительно несложные программы

Задачи

- Изучение процесса компиляции в современных ЯП
- ???Выбор языка и подготовка окружения
- Реализация фронтенда
  - Лексический анализ
  - Синтаксический анализ
  - Семантический анализ
- Реализация бекенда
  - Порождение RISC-V Assembly
- Рефакторинг кода и оформление репозитория

О процессе компиляции

- Каждый этап обрабатывается отдельным модулем
- Модуль принимает некоторое представление программы и возвращает новое представление, снабженное дополнительной информацией
- ???Компиляция представляет собой процесс последовательной передачи результатов вычислений предыдущего модуля в следующий модуль

Лексический анализ

Задачи:

- Преобразовать строку в последовательность токенов
- Выявить лексические ошибки

Сигнатура функции:

```
val tokenize :
    string -> (token list, lexer_error) result
```

В качестве токенов могут выступать:

- Ключевые слова:
  - `while`, `do`, `done`, `if`, `then`, `else`, `fi` ...
- Литералы:
  - `Int 42`, `true`, `false`, `String "hello world"`
- Унарные и бинарные операторы и скобки
- Идентификаторы переменных и функций

Синтаксический анализ

Задачи:

- Преобразовать последовательность токенов в AST
- Выявить синтаксические ошибки

Сигнатура функции:

```
val parse_to_program :
    token list -> (program, parser_error) result
```

Что из себя представляет AST:

```
type program = statement list

type statement =
| Assignment of string * expression
| While of expression * program
| Ite of expression * program * program
| ...

type expression =
| Var of string
| Int of int
| BinOp of operation * expression * expression
| ...
```

Семантический анализ

Задачи:

- Дополнить AST информацией о типах и областях видимости
- Выявить семантические ошибки

Сигнатура функции:

```
val infer_types :
    token list -> (program, parser_error) result
```

Поддерживаемые на данный момент типы:

- Int32
- Bool
- ASCIIZ String

Как выглядит расширенное AST:

```
type typed_program = (typed_statement * scope)
    list

type typed_statement =
| Typed_Assignment of string * typed_expression
| Typed_While of typed_expression *
    typed_program
| Typed_Ite of typed_expression * typed_program
    * typed_program
| ...

type typed_expression =
| Type_Int of ...
| Type_Bool of ...
| Type_Str of ...
```

Порождение кода

Полученное ранее представление программы является универсальным для любой платформы для которой существует компилятор OCaml подходящей версии.
Порождение кода реализовано для riscv64.

???

- Ассемблирование
  - Например с помощью `riscv64-unknown-elf-as`
  - Результат — объектный модуль
- Линковка
  - Например с помощью `riscv64-unknown-elf-ld`
  - Результат — исполняемый файл
- Исполнение
  - Или нативно, если архитектура соответствующая
  - Или помощью эмулятора, например `qemu`

Использовавшиеся инструменты

- `ppx_expect` и `ppx_deriving`
  - Специфичные для OCaml фреймворки значительно упрощающие тестирование и отладку
- `riscv64-unknown-elf-as` и `riscv64-unknown-elf-ld`
- `qemu-riscv64` и `spike`