Санкт-Петербургский государственный университет

Кафедра системного программирования

Реализация привязок к MLIR для OCaml

Семен Евгеньевич Хечнев, группа 21.Б07-мм

Научный руководитель: Д.С. Косарев, ассистент кафедры системного программирования

Санкт-Петербург
2023

Введение

- Существует множество инструментов, реализованных на разных языках
- Чтобы использовать решения, написанные на других языках программирования, реализуют привязки (от англ. bindings)
- LLVM[1] — инфраструктура для разработки компиляторов
- MLIR[2] (Multi-Level Intermediate Representation) — подпроект LLVM, инфраструктура для определения и внедрения промежуточных представлений
- MLIR реализован на C++, хочется использовать в OCaml

[1] `https://llvm.org/` (дата доступа: 4 января 2024 г.).
[2] `https://mlir.llvm.org/` (дата доступа: 4 января 2024 г.).

Обзор — MLIR (1)

MLIR предоставляет:

- Гибкое базовое промежуточное представление в SSA (Static Single Assignment) форме
  - SSA $\Leftrightarrow$ каждой переменной значение присваивается ровно один раз
- Инструменты для расширения и анализа IR (Intermediate Representation)
- Готовые проходы (passes)
- Инструменты для тестирования и отладки

Обзор — MLIR (2)

Базовые **объекты** в MLIR:

- Основная сущность — операция
- Атрибуты — compile-time константы, прикреплённые к операциям
- Типы
- Блоки — список операций, заканчивающийся терминатором
- Регионы — список блоков

Listing 1: Пример операции

```
^bb1:
  %res:2 = "mydialect.op"(%arg1) { attr = 3 }
      : (i32) -> (f32, i8) loc("mysrc.ml":10:8)
  ...
```

Обзор — MLIR (3)

- Для поддержки расширяемости IR MLIR использует диалекты
- Диалект — пространство имён для операций, типов и атрибутов
- Базовое промежуточное представление — диалект «builtin»[3]

[3] `https://mlir.llvm.org/docs/Dialects/Builtin/` (дата доступа: 4 января 2024 г.).

Постановка задачи

Целью работы является реализация привязок к MLIR для OCaml
Задачи:

- Реализовать привязки к MLIR
- Разработать компилятор для мини языка программирования с использованием разработанных привязок
- Реализовать тесты

Обзор — ctypes

- Ctypes[4] — это OCaml библиотека для создания привязок к C
  - Для создания привязки необходимо определить имя и сигнатуру связываемой функции

Listing 2: Пример привязки «print_int» для функции C, которая принимает `int` и возвращает `void`

```
let print_int = foreign "print_int"
                        (int @-> returning void)
```

[4] https://github.com/yallop/ocaml-ctypes (дата доступа: 4 января 2024 г.).

Обзор — существующие решения

- За основу взяты существующие привязки[5] к MLIR
- Особенности:
  - Реализованы к форку MLIR 11 версии
  - Не примитивного примера использования нет
  - Больше не поддерживаются

[5] `https://github.com/tachukao/ocaml-mlir` (дата доступа: 4 января 2024 г.).

Реализация — привязки

Что было сделано:

- Выбрана целевая версия MLIR — 16.0.6
- Обновлены зависимости и удалён устаревший код
- Добавлена возможность собирать привязки, используя dev пакеты[6] MLIR и LLVM
- Реализовано большинство недостающих привязок

[6] `https://apt.llvm.org/` (дата доступа: 4 января 2024 г.).

Реализация — мини язык программирования (1)

- Тип данных — тензоры (ранга 0–2) с элементами типа `float64`
- Операции:
  - печать
  - поэлементное сложение и умножение
  - транспонирование
  - изменение формы тензора
- Можно определять функции
- Переменные неизменяемы, статическая типизация, вывод типов

Listing 3: Пример программы на мини языке программирования

```
def main() {
    var a = [[1, 2, 3], [4, 5, 6]];
    var b<3, 2> = [1, 2, 3, 4, 5, 6];
    print(a * transpose(b));
}
```

https://mlir.llvm.org/docs/Tutorials/Toy/

Реализация — мини язык программирования (2)

- Процесс компиляции:
  - Парсинг в AST
  - Генерация из AST IR высокого уровня
  - Вывод типов
  - Оптимизация IR
  - Понижение (lowering) IR высокого уровня в более низкоуровневое
  - Оптимизация IR
  - Понижение в диалект LLVM
  - Понижение в LLVM IR
  - Компиляция LLVM IR при помощи JIT

Тестирование

- Для привязок уже были реализованы тесты, оставалось внести небольшие правки, чтобы они работали для версии 16.0.6
- Для каждого этапа компиляции мини языка программирования были реализованы тесты
- CI
  - Сборка
  - Запуск тестов
  - Обновление отчёта о покрытии
- Тестовое покрытие
  - Покрытие всего проекта — 55%
    - Привязок — 46%
    - Компилятора мини языка — 96%

Заключение

Результаты:

- Реализованы привязки к MLIR
- Разработан компиляторо для мини языка программирования с использованием реализованных привязок
- Реализованы тесты

Планы на будущее:

- Реализовать компилятор для подмножества некоторого ML языка с использованием MLIR

Код проекта: `https://github.com/s-khechnev/ocaml-mlir`
Имя аккаунта: s-khechnev