Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Разработка транслятора модельного функционального языка в Interaction Nets

ПОНОМАРЕВ Николай Алексеевич

Отчёт по производственной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Григорьев С. В.

Санкт-Петербург
2024

Оглавление

Введение

Лучше это не читать

v1

Интерес научного сообщества в данный момент обращен к таким областям, как анализ графов или искусственный интеллект, в основе которых лежат алгоритмы разреженной линейной алгебры. Поскольку требуется обрабатывать огромные объёмы информации, то для решения задач прибегают к техникам параллельного программирования. К сожалению, распараллеливание разреженной линейной алгебры — сложная задача для традиционных архитектуры: нелокальные обращения к памяти, непредсказуемое количество "агентов" (нерегулярность) (TODO: да? что-то ещё? ссылки?). Для решения этих проблем применяют GPU или ускорители на специализированных архитектурах.

Interaction Nets — модель вычислений, которая была изобретена Yves Lafont, в 1990 году. В этой модели «программа» представляется в виде графа, и, в силу свойств модели, вычисления происходят только локально и между конечным множеством вершин за шаг, поэтому в данной модели легко достигается параллельность.

(Тут чего-то не хватает) В рамках проекта Lamagraph исследуются возможности по разработке параметризуемого многоядерного сопроцессора для разреженной линейной алгебры на архитектуре Interaction Nets.

Поскольку архитектура, основанная на Interaction Nets, — полностью отличается от уже существующих, а сам проект экспериментальный, то использование существующих трансляторов может только усложнить разработку, поэтому целью данной работы является разработка транслятора модельного функционального языка в Interaction Nets.

v2

Искусственный интеллект и анализ графов — одни из наиболее привлекательных областей науки в данный момент [1, 2]. Многие алгоритмы, используемые в этих областях, основаны на линейной алгебре или могут

быть переформулированы в её терминах, (здесь нужно сказать почему линал хорош). Поскольку вычисления в линейной алгебре часто независимы друг от друга, разумно использовать возможности параллельного программирования для ускорения работы алгоритмов. А для больших объемов данных разумно использовать разреженную линейную алгебру.

1. Постановка задачи

Целью работы является разработка транслятора модельного функционального языка в Interaction Nets. Для её выполнения были поставлены следующие задачи:

1. Реализовать интерпретатор модельного ML-подобного языка
2. Реализовать транслятор из обогащенного λ-исчисления в Interaction Nets
3. Реализовать интерпретатор Interaction Nets
4. Провести эксперименты с наборами инструкций

2. Обзор

2.1. Описание Interaction Nets

Здесь базовое описание системы, списанное с Лафона [4] и/или Салихметова [7]. Пока отсутствует, потому что некогда.

Сейчас важно, что система — ориентированный граф, в вершинах которого некоторые метки-агенты. И правила редукции мы можем выбирать сами. Так что простор для параметризации огромный.

2.2. Обзор существующих решений

В области уже что-то делали.

2.2.1. inpla/train

[5] Прикольное — свой язык в терминах агентов. Однако программировать надо прямо на нём, что может быть сложно в большом проекте.

train попытка затащить на GPU.

https://github.com/inpla/inpla/

2.2.2. HVM 1,2,3 + Bend

Тут отдельно low-level HVM, отдельно high-level Bend. Видимо, набор агентов фиксирован. Вообще, надо подумать какие тут проблемы, почему не его доделываем? Треш синтаксис?

https://github.com/HigherOrderCO/HVM

Паперь не опубликован нормально https://github.com/HigherOrderCO/HVM/blob/main/paper/HVM2.pdf

2.2.3. lambda

Честный транслятор λ-исчисления в Interaction Nets [8], который поддерживает несколько правил трансляции. Жаль на JS, и жаль, что умер [проект].

https://github.com/codedot/lambda

2.2.4. interact

Выглядит интересно, язык чуть более функциональные, но похоже, что на самом там на всё будет свой агент.

https://github.com/szeiger/interact

2.2.5. Классические компиляторы

Тут про то, что в теории можно допилить GHC или OCaml. Про GHC точно известно, что он очень не-модульный[1] и всё это API внутреннее и нормально не версионируется. Про OCaml не знаю, чем аргументировать — ну, тоже взрослый компилятор, в который вообще можно впилиться?

F# Quotations, не знаю надо ли упоминать, но если надо, то тогда надо где-то выписать требования, которые откуда-то должны следовать :), но тогда будет понятно, что хотели гетерогенность всего стенда.

2.3. Выводы

Понятия не имею пока, какие выводы, кроме того, что по факту будем делать всё сами.

[1] https://gitlab.haskell.org/ghc/ghc/-/wikis/Make-GHC-codebase-more-modular (дата обращения: 17 декабря 2024 г.)

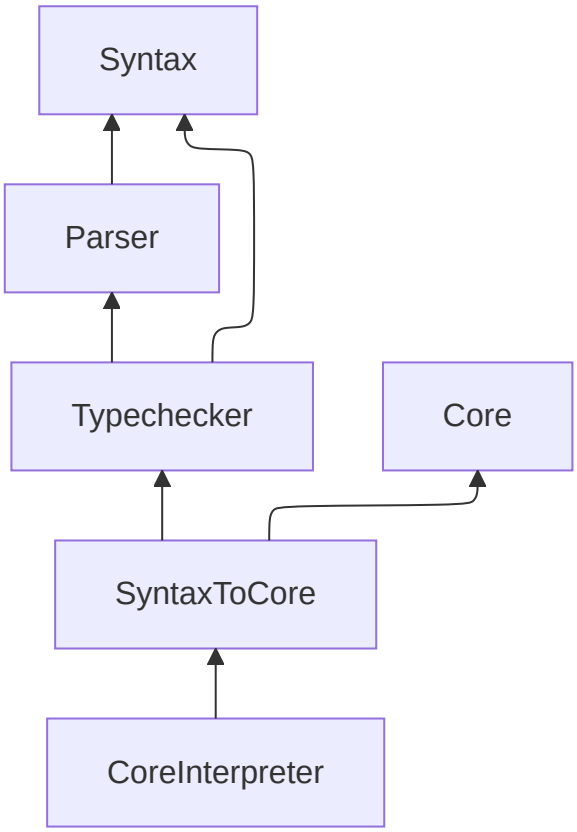

Рис. 1: Архитектура транслятора

3. Описание решения

Я пока не понимаю, куда запихнуть ту большую картинку. Или мб сюда она вообще не нужна? А только в презе понадобится. (Мысли дальше. Нет, где-то надо общее описание проекта, отсюда. Тут вырастет Haskell, Clash и всё такое)

Наш транслятор я буду называть `Lmlc` — от LamagraphML compiler.

3.1. Синтаксис языка

Основан на OCaml, который упросили по максимуму.

Для представления AST (дерева абстрактного синтаксиса) используется паттерн Trees That Grow (TTG) [9]. Он позволяет с помощью механизма type families гибко параметризовать дерево необходимыми аннотациями, более того они могут быть разными для разных узлов

дерева, тем самым поддерживая безопасность кода.

3.2. Парсер

Для парсинга используется связка лексера Alex и парсер-генератора Happy, которые являются аналогами flex и bison, написанными на Haskell. (Здесь Haskell возник из ниоткуда, надо подумать куда его лучше, сюда в начала или прямо в обзор) (Короче, скажи, где должно быть описание проекта в общем.)

На этой стадии получается дерево с параметром (type family) `LmlcPs`, `Ps` от `Parsed`. Аннотаций на данном этапе отсутствуют.

3.3. Вывод типов

Поскольку язык ML-подобный, используется система типов Хиндли-Милнера [3, 6].

Вот эта задача вышла какой-то очень сложной для понимания. Но мне кажется, что мои стенания на тему того, что там больно с расширением правил с простого языка, `let rec` и паттернами сюда писать не надо.

На данной стадии параметр `LmlcTc`, `Tc` от `Typechecker`. В аннотациях сохраняется тип каждого узла дерева.

3.4. Промежуточный язык

Вариант обогащенного λ-исчисления. Хочется примерно такого

```
type CoreExpr = Expr Var

data Expr b        -- "b" for the type of binders,
      = Var   Id
      | Lit   Literal
      | App   (Expr b) (Arg b)
      | Lam   b (Expr b)
```

```
        | Let    (Bind b) (Expr b)
        | Case   (Expr b) b Type [Alt b]
        | Type   Type

type Arg b = Expr b
type Alt b = (AltCon, [b], Expr b)

data AltCon = DataAlt DataCon | LitAlt  Literal | DEFAULT

data Bind b = NonRec b (Expr b) | Rec [(b, (Expr b))]
```

144 Списано с хаскелля[2]

145 3.5. Интерпретатор

146 Настолько ещё не думал, что написать нечего.

[2] https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/core-syn-type

4. Эксперимент

Сейчас что-то вообще сюда писать???

Я понимаю, что есть планы на всякие ”софтверные” счётчики для разных систем агентов, но надо ли?..

Заключение

В рамках данной производственной практики были достигнуты следующие результаты.

1. Реализован интерпретатор модельного ML-подобного языка

Остальные задачи планируется выполнить в течение весеннего семестра.

Список литературы

[1] Economic Potential of Generative AI / Michael Chui, Eric Hazan, Roger Roberts et al.

[2] García Roberto, Angles Renzo. Path Querying in Graph Databases: A Systematic Mapping Study. — Vol. 12. — P. 33154–33172. — URL: https://ieeexplore.ieee.org/document/10456906 (дата обращения: 2024-12-18).

[3] Hindley R. The Principal Type-Scheme of an Object in Combinatory Logic. — Vol. 146. — P. 29–60. — jstor : 1995158.

[4] Lafont Yves. Interaction Combinators. — Vol. 137, no. 1. — P. 69–101. — URL: https://www.sciencedirect.com/science/article/pii/S0890540197926432 (дата обращения: 2024-12-17).

[5] Mackie Ian, Sato Shinya. Parallel Evaluation of Interaction Nets: Case Studies and Experiments. — Vol. 73. — URL: https://eceasst.org/index.php/eceasst/article/view/2205 (дата обращения: 2024-12-17).

[6] Milner Robin. A Theory of Type Polymorphism in Programming. — Vol. 17, no. 3. — P. 348–375. — URL: https://www.sciencedirect.com/science/article/pii/0022000078900144 (дата обращения: 2024-12-17).

[7] Salikhmetov Anton. Interaction Nets in Russian. — arXiv : cs/1304.1309.

[8] Salikhmetov Anton. Token-Passing Optimal Reduction with Embedded Read-back. — Vol. 225. — P. 45–54. — arXiv : cs/1609.03644.

[9] Shayan Najd, Simon Peyton Jones. Trees That Grow. — URL: https://lib.jucs.org/article/22912 (дата обращения: 2024-12-17).