САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Болотов Сергей Сергеевич

Создание транслятора из РуСи в LLVM IR

Курсовая работа

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Санкт-Петербург
2015

Оглавление

Введение

Создание компилятора языка программирования - большая и сложная задача. Для упрощения этой задачи существует множество различных методов, одним из которых является создание инфраструктур компиляции. Успешность такого подхода связана с тем, что, несмотря на разнообразие различных языков программирования и целевых платформ, при разработке нового компилятора можно переиспользовать что-либо из предыдущих. Классическим приёмом является введение некоторого универсального промежуточного представления, независимого от архитектуры конечной платформы. Таким образом удаётся сократить число разрабатываемых компонентов с $m * n$ до $m + n$. Также инфраструктура компиляции включает в себя ряд инструментов для преобразования промежуточного представления, например, в целях оптимизации. Одной из таких инфраструктур является LLVM [3].

Инфраструктура LLVM - это спецификация промежуточного представления, называемого LLVM IR [2], и большое число программ, предназначенных для работы с ним. Основное место среди них занимает многопроходной оптимизатор. Его дополняют кодогенераторы для наиболее популярных платформ. Дизайн LLVM IR преследует две основных цели - во-первых, быть максимально независимым от языка высокого уровня, транслируемого в LLVM IR и достаточно выразительным, чтобы выразить все его абстракции, и во-вторых, сохранить как можно больше информации, полезной оптимизатору.

РуСи - проект создания языка программирования для начинающих. Он включает в себя спецификацию C-подобного языка программирования и компилятора к нему. Компилятор для РуСи использует непопулярные в современных промышленных компиляторах методы разбора и кодогенерации, такие как рекурсивный спуск, позволяющие получать больше информации об ошибках в исходной программе за счёт увеличения времени работы компилятора. В настоящее время компилятор поддерживает трансляцию в язык внутренней промежуточной вирту-

альной машины и интерпретатор для неё, что отражается на быстродействии работы программы и количестве поддерживаемых платформ.

Компилятор, в свою очередь, также является инструментом для обучения технике трансляции студентов. Поэтому одной из целей создания транслятора является показать реализацию транслятора в существующее и используещееся представление.

Таким образом, цель данной работы - создание транслятора языка РуСи в язык промежуточного представления LLVM IR, как для достижения высокого быстродействия исходных программ, так и для обучения студентов техники трансляции на его основе.

1. Обзор

Так как РуСи - проект новый, попыток создания трансляторов в языки, кроме языка внутренней виртуальной машины, ещё не существует. В то же время существует проект Clang [1], включающий в себя трансляторы языков семейства C в LLVM IR. Перед обсуждением проекта Clang, необходимо дать более подробную характеристику LLVM IR.

1.1. LLVM IR

LLVM IR - это строго типизированная RISC-подобная SSA-форма (Static Single Assignment form [4]), являющаяся промежуточным представлением программ в инфраструктуре LLVM. Система типов LLVM включает множество целочисленных типов произвольной битовой ширины, числовых типов с плавающей запятой, метки и конструкторы типов. Конструкторы типов позволяют определять структуры, массивы, указатели и функциональные типы произвольного уровня вложенности. Виртуальные регистры строго типизированы, их число бесконечно, операции над ними в большинстве своём трёхадресные. Значением регистра так же может быть указатель на адрес в памяти - локальной или глобальной. Адресная арифметика полностью абстрагирована операцией взятия адреса элемента сложного типа. Набор операций с памятью состоит из чтения значения по адресу и записи значения по адресу. Одна из инструкций позволяет выделить участок памяти на стеке функции во время активации. Ввиду использования SSA-формы, допускается только одно определение значение виртуального регистра. Для представления нескольких возможных значений виртуального регистра SSA-форма вводит понятие ϕ-функции, определяющая значение регистра в зависимости от того, по какому пути управление пришло в текущую точку программы. LLVM IR требует явного представления функции в виде линейной развёртки графа потока управления, каждый узел которого представляет именованный базовый блок. Код на LLVM IR имеет представления в виде байт-кода и человекочитаемую

форму.

1.2. Clang

Clang - самый известный компилятор для языков семейства C в инфраструктуре LLVM. Одной из основных целей проекта Clang является высокая производительность транслятора - в среднем Clang работает на 30% быстрее одного из самых популярных компиляторов - GCC, не уступая последнему в количестве платформ, для которых генерируется машинный код. Clang не только использует LLVM IR как внутреннее промежуточное представление, но и позволяет выводить код на LLVM IR в человекочитаемой форме. Такое представление было использовано в данной работе для извлечения правил трансляции конструкций языка C в коды LLVM IR.

2. Реализация

Существующий компилятор РуСи имеет 2 версии. Первая версия представляет собой однопроходный компилятор во внутреннюю виртуальную машину, не имеющий никаких промежуточных представлений, к которым приводится исходная программа в процессе трансляции. В данный момент эта версия компилятора не поддерживается, но она была использована для создания прототипа транслятора из языка РуСи в LLVM IR для получения практического представления о принципах устройства языка РуСи, компилятора языка РуСи и спецификации языка LLVM IR. Вторая версия представляет собой двупроходный компилятор с построением промежуточного видозависимого дерева из исходного кода программы. В данный момент эта версия является основной. Компилятор РуСи выполняет лексический, синтаксический и видозависимый анализы, а также обеспечивает проверку корректности исходной программы с точки зрения системы типов. Форма, следующая за этим деревом, теряет информацию, необходимую для реализации такой группы функций, как printid, о которой будет сказано в дальнейшем.

2.1. Операторы РуСи

Большая часть компонентов РуСи позаимствована из языка C, такие как циклы, условные конструкции, операции, массивы и прочие. В то же время РуСи имеет особенности, отличающие его как от языка C, так и от традиционных языков программирования в частности. В отличие от C, РуСи разрешает использование выражений, содержащих переменные, для указания размера массива при условии, что переменная может иметь единственное значение при инициализации размера массива (т.е. быть объявленной строго на том же уровне перед объявлением массива). Так же РуСи содержит функции print и printid, печатающие текущее значение переменной и название переменной с её текущим значением, которые в обычном C реализовать невозможно, так как для их исполнения необходимо знать тип переменной и её на-

звание, хранящееся в файле исходного кода.

2.2. Промежуточное представление в виде дерева

После выполнения первого прохода компилятор РуСи строит дерево - результат лексического, синтаксического и видозависимого анализов, - представляющее собой граф исполнения программы в линейной развёртке. Узлами дерева являются операторы исходной программы и определения функций и переменных. В дереве содержится вся информация о переменных в контексте работы компилятора. Выражения хранятся приведёнными в совершенную обратную польскую нотацию.

2.3. Основные сведения о реализации

Источником трансляции является видозависимое дерево, указанное выше. Представление в виде кода внутренней виртуальной машины компилятора РуСи теряет информацию, необходимую для генерации человекочитаемой формы LLVM IR, а так же для оптимизации итоговой программы на LLVM IR, в то время как для реализации транслятора из языка РуСи необходимо было бы дополнить транслятор также лексическим, синтаксическим и видозависимым анализатором, что нецелесообразно в контексте их наличия в компиляторе РуСи.

Итоговый транслятор реализован с применением LLVM API на языке C, как и компилятор РуСи. Транслятор реализован как дополнительная функция компилятора РуСи, а не как отдельная программа, и во время второго прохода компилятора генерирует код на LLVM IR и выводит его в виде файла с представлением LLVM IR в виде байт-кода.

LLVM C API является обёрткой для LLVM C++ API и предоставляет полный набор операций, описанных в спецификации языка LLVM IR.

Перед началом работы по реализации были составлены проекции операторов языка РуСИ в конструкции кода на LLVM IR. Каждая проекция представляла собой конструкцию из:

- узла дерева, который претерпит трансляцию, с переменными, соответствующими данному узлу, и
- получаемого кода на LLVM IR, с указанием, где используются и как преобразуются переменные, указанные выше.

Для создания проекций некоторых операторов использовались примеры проекций из кода на языке C в коды языка LLVM IR, полученные с применением инструмента Clang.

После создания проекций была начата работа над реализацией прототипа транслятора, работающего в первой версии компилятора РуСи. Данная реализация не использовала LLVM API, а генерировала текстовую версию программы в человекочитаемой форме LLVM IR. Таким образом перед началом работы с транслятором во второй версии компилятора РуСи было получено практическое знание о трансляции кода на языке РуСи в код на языке LLVM IR.

2.4. Трансляция объявлений

Объявления в глобальном контексте делятся на объявления переменных и объявления функций. Объявления переменных обязаны иметь начальное значение при исполнении программы. При отсутствии выражения инициализации для переменной её значение будет равно значению по умолчанию для переменных такого типа. Объявления и предописания функций в LLVM IR, в отличие от языка C, не обязательно должны стоять выше всех её вхождений в программу, поэтому явные предописания в результирующую программу не транслируются, оставляя только объявление функции.

Объявления в локальном контексте могут быть только объявлениями переменных и могут зависеть от прочих переменных в текущем контексте программы. Для каждого объявления переменной выделяется место на стеке текущей функции с помощью функции alloca, возвращающей адрес на стеке. В дальнейшем по этому адресу будет записываться значение переменной при её изменении. Объявление массива в

локальном контексте с инициализатором требует создания глобального массива, доступного только для чтения, из которого будут браться значения для локального массива с помощью функции memcpy.

2.5. Трансляция выражений

Большинство выражений языка РуСи, таких как константы, целочисленная и вещественная арифметики транслируются в LLVM IR прямолинейным образом. Трансляция тернарного оператора требует ветвления графа потока управления программы и использования ϕ-функции для определения результирующего значения в виртуальном регистре.

2.6. Трансляция конструкций манипуляции потоком управления

Трансляция конструкций манипуляции потоком управления в SSA-форму может быть нетривиальной. Сложности возникают при модифицировании значений - для трансляции циклов, где счётчик является постоянно модифицируемым значением, разработчики LLVM рекомендуют выделять место для счётчика на стеке функции и работать с памятью по адресу счётчика. Так как в языке РуСи, подобно языку C, счётчик необходимо явно объявлять до использования в цикле, изменения значения счётчика происходят вместе с записью нового значения по адресу переменной. Оптимизатор LLVM самостоятельно определяет такие шаблоны использования и переносит данные переменные в виртуальные регистры. LLVM версия оператора switch является достаточно высокоуровневой, чтобы быть напрямую транслированной из РуСи в LLVM IR.

Заключение

В результате работы был реализован транслятор языка РуСи в язык LLVM IR, которые вместе образуют оптимизирующий компилятор для достаточно полного подмножества языка РуСи на целый спектр различных платформ. Компилятор поддерживает:

- целочисленную и вещественную арифметику;
- основные типы данных и одномерные массивы;
- все конструкции манипуляции потоком управления;
- функции вывода print и printid.

Не была реализована работа с двумерными массивами.

Список литературы

[1] C. Lattner. C Language Family Frontend for LLVM. — URL: `http://clang.llvm.org`.

[2] Lattner C. Adve V. LLVM Language Reference Manual. — URL: `http://llvm.org/docs/LangRef.html`.

[3] Lattner C. Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. — University of Illinois at UrbanaChampaign, 2004.

[4] S. Muchnick. Advanced Compiler Design and Implementation. — Morgan Kaufmann, 1997.