Оглавление

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Вступительное слово

Форма с единственным статическим присваиванием (Static Single Assignment – SSA) в настоящее время является преобладающим представлением, на основе которого создаются инструменты анализа и преобразования программ. Можно разработать современный оптимизирующий компилятор, который преобразует входную программу в форму SSA после начального синтаксического и семантического анализа и хранит ее в этом представлении на протяжении всего процесса оптимизации вплоть до генерирования окончательного машинного кода. В этой книге описаны новейшие представления программ в форме SSA и алгоритмы, в которых она используется.

Когда мы только приступали к разработке формы SSA, методы анализа и преобразования программ были совсем другими. Многие тогдашние компиляторы были очень простыми – такие мог бы создать студент на своем первом курсе по компиляции. Они порождали код на основе шаблонов для каждой продукции грамматики языка. Первые оптимизирующие компиляторы преобразовывали код, глядя на окружающий контекст, не содержащий переходов. Иногда они удаляли вычисление, если его результаты не использовались, и, быть может, вычисляли результат более простым способом, применяя результаты предшествующих вычислений.

Первые попытки более масштабной оптимизации были предприняты в IBM, в нескольких небольших компаниях в окрестностях Бостона и в немногих академических учреждениях. Было показано, что можно добиться заметного выигрыша в производительности, если расширить область оптимизации, выйдя за рамки локального уровня. В общем случае оптимизирующий компилятор может преобразовывать промежуточное представление программы при условии существования предусловий, гарантирующих, что преобразование сохранит семантику программы. До появления формы SSA преобладали средства доказательства, строившие сводку утверждений о состоянии программы, которые должны быть истинны до и после каждого ее предложения. Доказательство конструировалось на основе понимания того, какие части состояния модифицировались предложением. Весь процесс назывался анализом потока данных. Затем результаты этого анализа использовались для преобразования промежуточного кода.

У этой схемы было несколько недостатков.

- Представление состояния не было разреженным. Для установления набора фактов требовалось решить систему уравнений, доказывающую справедливость каждого предложения в каждой точке программы. Поскольку состояние всей программы следовало представить в каждой ее точке, анализ выполнялся медленно.
- Преобразования аннулировали результаты предшествующего анализа. Хотя теоретически инкрементное обновление доказательств и было

возможно, такие алгоритмы не нашли практического применения, потому что представления не были разреженными. Поэтому каждый проход компилятора состоял из этапа анализа, за которым следовал этап преобразования.

- Для каждого типа преобразований существовал собственный набор предикатов, которые необходимо было доказать. Поскольку результаты анализа после каждого преобразования отбрасывались, не было никаких причин, побуждающих доказывать утверждения, необходимые для последующего преобразования. Это затрудняло проектирование новых преобразований, потому что прежде чем приступать к преобразованию, программисту нужно было точно установить, какая информация необходима и как построить доказательство.
- Преобразования были хрупкими. В промежуточных представлениях, доступных ранним компиляторам, для выражения зависимостей по данным между разными частями программы использовались выбранные программистом имена переменных. Проектирование любого преобразования включало разделение этих имен на значения, которые достигали каждого предложения.

Хотя работа, проделанная многими людьми, позволила повысить эффективность процесса анализа, плата за выполнение каждого вида анализа с нуля вкупе с необходимостью представлять каждый факт в каждой точке была велика. Этот процесс принципиально не масштабировался: в то время, когда мы разрабатывали форму SSA, программы становились все больше, и возникало желание сделать оптимизирующие компиляторы гораздо более агрессивными.

Многие люди пытались решить эти проблемы по отдельности. Цепочки определения-использования соединяли источник значения с его использованием, но делали это независимо от потока управления. Шапиро и Сэйнт улучшили понятие цепочки определения-использования, добавив описание точек рождения значений. Этим точкам суждено было стать местами, в которых в форме SSA появляются φ-функции. Джон Рейф использовал точки рождения в некоторых интересных видах анализа и предложил для их определения более удачные алгоритмы, чем Шапиро и Сэйнт. Но точки рождения, представленные как метод анализа исходной программы и ее преобразований, все равно требовали повторного выполнения анализа между этапами.

Разрабатывая форму SSA, мы поначалу хотели построить разреженное представление, которое позволило бы избежать необходимости доказательства каждого факта в каждой точке. Наша основная идея заключалась в том, чтобы закодировать зависимости по данным и управлению в схеме именования переменных. Мы поставили целью решить обобщенную задачу о распространении констант, которая была хорошо изучена Вегбрейтом. Мы представили свой алгоритм на 12-й конференции по принципам языков программирования (POPL). Алгоритм был гораздо эффективнее предложенного Вегбрейтом и к тому же обладал одним уникальным и весьма интересным свойством, в то время не оцененным по достоинству: преобразование не оказывало разрушающего влияния на корректность представления.

Эта первая работа была неполна. Она не охватывала семантику многих распространенных классов памяти, а алгоритм построения не был как следу-

ет продуман. Но мы вынесли неожиданный урок: путем построения единого представления, пригодного для преобразования и анализа, можно было упростить и повысить эффективность обоих.

На протяжении следующих нескольких лет мы объединились с коллегами в IBM с целью разработать не только эффективный метод перехода к форме SSA, но и набор методов, которые использовали эту форму и оставляли представление корректным по завершении. В состав этих методов вошли устранение мертвого кода, нумерация значений и перемещение инвариантного кода.

Разработанные нами оптимизации по существу очень просты, но эта простота – результат переноса сложности в SSA-представление. Аналоги этих оптимизаций в терминах потока данных сложны, и это различие не осталось незамеченным исследователями за пределами IBM. Применять оптимизации к функциональному в своих основных чертах представлению гораздо проще, чем к представлению, которому свойственны все неприглядные особенности реального языка программирования.

По современным стандартам, наша оригинальная работа была не слишком полезна: была описана лишь горстка методов, а форма SSA работала только для переменных без псевдонимов. Но в ней была определена схема выполнения преобразования языка программирования. Эта книга показывает, насколько далеко продвинулась форма SSA и что еще предстоит сделать, чтобы заменить устаревшие методы более эффективными и мощными альтернативами. Мы благодарны и польщены, что наша работа задала направление столь мощному развитию силами других исследователей.

Чаппаква, штат Нью-Йорк, США — Кеннет Задек
Йорктаун Хайтс, штат Нью-Йорк, США — Марк Н. Вегман
март 2021

Предисловие

Эту книгу можно было бы назвать по-разному: «Проектирование компилятора на основе SSA», «Инженерная разработка компилятора с помощью формы SSA» или даже «Форма с единственным статическим присваиванием на практике». Но так уж сложилось, что человек, говорящий «The SSA book», неизменно имеет в виду именно эту книгу[1].

Двенадцать лет понадобилось, чтобы родилась книга из 24 глав, написанная 31 автором. Двенадцать лет – не то чтобы мы гордились такой производительностью, но так уж вышло: в один прекрасный день опытный исследователь (Йенс Палсберг) сказал молодому (мне): «Мы должны написать книгу обо всем этом!» «Мы» включало Флорана Буше, Филипа Бриска, Себастьяна Хака, Фернандо Перейра и меня. «Все это» относилось к интереснейшему материалу по распределению регистров для SSA-переменных. На самом деле открытие, независимо совершенное группами исследователей из Калифорнии (Калифорнийский университет в Лос-Анджелесе), Германии (университет Карлсруэ) и Франции (Inria) (Ален Дартэ был одним из первых, кто стоял у истоков открытия во французской группе), было интригующим, поскольку ставило под сомнение значимость всех статей по распределению регистров, опубликованных за последние тридцать лет. То, как эти группы познакомились, само по себе достойно упоминания, поскольку без этого события наш мир, возможно, был бы другим (быть может, не так уж разительно, но точно без этой книги по SSA…): отсутствие pdf-файла с текстом доклада на симпозиуме по CGO [240] на моей веб-странице побудило Себастьяна и Фернандо независимо связаться со мной. Простой вопрос – «Только из любопытства, почему вас заинтересовала эта статья?» – стал завязкой длительного дружеского и плодотворного сотрудничества… Отсюда урок: не тратьте слишком много времени на свою веб-страницу, это может окупиться весьма неожиданным способом!

Но зачем ограничивать книгу по SSA одним лишь распределением регистров? SSA постепенно проникало в основные компиляторы, такие как LLVM, Hotspot и GCC, и побуждало многие группы разработчиков и исследователей возвращаться к различным видам анализа и оптимизации, чтобы приспособить их к этой новой форме. Лично мне повезло делать первые шаги в разработке компиляторов в период с 2000 по 2002 год [102], когда я занимался разработкой основанного на SSA компилятора уровня ассемблера в компании STMicroelectronics[2]. Благодаря SSA мне понадобилось всего четыре дня (вклю-

---

[1] Иначе говоря, эта книга отвечает критерию ссылочной прозрачности – минимальному требованию к книге о форме с единственным статическим присваиванием… Если эта шутка не кажется вам осмысленной, то вам непременно следует прочитать книгу. А если вы видите в ней смысл, то я уверен, что прочтение книги все равно поможет вам отточить свои знания.

[2] Точнее, ψ-SSA, см. главу 15.

чая два дня, ушедших на чтение соответствующей работы), чтобы разработать механизм частичного устранения избыточности, очень похожий на GVN-PRE Томаса Вандрунена [295] для предикатного кода! Учитывая мой крайне небогатый опыт, реализация алгоритма SSAPRE Фреда Чоу (*не* основанного на SSA – см. главу 11), вероятно, отняла бы у меня несколько месяцев на написание и отладку кода, хотя сам алгоритм устраняет избыточности менее эффективно. Тогда как мой проход содержал всего одну ошибку: я устранял избыточность указателя стека, потому что не знал, для чего он используется… Урок таков: если вы не знаете, что такое указатель стека, не сдавайтесь, ведь, приложив достаточно усилий, вы все же можете когда-нибудь стать членом группы по разработке компилятора.

Позже я понял, что в имеющихся книгах по компиляторам на многие вопросы нет ответов. Если для распределения регистров можно воспользоваться структурными свойствами SSA, то как насчет других низкоуровневых оптимизаций, например послепроходного планирования или выбора команд? Верно ли, что расширения для поддержки псевдонимов или предикации так же мощны, как оригинальная форма? Каковы преимущества и недостатки формы SSA? Я полагал, что в написании книги, где рассматриваются эти более широкие вопросы, должны принять участие специалисты в предметной области.

С помощью Себастьяна Хака я решил организовать семинар по SSA. Он состоялся в коммуне Отран близ Гренобля во Франции (люди старшего возраста помнят, что именно там в 1968 году проходили зимние Олимпийские игры) и собрал 55 человек (имена докладчиков выделены курсивом): *Philip Brisk*, Christoph Mallon, *Sebastian Hack*, Benoit Dupont de Dinechin, David Monniaux, *Christopher Gautier*, *Alan Mycroft*, Alex Turjan, Dmitri Cheresiz, *Michael Beck*, *Paul Biggar*, Daniel Grund, *Vivek Sarkar*, Verena Beckham, Jose Nelson Amaral, Donald Nguyen, *Kenneth Zadeck*, *James Stanier*, *Andreas Krall*, *Dibyendu Das*, Ramakrishna Upadrasta, *Jens Palsberg*, *Ondrej Lhotak*, Hervé Knochel, *Anton Ertl*, Cameron Zwarich, *Diego Novillo*, Vincent Colin de Verdière, *Massimiliano Mantione*, *Albert Cohen*, Valerie Bertin, *Sebastian Pop*, Nicolas Halbwachs, Yves Janin, Boubacar Diouf, *Jeremy Singer*, Antoniu Pop, Christian Wimmer, *Francois de Ferrière*, *Benoit Boissinot*, *Markus Schordan*, *Jens Knoop*, *Christian Bruel*, Florent Bouchez, Laurent Guerby, Benoît Robillard, *Alain Darte*, Fabrice Rastello, Thomas Heinze, *Keshav Pingali*, Christophe Guillon, *Wolfram Amme*, Quentin Colombet и Julien Le Guen.

Перед семинаром стояла двоякая цель: во-первых, убедить некоторых из хорошо известных специалистов по компиляторам подняться на вершину местной горы, а во-вторых, собрать потенциальных авторов книги и вместе подумать над тем, кто мог бы принять участие в ее написании. В конце недели некоторые участники все еще недоумевали, зачем они приняли приглашение этого сумасшедшего альпиниста (хотя до вершины добрались все), но план книги был в общих чертах намечен, а авторы всех глав определились. Было решено, что по стилю книга должна быть похожа на учебник: будет рассмотрено большинство этапов оптимизирующего компилятора, во всех главах будут употребляться единые обозначения и терминология, все необходимые понятия будут определены до использования (возможно, в другой главе). Но вместе с тем это не будет учебник в строгом понимании слова, поскольку книга предназначена только для опытных разработчиков. Тем, кто ищет хороший учебник, рекомен-

дуем, например, книгу с тигром [10] Эндрю Аппеля. А в этой книге некоторые главы лимитированы числом отведенных страниц, и понять их не всегда легко. Пол Биггар создал инфраструктуру на LaTeX, а я начал поторапливать авторов, будучи убежден, что книга выйдет в течение года… Из этого семинара я вынес урок: если альпинист предлагает вам «коротенькое» восхождение, скажите ему, что должны работать над книгой.

В этом месте один разумный человек отказался от участия в этой авантюре, и, как я подозреваю, интуиция подсказала ему, что проект чересчур амбициозный… Задача и вправду оказалась очень трудной, особенно для молодого исследователя. Написать книгу самому, пожалуй, было бы намного проще, но не так интересно. Соединить написанные независимо главы, как в обзоре современного состояния дел, также было бы не в пример легче, но тогда мы утратили бы связность изложения и не получили бы почти никакой добавленной стоимости. Поэтому каждая глава потребовала от авторов значительной переработки по сравнению с оригинальными публикациями, а главная причина, по которой все это так затянулось, заключалась в том, что крайне трудно установить жесткие сроки, когда для них нет сильных побудительных причин – в противовес необходимости представить к печати доклад на конференции. Поначалу мы думали, что шести месяцев хватит, чтобы все авторы написали свои главы, и я положил еще два месяца неспешной работы на редактуру: я хотел прочитать все главы сам, а также чтобы авторы отредактировали чужие главы. По истечении этого срока стало ясно, что мы ни при каких обстоятельствах не уложимся в намеченные временные рамки: большинство глав еще не было написано, а некоторые были просто кальками соответствующей статьи… Несмотря на это, процесс редактирования начался, и стало понятно, что редактирование должно быть очень глубоким. На самом деле чуть ли не в каждой главе присутствовал важный технический изъян; авторы давали разные определения (с небольшими, но тем не менее существенными отличиями) одного и того же понятия; кто-то расточал страницы на разработку нового формализма, хотя было достаточно уже сложившихся. Но подробная редактура одной главы занимает примерно неделю… Попробуйте отправиться на долгую прогулку по туннелю, где приходится переключаться с одной главы на другую, попробуйте дирижировать перекрестными рецензиями, умолять закончить начатые главы, молиться, чтобы состояние отсутствующих глав поменялось с «в работе» на какое-то другое, и в переписке метровой длины понукать коллег наконец-то прояснить вот это *одно* предложение из 8 слов, которое можно неправильно понять. Затем вы осознаете, что никто не сможет внести согласованные изменения вовремя, потому что у всех полно других дел (на своей настоящей работе), а когда они все-таки найдут время, не можете оторваться от других дел (на своей настоящей работе) вы сами. А тем временем двухмесячное окно закрывается, и вы понимаете, что следующее откроется только в будущем году. И время утекает, потому что у всех, включая и вас, есть что-то более срочное, чем книга, а сообщения в этот век почти мгновенной связи ведут себя так, будто ходят какими-то неведомыми путями вокруг Солнечной системы, испытывая наслаждение от растяжения времени в шестимесячном путешествии до колец Сатурна и обратно. Возвращаясь к прозе, я возлагаю вину на несовместимость между жесткими ограничениями и отсутствие чет-

кого общего срока для всех участников. И вот вам последний урок: если альпинист предлагает вам «коротенькое» восхождение, соглашайтесь: забудьте про книгу, дешевле обойдется!

Спустя несколько лет после начала проекта, будучи угнетен тем, сколько на это ушло времени, я встретил Чарльза Глезера из издательства Springer, который сказал: «У тебя еще куча времени, вот я знаю книгу, работа над которой длилась 15 лет». Когда я пишу эти строки, передо мной находится длинный список предстоящих дел, но я не хочу оказаться тем, о ком Чарльз отзывается так: «Да не парься ты, все равно не будет хуже, чем эта книга, которую они начали писать 15 лет назад, а она все еще в работе...». Уверен, что вы найдете и опечатки, и разнобой, но надеюсь, не очень много: в книге собраны знания многих талантливых экспертов по компиляторам, включая и важные неопубликованные материалы, и я рад, что она выходит сегодня.

Гренобль, Франция
февраль 2021

Фабриче Растелло

О редакторах

Фабриче Растелло – директор по исследованиям в Национальном институте исследований по информатике и автоматике (Inria) и руководитель группы CORSE (Compiler Optimization and Runtime SystEms – оптимизации компиляторов и систем времени выполнения) там же. Он занимался автоматическим распараллеливанием (докторская диссертация посвящена покрытиям как преобразованиям цикла) и оптимизации фазы синтеза компилятора (инженер группы компиляторов в компании STMicroelectronics и исследователь в Inria). Среди прочего, он выступал в роли научного руководителя нескольких соискателей звания PhD на тему переосмысления распределения регистров JIT-компилятором в свете свойств формы с единственным статическим присваиванием (SSA). Он любит задачи на пересечении теории (в основном теорий графов, алгоритмов и алгебры) и практики (промышленное внедрение). В сферу его текущих научных интересов входят (1) объединение методов времени выполнения со статической компиляцией, примером такого подхода, который он пытается продвигать, является гибридная компиляция; (2) отладка производительности средствами статического и динамического (оснащения двоичного кода) анализа и (3) пересмотр инфраструктуры компиляторов для программ специального вида.

Флоран Буше Тишаду защитил докторскую диссертацию по информатике в 2009 году в Высшей нормальной школе Лиона, где работал над компиляцией программ. Затем работал в качестве постдокторанта в Индийском научном институте (IISc) в Бангалоре. Три года он проработал в стартапе Kalray в Гренобльской коммуне во Франции. Начиная с 2013 года является доцентом в университете Гренобль-Альпы (UGA).