

Оглавление

Предисловие от издательства	12
Вступительное слово	13
Предисловие.....	16
О редакторах	20
ЧАСТЬ I. ОСНОВЫ SSA.....	21
Глава 1. Введение.....	22
1.1. Определение SSA	22
1.2. Неформальная семантика SSA	24
1.3. Сравнение с классическим анализом потока данных.....	26
1.4. SSA в контексте	28
1.5. Что будет в этой книге?	29
1.5.1. Преимущества SSA.....	29
1.5.2. Заблуждения, связанные с SSA.....	30
Глава 2. Свойства и варианты	31
2.1. Цепочки определения-использования и использования-определения....	31
2.2. Минимальность.....	33
2.3. Строгая форма SSA и свойство доминирования.....	34
2.4. Усеченная SSA-форма	36
2.5. Традиционная и преобразованная SSA-форма	37
2.6. Более сильное определение интерференции	39
2.7. Для дополнительного чтения	40
Глава 3. Стандартные алгоритмы построения и деструкции.....	41
3.1. Построение	42
3.1.1. Узловые множества и границы доминирования	42
3.1.2. Вставка ϕ -функций.....	43
3.1.3. Переименование переменных	47
3.1.4. Резюме	49
3.2. Деструкция	49
3.3. Преобразования свойств SSA	52
3.3.1. Пессимистическая вставка ϕ -функций	54
3.4. Для дополнительного чтения.....	56
Глава 4. Продвинутое построение формы SSA.....	58
4.1. Базовый алгоритм	58
4.2. Вычисление $DF^*(S)$ с помощью DJ-графов.....	59
4.2.1. Ключевые наблюдения	59
4.2.2. Основной алгоритм	61

4.3. Вычисление потока данных для DF^+ -графа с помощью DJ-графа	63
4.3.1. Нисходящее вычисление множества DF^+	63
4.4. Вычисление итерированной границы доминирования с помощью леса вложенных циклов.....	66
4.4.1. Лес вложенных циклов	66
4.4.2. Основной алгоритм	66
4.5. Заключительные замечания и дополнительная литература	69
Глава 5. Реконструкция SSA.....	71
5.1. Общие соображения	72
5.2. Реконструкция, основанная на границе доминирования.....	74
5.3. Реконструкция на основе поиска.....	75
5.4. Для дополнительного чтения.....	78
Глава 6. Функциональные представления SSA.....	79
6.1. Низкоуровневые функциональные представления программ.....	80
6.1.1. Присваивание и привязка переменной	81
6.1.2. Поток управления: продолжения	82
6.1.3. Поток управления: прямой стиль	85
6.1.4. Let-нормальная форма	87
6.2. Функциональное построение и деструкция SSA	89
6.2.1. Начальное построение с использованием анализа живучести	89
6.2.2. λ -отбрасывание	90
6.2.3. Вложенность, доминирование, замыкание цикла.....	94
6.2.4. Деструкция SSA	97
6.3. Улучшенное опускание блока и леса вложенных циклов	97
6.4. Дополнительная литература и заключительные замечания	101
ЧАСТЬ II. АНАЛИЗ	105
Глава 7. Введение	106
Глава 8. Распространение информации с использованием формы SSA.....	109
8.1. Предварительные сведения	110
8.1.1. Решения задач о потоке данных.....	111
8.2. Распространение потока данных в форме SSA.....	112
8.2.1. Представление программы	112
8.2.2. Разреженное представление потока данных	113
8.2.3. Обсуждение	116
8.3. Пример – распространение копирований	117
8.4. Для дополнительного чтения.....	119
Глава 9. Живучесть.....	121
9.1. Определения	122
9.2. Подходы к анализу потока данных.....	124
9.2.1. Множества живучести для приводимых графов.....	124

9.2.2. Множества живучести для неприводимых графов потока	128
9.2.3. Вычисление самого внешнего исключающего цикла (OLE)	129
9.3. Проверка живучести с помощью леса вложенных циклов и прямой достижимости	131
9.3.1. Вычисление модифицированной прямой достижимости	133
9.4. Вычисление множеств живучести с помощью исследования пути.....	134
9.5. Для дополнительного чтения.....	135
Глава 10. Дерево циклов и индуктивные переменные.....	138
10.1. Часть ГПУ и дерево циклов могут быть раскрыты с помощью SSA.....	138
10.1.1. SSA-представление без ГПУ	139
10.1.2. Естественные структуры циклов в форме SSA.....	140
10.1.3. Улучшение форматированной печати SSA для циклов.....	141
10.2. Анализ индуктивных переменных	142
10.2.1. Определение шага.....	142
10.2.2. Трансляция цепочек рекурренций	143
10.2.3. Инстанцирование символов и региональных параметров.....	144
10.2.4. Число итераций и вычисление значения в конце цикла	144
10.3. Для дополнительного чтения.....	146
Глава 11. Устранение избыточности	148
11.1. Почему устранение частичной избыточности и форма SSA взаимосвязаны	149
11.2. Как работает SSAPRE.....	152
11.2.1. Условие безопасности	153
11.2.2. Условие вычислительной сложности	154
11.2.3. Условие оптимальности времени жизни.....	155
11.3. Гипотетическое PRE	156
11.4. Перемещение в регистры с помощью PRE	158
11.4.1. Перемещение в регистры как оптимизация размещения	159
11.4.2. Оптимизация размещения операций загрузки	160
11.4.3. Оптимизация размещения операций сохранения	161
11.5. Устранение избыточности на основе анализа значений.....	162
11.5.1. Нумерация значений	163
11.5.2. Исключение избыточности с использованием нумерации значений.....	165
11.6. Для дополнительного чтения.....	165
ЧАСТЬ III. РАСШИРЕНИЯ	169
Глава 12. Введение	170
12.1. Форма с единственной статической информацией	170
12.2. Зависимости по управлению	171
12.3. Вентильные формы SSA.....	172
12.4. Форма ψ -SSA.....	172

12.5. Хешированная форма SSA.....	173
12.6. Форма SSA для массивов	174
12.7. Поток данных, основанный на памяти	175

Глава 13. Форма с единственной статической информацией 177

13.1. Единственная статическая информация.....	178
13.1.1. Разреженный анализ.....	178
13.1.2. Задачи типа решетки разбиений для каждой переменной (PLV) ...	180
13.1.3. Свойство единственной статической информации	181
13.1.4. Специальные команды для расщепления интервалов жизни.....	182
13.1.5. Распространение информации вперед и назад.....	183
13.1.6. Примеры разреженного анализа потока данных	186
13.2. Построение и деструкция промежуточного представления программы	189
13.2.1. Стратегия расщепления.....	189
13.2.2. Расщепление интервалов жизни	190
13.2.3. Переименование переменных	192
13.2.4. Устранение мертвого и неопределенного кода	193
13.2.5. Детали реализации	194
13.3. Для дополнительного чтения.....	195

Глава 14. Графы и вентильные функции 198

14.1. Графы потока данных	198
14.2. SSA-граф	199
14.2.1. Нахождение индуктивных переменных с помощью SSA-графа.....	200
14.3. Граф зависимостей программы	201
14.3.1. Обнаружение возможностей распараллеливания с помощью ГЗП	203
14.4. Вентильные функции и форма GSA.....	204
14.4.1. Обратный символический анализ с помощью GSA.....	206
14.5. Граф зависимостей состояний и значений	207
14.5.1. Определение ГЗСЗ.....	208
14.5.2. Устранение мертвых вершин ГЗСЗ	211
14.6. Для дополнительного чтения.....	212

Глава 15. Форма ψ -SSA..... 214

15.1. Определение и построение	215
15.2. Алгоритмы для работы с формой SSA	217
15.3. Алгоритмы для работы с формой ψ -SSA	217
15.4. Деструкция ψ -SSA	219
15.4.1. ψ -нормализация	221
15.4.2. ψ -сеть.....	222
15.5. Для дополнительного чтения.....	225

Глава 16. Хешированная форма SSA: HSSA..... 226

16.1. SSA и псевдонимия: μ - и χ -функции.....	227
16.2. Введение «нулевых версий» для ограничения комбинаторного роста числа версий	228

16.3. Форма SSA для косвенных операций с памятью: виртуальные переменные.....	230
16.4. Использование ГНЗ для построения формы HSSA	233
16.5. Построение формы HSSA.....	235
16.6. Для дополнительного чтения.....	237
Глава 17. Форма SSA для массивов.....	240
17.1. Форма SSA для массивов.....	241
17.2. Разреженное распространение констант для элементов массивов....	243
17.2.1. Решетка массива для разреженного распространения констант....	243
17.2.2. Неконстантные индексы.....	246
17.3. Расширение для объектов: устранение избыточных операций загрузки.....	248
17.3.1. Каркас анализа	249
17.3.2. Анализ заведомо совпадающих и заведомо различных значений для индексов массива в куче	249
17.3.3. Алгоритм поднятия	251
17.4. Для дополнительного чтения	252
ЧАСТЬ IV. ГЕНЕРИРОВАНИЕ И ОПТИМИЗАЦИЯ МАШИННОГО КОДА	255
Глава 18. Форма SSA и генерирование кода.....	256
18.1. Инженерные проблемы формы SSA	258
18.1.1. Команды, операнды, операции и операторы.....	258
18.1.2. Представление семантики команд	258
18.1.3. Ограничения на имена операндов	259
18.1.4. Неуничтожаемые операнды-приемники	260
18.1.5. Инварианты представления программы	261
18.2. Этапы генерирования кода и форма SSA	262
18.2.1. Преобразование if	262
18.2.2. Предварительное планирование команд.....	266
18.3. Алгоритмы деструкции формы SSA.....	268
Глава 19. Выбор команд	271
19.1. Выбор команд для деревьев-образцов на SSA-графах	275
19.1.1. Булева квадратичная задача с разделением	275
19.1.2. Выбор команд как задача PBQP	276
19.2. Расширения и обобщения	278
19.2.1. Выбор кода для ОАГ-образцов.....	278
19.3. Заключительные замечания и литература для дополнительного чтения.....	282
Глава 20. Преобразование if	283
20.1. Архитектурные требования.....	285
20.2. Базовые преобразования.....	286
20.2.1. Операции с простыми блоками	286
20.2.2. Работа с моделью предикатного выполнения	289

20.3. Глобальный анализ и преобразования	292
20.3.1. Инкрементный алгоритм преобразования If для формы SSA	292
20.3.2. Дублирование хвоста	293
20.3.3. Целесообразность.....	294
20.4. Заключительные замечания и литература для дополнительного чтения.....	297

Глава 21. Деструкция формы SSA для машинного кода 298

21.1. Корректность	300
21.2. Качество кода	305
21.3. Скорость и потребление памяти.....	308
21.4. Для дополнительного чтения.....	315

Глава 22. Распределение регистров 317

22.1. Введение	317
22.1.1. Вопросы, стоящие перед распределителем регистров.....	318
22.1.2. Maxlive и расщепление переменных	321
22.1.3. Тест выгрузки для формы SSA.....	322
22.2. Выгрузка	324
22.2.1. Графовый подход	325
22.2.2. Подход на основе сканирования.....	327
22.3. Раскрашивание и консолидация.....	333
22.3.1. Жадная схема раскрашивания	333
22.3.2. Консолидация в форме SSA	336
22.3.3. Графовый подход	336
22.3.4. Подход на основе сканирования.....	339
22.4. Обсуждение практических вопросов и литература для дополнительного чтения.....	339

Глава 23. Аппаратная компиляция в форме SSA 344

23.1. Краткая история и обзор	344
23.2. Зачем использовать форму SSA для аппаратной компиляции?.....	349
23.3. Отображение графа потока управления на оборудование	350
23.3.1. Отображение простого блока	350
23.3.2. Отображение простого графа потока управления.....	351
23.3.3. Отображение графа потока управления с использованием формы SSA	353
23.3.4. ϕ -функция и оптимизации мультиплексов	356
23.3.5. Влияние формы SSA на компоновочный план	358
23.4. Существующие примеры аппаратной компиляции на основе SSA ...	359

Глава 24. Построение формы SSA в компиляторе РНР 362

24.1. Форма SSA в статически типизированных языках.....	363
24.2. РНР и псевдонимия.....	364
24.3. Наш общепрограммный анализ.....	366
24.3.1. Алгоритм анализа	366

24.3.2. Результаты анализа.....	367
24.3.3. Построение множеств определения-использования	367
24.3.4. Форма HSSA	368
24.4. Другие проблематичные средства РНР	369
24.4.1. Таблицы символов на этапе выполнения	369
24.4.2. Исполнение произвольного кода.....	370
24.4.3. Обработчики объектов.....	370
24.5. Заключительные замечания и литература для дополнительного чтения.....	371
Литература	373
Предметный указатель.....	388

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 приступить 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, сделанная 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 Сэйтн 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 Сэйтн. 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. Аналогии 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.

Чапаква, штат Нью-Йорк, США
Йорктаун Хайтс, штат Нью-Йорк, США
март 2021

Кеннет Задек
Марк Н. Вегман

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 книгу¹.

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 STMicroelectronics². 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 слепопроходного 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 Halbwegs*, *Yves Janin*, *Boubacar Diouf*, *Jeremy Singer*, *Antoni Pop*, *Christian Wimmer*, *Francois de Ferrière*, *Benoit Boissinot*, *Markus Schordan*, *Jens Knoop*, *Christian Bruehl*, *Florent Bouchez*, *Laurent Guerby*, *Benoît Robillard*, *Alain Darté*, *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-

дую, 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 редакция 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 сфере 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).

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru