Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Синтаксический анализатор языка C

Курсовая работа студента 345 группы

Авдюхина Дмитрия Алексеевича

Научный руководитель ..... Я. А. Кириленко

ст. преподаватель /подпись/

Санкт-Петербург

2011

Содержание

1 Введение

Реинжиниринг программного обеспечения [1] включает в себя целый комплекс задач по оценке, изучению и модификации исходного кода. Одна из этих задач – нахождение участков кода, в которых нарушены best-practices – рекомендации и требования по написанию, признанные оптимальными для конкретного языка программирования. Эти участки принимаются компилятором, но могут стать причиной ошибок во время выполнения, либо, даже работая корректно, отрицательно повлиять на переносимость программы. Современные инструменты, найдя их, могут выводить предупреждение, тем самым обращая на них внимание разработчика. Они также могут предоставлять возможность автоматического исправления программ, что также играет важную роль в реинжиниринге, поскольку позволяет выполнить много рутинной работы без участия человека.

Статические анализаторы могут вычислять различные метрики – количественные показатели, отражающие некоторые характеристики программы. В качестве примеров можно привести цикломатическую сложность, различные метрики размера программ и многие другие. Знание этих величин помогает уделить больше внимания самым сложным участкам программы, упростить тестирование и облегчить процесс оптимизации, выявляя наиболее трудоемкие по времени исполнения участки. Метрики также во многом позволяют оценить качество продукта, что играет важную роль в реинжиниринге при оценке стоимости изменения программы.

Существует множество инструментов, используемых в реинжиниринге, как с открытым исходным кодом, так и коммерческих, но большинство имеет узкую специализацию, что и послужило мотивацией к работе над собственным проектом, который планируется сначала создавать как статический анализатор языка C, а затем, отработав определенную технику, распространить на другие языки. Разрабатываемый инструмент должен обеспечивать комплексный подход, решая большинство описанных задач, и быть достаточно универсальным, чтобы применяться для многих языков.

2 Постановка задачи

В рамках данного проекта реализуется синтаксический анализатор языка C с расширенным внутренним представлением исходного кода, позволяющим хранить всю историю макро-подстановок.

В данной работе были поставлены следующие задачи:

- Портировать синтаксический анализатор языка C из проекта CIL на платформу Microsoft .NET. Полученный анализатор должен строить дерево разбора для препроцессированного файла.
- Апробировать инструмент YaccConstructor [6], разрабатываемый группой студентов кафедры системного программирования Математико-Механического факультета СПбГУ. В работе он должен использоваться для трансляции описания грамматик на различные языки и для создания исполняемого модуля по полученной грамматике.

3 Обзор существующих решений

Рассмотрим некоторые инструменты и задачи, которые они решают:

- Есть много инструментов, занимающихся автоматическим форматированием текста (например, Artistic Style [5], Indent [9]), но эта функциональность очень ограничена, для ее реализации обычно требуется минимальная часть синтаксиса языка, и для ее расширения на статический анализ требуется много работы.

- Также есть инструменты, которые могут проводить хороший анализ кода для определенного языка (например, Splint[15], PREFast [11] для C), но обычно их тяжело переносить на другие языки.

- Есть более универсальные инструменты, такие как FxCop [10], который способен проводить анализ любого кода под .NET, но не предоставляет возможности автоматического исправления. Для языка не под .NET он уже не предназначен, кроме того, не может работать с некорректным входом, поскольку анализирует уже откомпилированные программы.

- Есть коммерческие продукты (например, DMS [14]), возможности которых очень велики, но в данном случае ставится задача написания именно open-source инструмента.

- В существующих инструментах при подсчете метрик могут поразному трактоваться определения и сильно различаться результаты. Например, при подсчете цикломатической сложности для больших проектов ответы могут отличаться на 20%. Существуют различные модификации этой метрики, дающие разные результаты, и не всегда задокументировано, как именно она вычисляется конкретным инструментом.

- Большинство оптимизирующих компиляторов могут выполнять полезные дополнительные статические проверки на этапе трансляции, но это их не основная, а вспомогательная функциональность, и их внутренние структуры оптимизированы на представление кода для

транслятора. Как следствие, не сохраняется важная для статического анализа информация, например, пропадает возможность вычислять некоторые метрики размера программ и теряется связь полученного кода и исходного.

- В случае языка C связь с исходным текстом частично исчезает уже после препроцессирования, поэтому большинство инструментов не позволяют находить в коде неименованные константы, поскольку не могут определить, появились ли они в результате макро-подстановки, либо были написаны в коде изначально. Что важно, потеря точной привязки синтаксических элементов к исходному тексту делает невозможным автоматическое изменение кода.

  Разрабатываемый инструмент должен быть лишен этого недостатка, поскольку благодаря сохранению истории макро-подстановок всегда есть возможность восстановить исходный код.

Как видно из приведенных примеров, большинство инструментов недостаточно универсальны, что послужило причиной для создания собственного инструмента.

Так как существует много синтаксических анализаторов языка C, в том числе и с открытым исходным кодом, то было решено не писать код с нуля, а перенести на .NET какой-либо из них. Нами был выбран проект CIL [13] (C Intermediate Language), написанный на языке OCaml, в котором реализован и лексический, и синтаксический анализ. Таким образом, основной задачей стало портирование грамматики на CIL под .NET.

4 Реализация

4.1 Окружение

Предполагалось, что основным инструментом разработки будет YaccConstructor – генератор синтаксических анализаторов. Он является достаточно удобным, поскольку в нем реализованы многие синтаксические конструкции [4]. Также он имеет модульную структуру [3], реализацией которой занимается Константин Улитин из 545 группы.

Для работы с инструментом необходимы:

- Парсер входной грамматики (фронтенд). Он преобразуют грамматику языка, записанную в некотором, конкретном для каждого фронтенда формате во внутреннее представление. Именно над этим представлением производятся все преобразования грамматики в инструменте.

- Генератор (бэкенд), при помощи которого можно преобразовывать внутреннее представление в некоторый выходной формат. Например, можно создать исполняемые файлы на основе входной грамматики, что можно выполнять при помощи RACC-генератора [2], которым занимается Семен Григорьев из 561 группы. Исполняемым файлам будет на вход подаваться программа на целевом языке, и они будут строить его дерево разбора.

  Есть возможность реализовать бэкенд, называемый Printer, который на основе внутреннего представления должен вернуть исходную грамматику на определенном языке. Для каждого языка необходима своя реализация этого бэкенда. При помощи него можно переводить грамматику из одного языка в другой, чем предполагалось воспользоваться в ходе курсовой.

В качестве основного языка разработки был выбран F# – разрабатываемый в Microsoft Research мультипарадигмальный язык, на котором был написан YaccConstructor и для которого существуют инструменты FParsec, FsLex и FsYacc, входящие в комплект F# PowerPack, и нашедшие непосредственное применение в данной работе. FParsec [16] – библиотека комбинаторов парсеров, созданная под F#. FsLex [7] – генератор лексических анализаторов, в котором грамматики описываются в той же форме, что и в OCamlLex. FsYacc [8] – LALR генератор синтаксических анализаторов, имеющий спецификацию, схожую с OCamlYacc.

Что важно, язык OCaml, на котором создавался CIL, во многом совместим с такими инструментами, как FsLex и FsYacc, что позволяет по большей части использовать уже написанный код практически без модификации.

Проект создается под платформу Microsoft .NET, поэтому его можно интегрировать в другие проекты под .NET и, возможно, когда он разовьется до статического анализатора, как расширение в Microsoft Visual Studio.

4.2 Разработка

Проект состоит из двух частей – лексера и парсера. Лексер по большей части он был взят из CIL и перенесен под FsLex с незначительными изменениями, связанными с отличиями языков OCaml и F#. Также пришлось изменить некоторые конструкции языка из OCaml на более эффективные эквиваленты языка F#. Например, вместо обычной конкатенации строк был использован StringBuilder, что увеличило скорость токенизации.

Для разработки парсера планировалось использование инструмента YaccConstructor, который должен осуществлять трансляции описания грамматик на нужные языки. Предполагалось, что сама грамматика будет написана на YARD [4], поскольку он обладает достаточно широкими возможностями для описания грамматик, например, EBNF, поддержкой макроправил и атрибутов. Эти преимущества должны оказаться полезными в дальнейшем развитии инструмента и, кроме того, позволить уменьшить размер кода описания.

Для получения грамматики на YARD-е был взят парсер из CIL, напи-

санный под OcamlYacc, адаптирован под FsYacc (также с незначительными изменениями). Затем над полученным файлом запустили YaccConstructor с фронтендом FsYaccFrontend, принимающим грамматику на языке FsYacc, и бэкендом YARDPrinter, выводящим описание грамматики на YARD-е. Тем самым, было получено желаемое описание грамматики на языке YARD.

Следующей задачей стало создание по полученной грамматике исполняемого модуля, который принимает на вход поток лексем, и возвращает дерево разбора исходного файла. Для этого также использовался инструмент YaccConstructor с фронтендом, разбирающим грамматику на YARD-е. Были рассмотрены разные бэкенды и сделаны следующие попытки преобразования грамматики:

- FParsecGenerator – переводит грамматику в код для инструмента FParsec, что позволяет включить его в проект на языке F#. Основной проблемой стало то, что FParsec предоставляет развитые средства для описания трансляции над файлами в текстовом формате, но приспособить его для того, чтобы он принимал на вход поток лексем, оказалось трудно, и в рамках данного проекта было решено, что этот подход неоправданно сложен.

- RACCGenerator (Recursive-Ascent Compilers Compiler) [2] – генератор трансляторов для неоднозначных контекстно-свободных грамматик с непосредственной поддержкой EBNF. В процессе тестирования выяснилось, что полученный анализатор недостаточно эффективно справлялся с большими файлами, из-за чего от него пришлось отказаться.

- FsYaccGenerator – переводит грамматику в код для FsYacc, тем самым после работы инструмента FsYacc над полученным кодом также можно сгенерировать файл, который можно включить в проект. Данный подход не удался, поскольку YARD – бестиповый язык, поэтому при преобразовании из FsYacc в YARD теряется информация о типах. При обратном преобразовании типы нужно восстанавливать, что на данный момент не было реализовано.

Таким образом, было решено на данный момент остановиться на грамматике на FsYacc.

5 Заключение

В ходе выполнения данной курсовой работы были достигнуты следующие результаты:

- Получен синтаксический анализатор под платформу .NET, перенесенный из CIL. В случае, если поданный ему на вход файл корректен, будет построено его дерево разбора. Иначе программа сгенерирует исключение и завершит свою работу с сообщением об ошибке.

  Также как и CIL, анализатор способен обрабатывать программы не только на чистом ANSI-C, но и на C с более широким синтаксисом, например Microsoft C или GNU C.

- Выявлены некоторые недостатки и ошибки в работе YaccConstructor, о которых были уведомлены разработчики, тем самым предложены пути улучшения инструмента.

- Получен опыт работы с языками F# и OCaml.

- Изучены следующие инструменты:

  - FsLex
  - FsYacc
  - FParsec
  - YaccConstructor

- Работа была представлена на конкурсе-конференции “Технологии Microsoft в теории и практике программирования”, Тезисы доклада были опубликованы в сборнике конференции, работа получила диплом третьей степени.

Дальнейшим развитием работы будет осуществление семантического анализа над получившимся деревом разбора.

Исходный код проекта можно найти на сайте http://code.google.com/p/claret/. Основной результат на момент написания данного текста соответствует ревизии 69. Автор работы принимал участие в проекте под учетной записью dimonbv.

Список литературы

[1] Автоматизированный реинжиниринг программ / Под ред. проф. А.Н. Терехова и А.А. Терехова. — СПб.: Издательство С.-Петербургского университета, 2000 — 332 с.

[2] Григорьев С.В. Генератор синтаксических анализаторов для неоднозначных контекстно-свободных грамматик, 2010. 49 с. (http://recursive-ascent.googlecode.com/files/Semen_Grigorev_Diploma.pdf)

[3] Улитин К.А. Разработка архитектуры для генератора синтаксических анализаторов, 2010. 15 с. (http://recursive-ascent.googlecode.com/files/KonstantinUlitin_CompilerCompilerArchitecture.pdf)

[4] Чемоданов И.С. Генератор синтаксических анализаторов для решения задач автоматизированного реинжиниринга программ, 2007. 37 с. (http://recursive-ascent.googlecode.com/files/IlyaChemodanov_Yard.pdf)

[5] http://astyle.sourceforge.net/ (сайт инструмента Artistic Style)

[6] http://code.google.com/p/recursive-ascent/ (сайт разработки инструмента YaccConstructor)

[7] `http://fsharppowerpack.codeplex.com/wikipage?title=FsLexDocumentation` (описание FsLex)

[8] http://fsharppowerpack.codeplex.com/wikipage?title=FsYacc (пример использования FsYacc)

http://cs.wellesley.edu/~cs301/htmlman-3.06/manual026.html (описание синтаксиса FsYacc)

[9] http://linux.maruhn.com/sec/indent.html (инструмент Indent)

http://www.gnu.org/software/indent/manual/indent.html (описание его работы)

[10] http://msdn.microsoft.com/en-us/library/bb429476%28v=vs.80%29.aspx (описание работы с FxCop)

[11] http://msdn.microsoft.com/en-us/library/ms933794.aspx (описание работы с PREFast)

[12] http://msdn.microsoft.com/fsharp/ (центр разработки Microsoft F#)

[13] http://www.cs.berkeley.edu/~necula/cil/ (CIL – документация)

[14] http://www.semdesigns.com/Products/DMS/ (сайт инструмента DMS)

[15] http://www.splint.org/ (сайт инструмента Splint)

[16] http://www.quanttec.com/fparsec/ (описание и руководство FParsec)