Воронежский Государственный Университет

Пахомов Александр Сергеевич

Разработка лексического и синтаксического анализаторов с целью подсветки синтаксиса и автодополнения исходного кода для предоставленного языка

Специальность 02.03.03 —
«Математическое обеспечение и администрирование информационных систем»

Научный руководитель:
уч. степень, уч. звание
Огаркова Наталья Владимировна

Воронеж — 2019

Оглавление

Стр.

Введение

После первоначального анализа источников напишу.

Объем и структура работы. Полный объём дипломной работы составляет 18 страниц, включая 0 рисунков и 0 таблиц. Список литературы содержит 62 наименования.

Глава 1. Постановка задачи

Изучить существующие подходы к лексическому и синтаксическому анализу языков программирования с контекстно-свободной грамматикой и динамической компиляцией, разработать лексический и синтаксический анализаторы с целью подсветки синтаксиса и автодополнения исходного кода для предоставленного языка (предметно ориентированного).

Глава 2. Анализ задачи

2.1 Анализ существующих подходов и инструментов

Тут будет обзор всех существующих инструментов и подходов. [1—62]

2.1.1 Относительная эффективность разпознаваний контекстно-свободных грамматик

В статье [41] приводятся основные понятия, относящиеся к контекстно-свободным граматикам, и процедуры их распознавания. Так же авторы приводят сравнительный анализ эффективности различных подходов, путем замеров скорости компиляций программ.

Статья содержит полезную информацию по теме, но в силу винтажности (1965 год) языков и сильной погруженности авторов в непосредственное сравнение, вынести из нее можно разве что только базовые понятия

2.1.2 Семантика контекстно-свободных языков

В статье [31] приводятся базовые определения семантики контекстно-свободных языков, дается определение семантической и синтаксической однозначности граматик.

Статья состоит из пяти разделов, в разделе 1 описывается «значение» языков, определённых с помощью контекстно-свободных граматик. Математические основы описываются в 2 и 3 разделах. В 4 разделе описано формальное определение языка программирования с помощью контекстно-свободных граматик. Поледняя секция содержит сравнение основных подходов определения языков программирования с помощью контекство-свободных граматик.

Классика, (Кнут!), обязательно провести более глубокий анализ

2.1.3 Эффективный алгоритм разбора контекстно-свободных грамматик

В статье [47] описывается «Эффективный алгоритм» разбора контекстно-свободных грамматик Джея Йорлей и его сравнение с другими алгоритмами (Кнута, Гриффинса и Патрика).

В разделе 2 приведены основные термины статьи. В 3 и 4 разделах содержится описание алгоритма, 5 раздел описывает эффективность алгоритма. В 6 разделе приведены эмпирические сравнения алгоритма с другими. 7 раздел содержит описание практического применения алгоритма.

Полезная информация, думаю, стоит включить

2.1.4 Трансляции контекстно свободных граматик

В вырезке из книги [61] содержатся математические основы методов трансляции контекстно-свободных граматик. Приведены теоремы и леммы (с доказательствами) корректности таких методов.

Стоит рассматривать глубже только в случае необходимости приведения теорем и лемм. Материал сухой и сугубо научный.

2.1.5 Yacc, статья Стефана Джонсона

Статья «Еще один компилятор-компилятор» [35] описывает Yacc. Yacc — это инструмент, предоставляющий пользователю возможность описать входящие структуры, а затем связать их с некоторыми действиями. По сути, Yacc является мощным инструментом разбора входящего потока строк, который потом предпринимает опредеоленные пользователем действия в случае совпадения некоторых правил, которы так же определены пользователем.

Статья достаточно полно покрывает основные возможности Yacc, а так же приводит примеры их использования.

Yacc в любом случаю должен быть подробно разобран, статья хорошая, но пока оставлю ее на первом уровне, так как ожидаю найти более полный материал

2.1.6 Обобщение регулярных множеств и их прикладные значения для изучения контекстно-свободных языков

Научная работа [10] Масакко Такаши проводит глубокий анализ регулярных множеств строк. В том числе деревьев и лесов, что бы это не значило. В первую очередь автор приводит ряд базовых определений и теорем (с доказательством) для регулярных выражений строк, и разбирает (опять же с математической точки зрения) их прикладное значение для языков с контекстно-свободным определением.

Я считаю, что это не то, что мне нужно, так как базовые определения даны в других источниках, а такие вещи, как Эс-локальные множества, наврятли мне понадобятся.

2.1.7 Алгоритма трансляции промежуточного представления кода программы в код ассемблера или в машинный код

Статья [42] описание алгоритма трансляции промежуточного представления кода программы (intermediate representation) в код ассемблера или в машинный код. Метод умеет транслировать в большинство машинного кода того времени (1978), и достаточно эффективен. Для смены целевого компьютера, под который генерируется машинный код не требется особых усилий (для того времени это достижение), хотя в конце статьи авторы говорят, что после трансляции промежуточного кода в код ассемблера под PDP-11, им понадобилось около часа, что бы настроить алгоритм на трансляцию того же кода в машинный код под IBM 370.

В силу винтажности статьи и довольно ограниченного материала, не стоит включать статью в основной список литературы.

2.1.8

Статья [1] очень абстракно и верхнеуровнево описывает архитектуру компьютеров и компиляторов. Значительная часть статьи посвещена «большим темпам роста» промышленности в сфере комьютеров и программ того времени (1981), а так же тому, как правильно писать софт.

Совсем не то что нужно, ссылаться там не на что.

2.1.9 Советы по написанию компиляторов и доказательство их корректности

Статья [7] описывает способ написания компилятора, который гарантированно является корректным с точки зрения математики. Авторы приводят 18 аргументов, которые доказывают абсолютную корректность алгоритма. «Кореректность» означает, что существует однозначная и связь исходного кода с машинным (скомпилированным), и она справедлива с точки зрения логики.

В статье приведен мини-язык, на примере которого показывается корректность компилятора.

Много математических терминов, основа почти не дана. Статья является очень узкой (авторы описывают корректность компилятора, которы был озвучен на какой-то там конференции самим Моррисом(кто бы это не был)). Считаю, что дальше статью рассматривать не стоит.

2.1.10 Семантико-направленный генератор компиляторов

Статья [20] содержит описание генератора компиляторов, который Паульсон написал на Паскале. На вход он получает семантические грамматики, описанные в БНФ, а на выходе получается компилятор. Сам генератор состоит из трёх частей — это анализатор грамматик, универсальный транслятор и стэк машина.

В качестве демонстрации работы генератора автор приводит описание самого языка Паскаль в БНФ и генерацию компилятора на её основе. В конце

приводятся результаты сравнения скорости итоговых программ и их компиляций с помошью стандартного компилятора Паскаля и сгенерированного, где второй, очевидно, значимо проигрывает.

Интересная статья, но практического опыта из нее извлечь трудно, да и материал не новый. Думаю что не стоит ее разбирать глубже.

2.1.11 Эффективный алгоритм разбора контекстно-свободных грамматик

Статья [46] является описанием алгоритма, разработанного в 1968г Джеем Орлей. В относительно небольшом объеме разобран алгоритм, основные термины и сравнения производительности. В конце приводятся приемущества над другими алгоритмами тех лет.

Классика, нужно разобрать подробнее. Но необходимо выбрать материал между настоящим и 2.1.3. Статья [46] мне кажется более доступной для понимания.

2.1.12 Практичный инструмент для создания портативных компиляторов

Статья [9] описывает «Амстердамский инструмент для компиляторов» который решает проблему написания $N \times M$ программ. То есть для интерпритации и компиляции N языков программирования на M разных компьютерных архитектурах, требуется $N \times M$ программ. «Амстердамский инструмент для компиляторов» решает эту проблему путем написания N программ, которые транслируют N языков в единое промежуточное представление и M программ, которые транслируют единое промежуточное представление в язык ассемблера для каждой из M архитектур. Таким образом нужно написать всего $N + M$ программ. Для поддержания нового языка на всех архитектурах требуется написать всего 1 программу, также как и для поддержания всех языков новой архитектурой.

Стоит отметить, что задача $N \times M$ программ совсем не простая, и для ее решения авторам пришлось пойти на некоторые уступки, а именно, они ограничились только алгебраическими языками и 8-ми битными архитектурами.

Очень полезная статья. Похоже, что это «дедушка» всех современных инструментов, нужно изучить.

2.1.13 Написание YACC на Sasl

Статья [34] содержит описание процесса написания YACC на функциональном языке программирования Sasl. Автор приводит сравнение имперического подхода к программированию и функционального путем написания «средней» программы.

В результате получиласть программа в 2 раза меньшая в объеме, чем такая же, но написанная на процедурном языке.

Очень интересно окунуться вглубь и посмотреть реализацию, но тема статьи не пересекается с темой дипломной работы. Разве что YACC, но совсем с другой стороны.

2.1.14 Эффективный контекстно-свободный алгоритм для разбора естественных языков

Статья [6] описывает алгоритм разбора естественных языков. Автор описывает существующие на то время алгоритмы разбора языков и объясняет почему они не годятся для разбора естественных языков.Он разделяет все алгоритмы на 2 группы:

1. Алгоритмы разбора языков программирования.
2. Алгоритмы разбора общих контексно свободных языков.

Обе группы, по словам автора, не подходят для разбора естественных языков. Первые — ограниченны слишком маленьким набором грамматик, вторые — слишком громоздкие для естественных языков, которые ближе к языкам программирования, а не к контекстно-свободным грамматикам.

Автор предлагает свой алгоритм, который находится между алгоримами разбора языков программирования и алгоритмами разбора контекстно-свободных языков. И подробно его описывает.

Будет полезно для дипломной работы, стоит изучить статью.

2.1.15 Компиляторы и поэтапные преобразования

Статья [33] содержит описание техник преобразования кода. А именно, «предкомпиляция»(precomputation) и «сокращение частотности»(frequency reduction), обощенно их можно назвать «поэтапные преобразования». Эти два приема автор демонстрирует на простых примерах, а также на фрагмете компиляции Пролога.

Мне кажется, это шаг в сторону инкрементальной компиляции. Наверно, было бы полезно разобраться в этом. Добавлю статью в список основных, если не найду тот же материал, но более доступным языком. Мне показалось, что статья немного непонятная для меня.

Заключение

Основные результаты работы заключаются в следующем.

1. На основе анализа …
2. Численные исследования показали, что …
3. Математическое моделирование показало …
4. Для выполнения поставленных задач был создан …

И какая-нибудь заключающая фраза.

Список литературы

1. *Wulf, W. A.* Compilers and computer architecture / W. A. Wulf // Computer. — 1981. — № 7. — С. 41—47.
2. *Wimmer, C.* One VM to rule them all / C. Wimmer, C. Seaton // In Proceedings of the JVM Language Summit. — 2013.
3. *Wimmer, C.* Truffle: a self-optimizing runtime system / C. Wimmer, T. Würthinger. — 2012.
4. *Vergu, V.* Specializing a meta-interpreter: JIT compilation of dynsem specifications on the graal VM / V. Vergu, E. Visser. — 2018.
5. *Van Deursen, A.* Domain-specific languages: An annotated bibliography / A. Van Deursen, P. Klint, J. Visser // ACM Sigplan Notices. — 2000. — Т. 35, № 6. — С. 26—36.
6. *Tomita, M.* An Efficient Context-Free Parsing Algorithm for Natural Languages. / M. Tomita. — 1985.
7. *Thatcher, J. W.* More on advice on structuring compilers and proving them correct / J. W. Thatcher, E. G. Wagner, J. B. Wright // Theoretical Computer Science. — 1981. — Т. 15, № 3. — С. 223—249.
8. *Temkin, J. M.* Extraction of protein interaction information from unstructured text using a context-free grammar / J. M. Temkin, M. R. Gilder // Bioinformatics. — 2003. — Т. 19, № 16. — С. 2046—2053.
9. A practical tool kit for making portable compilers / A. S. Tanenbaum [и др.] // Communications of the ACM. — 1983. — Т. 26, № 9. — С. 654—660.
10. *Takahashi, M.* Generalizations of regular sets and their application to a study of context-free languages / M. Takahashi // Information and Control. — 1975. — Т. 27, № 1. — С. 1—36.
11. Delite: A compiler architecture for performance-oriented embedded domain-specific languages / A. K. Sujeeth [и др.] // ACM Transactions on Embedded Computing Systems (TECS). — 2014. — Т. 13, 4s. — С. 134.
12. An experimental study of the influence of dynamic compiler optimizations on Scala performance / L. Stadler [и др.]. — 2013.

13. *Spinellis, D.* Notable design patterns for domain-specific languages / D. Spinellis // Journal of systems and software. — 2001. — Т. 56, № 1. — С. 91—99.

14. *Sikkel, K.* Parsing of context-free languages / K. Sikkel, A. Nijholt. — Springer, 1997. — С. 61—100.

15. Frame model of a compiler of cluster parallelism for embedded computing systems / V. Ruchkin [и др.]. — 2017.

16. Bringing low-level languages to the JVM: efficient execution of LLVM IR on Truffle / M. Rigger [и др.]. — 2016.

17. *Rigger, M.* Sulong-execution of llvm-based languages on the jvm / M. Rigger, M. Grimmer, H. Mössenböck // ICOOOLPS'16. — 2016.

18. A Case for Context-Free Grammar / C. Price [и др.] // Journal of Computer Science and Software Engineering. — 2018. — Т. 9, № 4.

19. *Pool, T.* Trufflereloader: a low-overhead language-neutral reloader / T. Pool, A. R. Gregersen, V. Vojdani. — 2016.

20. *Paulson, L.* A semantics-directed compiler generator / L. Paulson. — 1982.

21. binpac: A yacc for writing application protocol parsers / R. Pang [и др.]. — 2006.

22. *Niephaus, F.* GraalSqueak: A Fast Smalltalk Bytecode Interpreter Written in an AST Interpreter Framework / F. Niephaus, T. Felgentreff, R. Hirschfeld. — 2018.

23. *Moore, R. C.* Removing left recursion from context-free grammars / R. C. Moore. — 2000.

24. *Mernik, M.* When and how to develop domain-specific languages / M. Mernik, J. Heering, A. M. Sloane // ACM computing surveys (CSUR). — 2005. — Т. 37, № 4. — С. 316—344.

25. An evaluation of vectorizing compilers / S. Maleki [и др.]. — 2011.

26. *Lovato, M. E.* Parser visualizations for developing grammars with yacc / M. E. Lovato, M. F. Kleyn. — 1995.

27. *Louden, K. C.* Compiler construction / K. C. Louden // Cengage Learning. — 1997.

28. Lex & yacc / J. R. Levine [и др.]. — 1992.

29. *Leijen, D.* Domain specific embedded compilers / D. Leijen, E. Meijer. — 1999.

30. *Lee, L.* Learning of context-free languages: A survey of the literature / L. Lee. — 1996.

31. *Knuth, D. E.* Semantics of context-free languages / D. E. Knuth // Mathematical systems theory. — 1968. — Т. 2, № 2. — С. 127—145.

32. *Kitaura, K.* Random testing of compilers' performance based on mixed static and dynamic code comparison / K. Kitaura, N. Ishiura. — 2018.

33. *Jørring, U.* Compilers and staging transformations / U. Jørring, W. L. Scherlis. — 1986.

34. *Jones, S. L. P.* YACC in SASL—an exercise in functional programming / S. L. P. Jones // Software: Practice and Experience. — 1985. — Т. 15, № 8. — С. 807—820.

35. Yacc: Yet another compiler-compiler / S. C. Johnson [и др.]. — 1975.

36. Context-free grammar induction using genetic programming / F. Javed [и др.]. — 2004.

37. Librando: transparent code randomization for just-in-time compilers / A. Homescu [и др.]. — 2013.

38. *Hearnden, D.* Anti-Yacc: MOF-to-text / D. Hearnden, K. Raymond, J. Steel. — 2002.

39. *Grosch, J.* A tool box for compiler construction / J. Grosch, H. Emmelmann. — 1990. — С. 106—116.

40. TruffleC: dynamic execution of C on a Java virtual machine / M. Grimmer [и др.]. — 2014.

41. *Griffiths, T. V.* On the relative efficiencies of context-free grammar / T. V. Griffiths, S. R. Petrick // Communications of the ACM. — 1965. — Т. 8, № 5. — С. 289—300.

42. *Glanville, R. S.* A new method for compiler code generation / R. S. Glanville, S. L. Graham. — 1978.

43. *Gaikwad, S.* Performance analysis for languages hosted on the truffle framework / S. Gaikwad, A. Nisbet, M. Luján. — 2018.

44. *Ferro, M. V.* Efficient incremental parsing for context-free languages / M. V. Ferro, B. A. Dion. — 1994.

45. *Ekman, T.* The jastadd extensible java compiler / T. Ekman, G. Hedin // ACM Sigplan Notices. — 2007. — Т. 42, № 10. — С. 1—18.

46. *Earley, J.* An efficient context-free parsing algorithm / J. Earley // Communications of the ACM. — 1983. — Т. 26, № 1. — С. 57—61.

47. *Earley, J.* An efficient context-free parsing algorithm / J. Earley // Communications of the ACM. — 1970. — Т. 13, № 2. — С. 94—102.

48. *Duboscq, G.* Speculation without regret: reducing deoptimization meta-data in the Graal compiler / G. Duboscq, T. Würthinger, H. Mössenböck. — 2014.

49. An intermediate representation for speculative optimizations in a dynamic compiler / G. Duboscq [и др.]. — 2013.

50. *Deursen, A. V.* Little languages: Little maintenance? / A. V. Deursen, P. Klint // Journal of Software Maintenance: Research and Practice. — 1998. — Т. 10, № 2. — С. 75—92.

51. *Cooper, K. D.* Adaptive optimizing compilers for the 21st century / K. D. Cooper, D. Subramanian, L. Torczon // The Journal of Supercomputing. — 2002. — Т. 23, № 1. — С. 7—22.

52. A generative programming approach to developing DSL compilers / C. Consel [и др.]. — 2005.

53. A DSL compiler for accelerating image processing pipelines on FPGAs / N. Chugh [и др.]. — 2016.

54. Testing and Verification of Compilers (Dagstuhl Seminar 17502) / J. Chen [и др.]. — 2018.

55. *Brantner, M.* Modern stored procedures using GraalVM: invited talk / M. Brantner. — 2017.

56. *Bhamidipaty, A.* Very fast YACC-compatible parsers (for very little effort) / A. Bhamidipaty, T. A. Proebsting // Software: Practice and Experience. — 1998. — Т. 28, № 2. — С. 181—190.

57. How should compilers explain problems to developers? / T. Barik [и др.]. — 2018.

58. *Aho, A. V.* Compilers: principles, techniques and tools (for Anna University), 2/e / A. V. Aho. — Pearson Education India, 2003.

59. *Aho, A. V.* Compilers: principles, techniques and tools (for Anna University), 2/e / A. V. Aho. — 2003.

60. *Aho, A. V.* A minimum distance error-correcting parser for context-free languages / A. V. Aho, T. G. Peterson // SIAM Journal on Computing. — 1972. — Т. 1, № 4. — С. 305—312.

61. *Aho, A. V.* Translations on a context free grammar / A. V. Aho, J. D. Ullman // Information and Control. — 1971. — Т. 19, № 5. — С. 439—475.

62. OCEANS: Optimizing compilers for embedded applications / B. Aarts [и др.]. — 1997.