Тестирование эффективности LR(1)-разбора

Владимир Лидовский

Статья посвящена анализу актуальности LR(1)-сканеров для современного компьютерного оборудования. Она описывает результаты, полученные специально разработанной программой, которая генерирует LR(1)-, LALR(1)-таблицы и соответствующие числовые величины для заданных формальными грамматиками языков программирования. Делается вывод о полезности опциональной поддержки LR(1)-разбора программой GNU bison.

This article analyzes the actuality of LR(1)-parsers for the modern computer hardware. It describes the results produced by specially designed computer program which generates LR(1), LALR(1) tables, and numeric values corresponding to the given formal grammars of the programming languages. It concludes that as option LR(1) parsing will be useful as expansion to GNU bison program.

Как и LALR-разбор, LR-разбор имеет линейную временную сложность. LR-разбор имеет несколько теоретических преимуществ. Использование LR(1)-разбора вместо LALR(1)-разбора может устранить некоторые конфликты типа свёртка-свёртка, которые, как отмечается в [8], “обозначают серьёзную проблему” при разработке компилятора: LR(1)-разбор успешно справляется с ситуациями, порождающими “таинственные конфликты свёртка-свёртка” [1] при LALR(1)-разборе. Кроме того, использование LR(1)-разбора позволяет наиболее точно специфицировать ошибки при компиляции. При LALR(1)-разборе диагностика ошибок менее точна, так как ошибочные ситуации могут объединяться.

После опубликования Д. Кнутом работы [2], определяющей LR(k)-алгоритм, прошло более 10 лет прежде чем его упрощённый вариант, LALR(1), чрезвычайно удачно реализованный С. Джонсоном в программе Як (yacc), стал вытеснять прежде преобладавшие методы разбора сверху-вниз. Необходимость упрощения вытекала из аппаратных ограничений вычислительных машин того времени. В фундаментальном труде [7] отмечается, что “построение полной системы LR(1)-множеств пунктов требует слишком много памяти и времени, чтобы использоваться на практике”.

Компьютер с несколькими гигабайтами оперативной памяти уже стал фактическим стандартном и можно попытаться усомниться в верности приведённой цитаты, сформулированной тогда, когда стандартным объёмом памяти считались 64–512 килобайт. Однако, в новом учебнике для вузов [10] по-прежнему отмечается “высокая трудоемкость построения управляющей таблицы для LR(1)-грамматик”.

Автором и М. Молокановым на кафедре “Моделирование систем и информационные технологии” Российского государственого технологического университета — «МАТИ» им. К. Э. Циолковского было проведено соответствующее исследование в течении семестра. На языке программирования Си++ со стандартной библиотекой была реализована компьютерная программа lr-lalr-test, которая по заданной грамматике строит соответствующую ей полную каноническую систему LR(1)-множеств пунктов, а также полную систему LALR(1)-множеств пунктов. Последнее построение позволяет, в частности, верифицировать результаты программой Бизон. Программа также вычисляет количества терминалов, метасимволов, правил, LR- и LALR-множеств пунктов и другие численные характеристики грамматики. Выбор языка программирования был обусловлен прежде всего необходимостью проведения больших объемов расчетов за приемлемое время и поддержкой быстрой работы с множествами. Языки Лисп и Пролог, а также Перл, Питон, Рубин и некоторые другие, имеющие необходимые средства, показали значительно меньшую скорость работы. Использовалась система программирования GNU Си++: компилятор, отладчик, профайлер и компоновщик. Алгоритм для построения канонической системы множеств пунктов взят из [7].

Для анализа использовались следующие языки в их свободно-распространяемых вариантах с открытыми текстами исходников (далее следует список языков с указанием названия, года версии исходников, разработчика, сетевого адреса, примечаний):

1. Бизон (bison), 2008, FSF (Free Software Foundation), `http://www.gnu.org/software/bison/`, GNU вариант синтаксического сканера Як;
2. Биси (bc), 1997, FSF, `http://www.gnu.org/software/bc/`, программируемый консольный калькулятор неограниченной точности;
3. Бэш (bash), 2009, FSF, `http://tiswww.case.edu/php/chet/bash/bashtop.html`, язык стандартной оболочки Linux;
4. Майэскьюэль (mysql), 2008, MySQL AB и Sun Microsystems (ныне — Oracle), `http://www.mysql.com/`, вариант Эскьюэль (SQL);
5. Обджектив Си (objective C), 2001, FSF, `http://gcc.gnu.org/`, GNU вариант языка фирмы Apple);
6. Оук (awk), 2009, FSF, `http://www.gnu.org/software/gawk/`, gawk (GNU вариант);
7. Паскаль (pascal), 2006, FSF, `http://www.gnu-pascal.de/gpc/h-index.html`, gpc (GNU вариант расширенного Паскаля);

8. Перл (perl), 2008, Larry Wall and others, `http://www.perl.org/`;
9. Питон (python), 2008, PSF (Python Software Foundation), `http://www.python.org`, исходники из расширенных НФБН, принятой в PSF, были преобразованы к обычным НФБН;
10. Постгрескьюэль (postgreSQL), 2010, Global Development Group, `http://www.postgresql.org/`, вариант Эскьюэль;
11. ПХП (php), 2010, Zend Technologies, `http://www.php.net`;
12. Рубин (ruby), 2007, Юкихиро Мацумото (Yukihiro Matsumoto), `http://www.ruby-lang.org/en/`;
13. Си (C), 2001, FSF, `http://gcc.gnu.org/`, gcc (GNU вариант Си);
14. Си++ (C++), 2001, FSF, `http://gcc.gnu.org/`, g++ (GNU вариант Си++);
15. Флекс (flex), 1990, The Regents of the University of California , `http://flex.sourceforge.net/`, GNU вариант лексического сканера Лекс (lex);
16. Хок (hoc), 1984, Керниган и Пайк, `http://litwr.atspace.com`, расширение описанного в [8] программируемого консольного калькулятора;
17. Ява (java), 2001, FSF, `http://gcc.gnu.org/`, GNU вариант языка Ява;
18. PL/pgSQL, 2010, Global Development Group, `http://www.postgresql.org/`, встроенный в Постгрескьюэль процедурный язык.

Грамматика для программы lr-lalr-test задаётся файлом в особом формате, получаемом из исходного файла в формате Як/Бизон обработкой специально разработанной программой-препроцессором. Цель препроцессора — получить “чистую” грамматику, в частности, без информации о приоритетах знаков и правил. Полученные результаты приводятся в следующей таблице. Количество LALR-состояний получается на единицу меньшим, чем у программ Як/Бизон, из-за отсутствия избыточного финального сдвига по особому конечному символу входной цепочки. Количество терминалов на два, а нетерминалов на $n + 1$, где n — это количество неявных пустых правил (правил, порождаемых семантическими действиями между символами правил грамматики), больше, чем в исходной грамматике. Добавленные терминалы — это уже упомянутый конечный символ входной цепочки и символ-ошибка, а добавленный нетерминал — это начальный символ расширенной грамматики.

Очевидно, что общее количество вариантов LR(1)-пунктов не превосходит wt, где w — это суммарное количество символов, кроме пустых, во всех правилах — *вес грамматики*, а t — количество терминалов. Следовательно, общее количество LR(1)-состояний не превосходит 2^{wt}. Из-за того, что в LALR-пунктах FIRST-часть

пунктов теряет различающее значение, максимум количества LALR-пунктов в t раз меньше и, соответственно, максимум общего количества подмножеств всех LALR-пунктов — 2^w.

Язык	**Количество**			**вес грамматики**	**Число множеств**		**время, сек**
	терминалов	**нетерминалов (неявных)**	**правил**		**LR(1)**	**LALR(1)**	
Бэш	59	38 (0)	167	680	3684	343	0.84
Бизон	56	33 (3)	105	267	239	142	0.04
Биси	49	35 (13)	107	324	1149	184	0.41
Майэскьюэль	588	837 (176)	2377	6937	4823743	4076	?
Обджектив Си	87	237 (65)	589	1814	4353	991	13.57
Оук	66	56 (5)	165	526	3217	307	4.98
Паскаль	138	294 (65)	797	2394	39355	1329	20.22
Перл	89	66 (2)	209	727	3600	418	18.37
Питон	85	174 (83)	333	876	4758	529	1.62
Постгрескьюэль	430	515 (0)	2090	6926	873573	3837	2426.76
ПХП	153	181 (68)	463	1512	10986	892	33.4
Рубин	148	170 (32)	561	1730	180954	970	306.55
Си	87	167 (40)	426	1357	2888	728	7.29
Си++	112	294 (57)	918	3030	31683	1789	109.25
Флекс	68	27 (0)	97	275	232	139	0.05
Хок	40	17 (2)	63	210	692	119	0.93
Ява	110	163 (12)	504	1721	5594	774	17.01
PL/pgSQL	99	78 (2)	179	446	684	249	0.26

Время работы по сравнению с программой Бизон увеличилось значительно — на несколько порядков — соответственно росту количества множеств-состояний. Однако, время построения LR(1)-системы вполне сопоставимо с временем компиляции файлов исходников языков программирования в исполняемые файлы. Кроме

того, возможна лучшая оптимизация кода. Как уже отмечалось, разница в количестве LALR- и LR-состояний, в общем случае, определяется экспоненциальной зависимостью, что делает LR-разбор в крайних ситуациях фундаментально неэффективным. Эта общая теоретическая неэффективность практически проявила себя только к языку Майэскьюэль, для которого даже 16 гигабайт оперативной памяти оказались недостаточными и, частично, к языкам Рубин и Постгрескьюэль, потребовавшим соответственно 1 и 4 гигабайт для вычислений без привлечения медленной виртуальной (дисковой) памяти. Поэтому можно предположить, что *использование LR-разбора, как одной из опций восходящего анализа, является вполне приемлемым для ресурсов существующих компьютеров во многих случаях.* Кроме того, необязательно использовать именно канонические таблицы. Существуют несколько алгоритмов [3–6,9], позволяющих получать только существенные для работы анализатора состояния, что позволяет значительно сократить общее количество LR-состояний и намного увеличить скорость работы программы. Хотя в этом случае *будет теряться часть информации об ошибках разбора.*

Помимо использования программы Бизон результаты верифицировались LR-сканерами Мста (Msta) и Нуасс. Для Майэскьюэль результат удалось получить только Мстой. Учитывая идентичность алгоритмов LALR- и LR-разборов (они отличаются только наполнением таблиц) можно естественно интегрировать LR-разбор в существующие программы для генерации компиляторов, права копирования на которые позволяют производить такую разработку. Можно, например, добавить как опцию LR-разбор программам Як или Бизон. В последнем случае можно использовать возможность подключения к проекту Бизон в отдельной svn-ветке.

Уже существует несколько программ для LR(1)-разбора (CSP, Dragon, Нуасс, Lisa, Yooparse, Whale, ...), но они либо вводят собственный не вполне совместимый с Бизон синтаксис для определения грамматик, либо не реализуют всех возможностей программы Бизон, что, учитывая широкую распространённость исходников и прочих материалов для Як/Бизон, не вполне удобно. В частности, при тестировании программы Yooparse были обнаружены ошибки в построенной системе множеств пунктов для языка Перл — точнее не всегда вычисляются правильно FIRST-части пунктов. Программа Нуасс показала не полную совместимость с форматом грамматики программ Як/Бизон и слишком расточительное обращение с памятью — она не смогла вычислить каноническую таблицу LR-анализа на компьютере с 8 гигабайтами памяти для языка Рубин.

Исходники программы lr-lalr-test и документация к ней доступны по адресу `http://litwr.atspace.com` — они распространяются на условиях лицензии GNU GPL (`http://www.gnu.org/copyleft/gpl.html`).

Среди побочных результатов исследования было обнаружено значительное расхождение с [11] по численным

характеристикам синтаксиса, что свидетельствует о том, что грамматики для разных версий языка, а также грамматики, написанные для генерации синтаксических анализаторов, и грамматики, теоретически определяющие язык, могут весьма значительно различаться по этим характеристикам.

Литература

1. C. Donnely, R. Stallman *Bison*: 2 April 2009, version 2.4.1 — Free Software Foundation, ISBN 1-882114-44-2 (`http://www.gnu.org`). 174 pages.
2. D. Knuth *On the translation of languages from left to right* //Information and control 8, 1965. P. 607–639.
3. D. Pager *A Practical General Method for Constructing LR(k) Parsers* //Acta Informatica 7, 1977. P. 249–268.
4. D. Pager *Eliminating Unit Productions from LR Parsers* //Acta Informatica 9, 1977. P. 31–59.
5. D. Pager *The Lane-Tracing Algorithm for Constructing LR(k) Parsers and Ways of Enhancing Its Efficiency* //Information Sciences 12, 1977. P. 19–42.
6. D. Pager *The lane tracing algorithm for constructing LR(k) parsers* //Proceedings of the fifth annual ACM symposium on Theory of computing, 1973. P. 172–181.
7. А. Ахо, Р. Сети, Д. Ульман *Компиляторы: принципы, технологии и инструменты*: Пер. с англ. М.: Издательский дом «Вильямс», 2003. 768 с.
8. Б. Керниган, Р. Пайк *UNIX — универсальная среда программирования* Пер. с англ. М.: Финансы и статистика, 1992. 304 с.
9. В. Н. Макаров *Практичный метод оптимизации LR(1)-анализаторов* //Программирование. 1988. №3. С. 38–48.
10. А. Ю. Молчанов *Системное программное обеспечение* СПб.: Питер, 2010. 400 с.
11. С. Свердлов *Арифметика синтаксиса* //PC Week. 1998. №42–43 (166–167). С. 84–87.

Для подготовки материалов использовалась система Plain TEX

Опубликована в виде доклада в материалах (том 22) VII международной научно-практической конференции НАУЧНЫЙ ПРОГРЕСС НА ГРАНИЦЕ ТЫСЯЧЕЛЕТИЙ, проходившей в Праге в 2011 г., а также в статье «Анализ LR-разбора для 18 языков программирования» в журнале “Информационные технологии” №12, 2011.