В 1965 году Кнут [1] ввел синтаксический анализ LR (A-), метод синтаксического анализа снизу вверх

может использоваться для распознавания самого большого класса детерминированных контекстно-свободных языков. (The

«L» означает сканирование входа слева направо, «R» - построение самого правого

вывод в обратном порядке, и к для числа входных символов предвкушения, которые

используется при принятии решений о синтаксическом анализе.) С годами этот метод синтаксического анализа привлек

большое внимание, потому что в дополнение к его способности распознавать большой класс языков

Полученные парсеры предлагают следующие преимущества:

• они могут быть построены автоматически из определения грамматики без контекста:

• они экономят время, так как могут принимать или отклонять входные данные в одном слева направо

сканирование его без резервной копии;

• они могут обнаружить ошибку как можно раньше.

Не зависящая от контекста грамматика называется LR (A-), если анализатор LR (A-) может быть успешно создан.

от него. Язык называется LR (A-), если он может быть определен грамматикой LR (A).

В их канонической форме. Парсеры LR (A-) (когда k> 0) обычно требуют слишком много места для

иметь практическое применение. (Связь между произвольной контекстно-свободной грамматикой и

Размер его канонического LR (A-) парсера никогда точно не демонстрировался, но для типичного

грамматика языка программирования, когда k = синтаксический анализатор обычно содержит несколько сотен

состояния: когда k = 1. парсеры с несколькими тысячами состояний являются общими.). ^ s результат, два

Вариант ^ парсеров LR (A-), который был изобретен DeRemer, приобрел положительную

года. Они известны как Look.Ahead LR (A-) (L .- \ LR (A-)), представленный в 1969 году и описанный

в [3] и простой LR (A-) (SLR (A-)), введенный в 197] и описанный в [-4]. Эти

варианты LR (A-

) парсеры относительно компактны, потому что их основной автомат

является машиной LR (0), независимо от значения A-. Набор языков, который является SLR (A)

надлежащее подмножество набора языков LALR (A-), которое, в свою очередь, является надлежащим подмножеством

набор языков LR (A). Однако на практике грамматики LALR (A) используются потому, что они

достаточно мощный, чтобы вместить большинство конструкций языка программирования.

Поддерживая минимальное количество состояний, варианты SLR (A-) и LALR (A-)

помочь уменьшить требования к пространству парсера LR (A-), сохраняя преимущество в скорости

из последних.

Символы контекстно-свободной грамматики делятся на два класса: терминалы (входные

символы) и нетерминалы (символы фразы). LR (A-) синтаксический анализатор (или вариант) для contextfree

грамматика является детерминированным автоматом, который может быть представлен двумя

матрицы: ДЕЙСТВИЕ, которое представляет отображение функции действия разбора и GOTO,

который представляет отображение функции goto. Эти матрицы будут упоминаться

в общем случае как таблицы разбора.

Функция действия разбора принимает в качестве аргументов состояние и строку из k терминалов.

(называется строкой lookahead) и выдает одно из четырех значений: shift, lower /. Ошибка или принять.

Функция goto является переходной функцией автомата. Принимает в качестве аргументов

состояние и грамматический символ (терминальный или нетерминальный) и создает либо новое состояние

что парсер должен войти или ошибка. Таким образом, строки таблиц разбора индексируются

устаревшим автоматом, каждый столбец ACTION индексируется строкой терминалов

длины k и каждый столбец GOTO индексируется отдельным символом грамматики. каждый

запись в таблице синтаксического анализа является либо полезной записью, которая представляет допустимое движение, которое нужно предпринять

автоматом (для соответствующей пары индексов) или ошибочной записью.

1.2 Проблемы

Как можно видеть из определения таблиц синтаксического анализа, для данной грамматики

количество состояний (строк) в его автомате SLR (A-) или LALR (A) и количество столбцов

в его матрице GOTO остаются фиксированными для любого значения k: но количество столбцов в его

Матрица ACTIO.N экспоненциальна относительно k. Однако, если записи об ошибках сохраняются

как пустые слоты эти матрицы очень редки. Как правило, менее 2'7 (.

Таблицы синтаксического анализа парсера L. \ LR (1) полезны.

Одним из наиболее важных вопросов при разборе LR является поиск подходящих структур данных для

эти таблицы разбора, чье пространство требуется. в худшем случае, пропорционально количеству

полезные записи в таблицах, но эффективность которых сравнима с временной эффективностью матрицы

представление. .Другим важным вопросом является обеспечение эффективного восстановления после ошибок.

Система для этого синтаксического анализа. В частности, варианты LR (A-) теряют присущие

способность их канонического аналога обнаружить ошибку в самой ранней возможной точке.

Попытка сделать LR (A-) парсеры более полезными, то есть быстрее, меньше и больше

автоматизация поднимает следующие важные вопросы:

• Как можно эффективно создать такой парсер?

• Как можно исправить ошибки в этих рамках \*? Точнее, может автомат

или полуавтоматическая система восстановления после ошибок, которая будет работать со всеми

LR (A) парсеры?

• Какова связь между конкретным типом анализа и скоростью и размером

результирующего автомата? (Например, использование дополнительных прогнозных символов может

Потенциально влияет на размер и скорость).

• Как таблицы разбора могут быть представлены компактно, не жертвуя скоростью?

Было опубликовано несколько работ, посвященных этим проблемам. В следующих

В подразделах кратко описаны результаты некоторых плодотворных работ в этих областях.

с последующим описанием основных нововведений этого тезиса.

1.2.1 Конструкция парсера LALR (A)

LALR (A-) парсеры почти всегда используются, потому что они более компактны, чем оба

парсеры LR (A :) и SLR (A-) и, кроме того, они более мощные, чем SLR (A-)

парсеры. Большинство коммерчески доступных генераторов синтаксических анализаторов имеют дело только со случаем k = 1.

Парсер LALR (A-) можно построить, сначала собрав парсер LR (/ c), а затем

объединение некоторых штатов Однако такой подход нецелесообразен, так как он обычно труден

создать парсер LR (/ r) из-за его большого пространства. Вместо этого два шага

подход обычно принимается. На первом этапе строится автомат LR (0) (все

LALR (A-) парсеры основаны на этом автомате); и на втором этапе результирующие таблицы

дополняются необходимыми заблаговременными действиями.

Неформально, элемент LR (0) - это контекстно-грамматическое правило с маркером, который разделяет

его правая часть в префикс, который был обработан и суффикс, который еще не был

обработанный. Элемент называется финальным, когда маркер показывает, что его префикс

вся правая часть (и ее суффикс пуст). Каждое состояние LR (0) автомата

соответствует набору предметов.

Набор LALR (A-) для конечного элемента в состоянии автомата LR (0) является набором

терминальных строк длины k \ Во время синтаксического анализа, если следующие k символов на входе совпадают

одна из строк предпросмотра, синтаксический анализатор должен выполнить сокращение по определенному правилу

из которого происходит соответствующий элемент. Следовательно, на втором этапе строительства

синтаксический анализатор LALR (A-) для каждого сланца и каждого элемента строки предпросмотра из набора предисловия

вычисляется в этом состоянии,

из этих алгоритмов, однако, менее эффективны, чем подход DeRemer и Penello.

Только Кристенсен и Мадсен обобщили свой алгоритм для вычисления множеств

для k> 1. Однако их обобщенный алгоритм представляет только теоретический интерес, поскольку

он вычисляет полные наборы предпросмотра, необходимые для каждого неоднозначного состояния. Как можно

наблюдается из определения ACTIO.X matri.x, проблема вычисления полной

опережающие наборы для синтаксического анализатора LALR (Ar) неразрешимы в том, что размер

само решение может быть экспоненциальным по отношению к k.

1.2.2 Восстановление после ошибок

Восстановление после ошибок традиционно делится на простое восстановление [21] [37]. уровень фразы (или вторичный)

восстановление [9] [21] [27] [37]. и восстановление объема [37].

В простом восстановлении делается попытка исправить ошибочный ввод с помощью примитива

операции редактирования над символом ошибки. То есть. перед ним может быть вставлен символ. Это

может быть заменен другим символом или может быть удален.

При восстановлении на уровне плиазы последовательность нулей или более токенов в окрестности ошибки

Символ сбрасывается со входа или заменяется нетерминалом. Erroi Prodvctions

подход '^' acc является формой вторичного восстановления, когда нетерминальный кандидат> должен быть

Выбранные для этого вида ремонта определяются производством, правые стороны которого включают

специальный символ терминала, называемый символом ошибки. Sippu и Soisalon-Soininen [27] представлены

более сложный метод вторичного восстановления, который не требует использования

ошибки производства, но это несколько дорого, потому что это требует, чтобы некоторая информация была

вычисляется во время выполнения.

Восстановление области было введено Берком и Фишером. Идея состоит в том, чтобы вставить последовательность

закрытие синтаксических фрагментов в тексте, где это уместно, для завершения спецификации

определенных блоков или блочных структур. Этот подход восстановления очень эффективен, когда

используется в сочетании с первичным и вторичным восстановлением, как предложено в [37]. Тем не мение,

каждый соответствующий закрывающий фрагмент должен был быть указан явно как последовательность терминала

символы. Поэтому их метод требовал, чтобы пользователь был знаком с языком в

вопрос.

Метод устранения ошибок Бёрка и Фишера является наиболее практичным и эффективным

metiiod на сегодняшний день. Тем не менее, он основан на методе отложенного анализа, который требует

двойной анализ ввода даже для правильных программ. В дополнение к введению

Берк и Фишер также внесли некоторые улучшения в первичное восстановление

Рассматривая mtrging из двух смежных токенов и неверный набор ключевых слов. Другая ошибка

методы восстановления (например, [17]) были опубликованы, но они в основном представляют теоретический интерес

и не используются на практике.

1.2.3 Синтаксический анализ таблиц

Проблема сжатия таблицы разбора LR широко изучалась, но до сих пор. нет

Общий метод был разработан, который хорошо подходит для различных приложений.

Сжатие таблиц до сих пор рассматривается в литературе как проблема времени и пространства.

В зависимости от приложения, методы из разреженного матричного представления с последовательным

поиск, хеширование и другие, более экономичные по времени, но занимающие много места, прямой доступ

методы были предложены.

Техника сжатия таблиц, используемая в Yacc [9], состоит из комбинации прямого

методы доступа для переходов и методы последовательного поиска для сокращения действий. Для

Грамматика SMAU, это приемлемый подход. Однако такой подход не всегда

хорошо выполнять на больших грамматиках. Тарьян и Яо [23] опубликовали анализ прямого

метод доступа Циглера, и сформулировал точные условия, при которых это сжатие

Техника выполнения WeU. В 1984 году Денкер, Дурре и Хейфт представили

метод доступа, основанный на раскраске графа, который очень хорош в минимизации пространства. К несчастью,

их подход требует ссылки на упакованную булеву матрицу для проверки действительности

каждого действия. На практике этот тест делает их метод медленнее, чем метод Yacc.

1.3 Вклад диссертации

В этом тезисе делается несколько вкладов в каждой из областей, упомянутых выше. Эти

результаты были интегрированы в систему генератора синтаксических анализаторов, которая автоматически производит

эффективные парсеры L. '\ LR (A-) с восстановлением ошибок из контекстно-свободного определения грамматики.

Эти нововведения обобщены как следующие:

• .A. новый фреймворк для парсеров L.A.LR (A-). Как указывалось ранее, строительство

традиционный синтаксический анализатор LALR (A :) нецелесообразен, так как требуется требуемый размер

для такого парсера может быть экспоненциальный. Подход, принятый в этом методе, может быть лучшим

описывается как генерация синтаксического анализатора L. \ LR (A-) со строками прогнозируемой длины voriabk-lfngth.

набор для конечного элемента в заданном состоянии синтаксического анализатора LR (A-) состоит из набора

из строк длины k, которые могут появиться на входе, когда парсер входит в это состояние.

В синтаксическом анализаторе L. \ LR (A) со строками предварительного просмотра переменной длины каждый набор предварительного просмотра

заменяется минимальным подмножеством префиксов его строковых элементов, достаточным для

сделать синтаксический анализатор детерминированным. Эта новая структура обсуждается в главе 3.

Практический алгоритм построения переменных парсеров LALR (A-). Этот метод не

вычисляет только минимальное количество необходимой информации, но

это постепенно. Таким образом, пространство, необходимое для построения этих множеств, сохраняется

до минимума. Этот алгоритм представлен в разделе 3.3.

• Полностью автоматический метод устранения ошибок, который является более практичным и эффективным, чем

другие известные методы. Этот независимый от языка и машины метод применим

ко всем формам синтаксического анализа LR (A-), но это особенно эффективно в контексте парсера

генерируется вышеуказанным методом. Восстановление после ошибок является предметом главы 4.

• Практичный и эффективный метод сжатия таблиц синтаксического анализа LR (A-). Это сжатие

Метод также применим ко всем формам синтаксических анализаторов LR (A-), но особенно

эффективно в этих рамках. Таблица сжатия описана в главе 5.

Глава 2

Генератор парсера

2.1 Основные понятия и терминология

Контекстно-свободная грамматика (CFG) - это четверка (. \, T, P, S), где N - конечное множество нетерминалов

символы. T является конечным набором терминальных символов, отличных от A ". 5 является выделенным

символом N называется начальный символ, а P - конечный набор произведений, каждое из которых имеет вид

A - lj, где A € A 'и ^ € V ". Учитывая грамматику 6', V (словарь) расшифровывается как

ОРЕХ.

Строчные греческие буквы, такие как q. J и) используются для обозначения строк в 1 ". Строчные

Римские буквы в начале алфавита {а. б. в) и / используются для обозначения

символы в T, в то время как те в конце алфавита {i.y.z] обозначают строки в T '. Верхний регистр

буквы в начале алфавита (A.B.C) обозначают нетерминалы в .V, а

те, что в конце (A'.V.Z) обозначают символы в V. Пустой символ обозначается <и

пустая строка обозначается как r. Маркер конца файла обозначается как ±. Длина строки -

обозначается | -) |.

Следующие обозначения SETL2 [42] также будут использоваться. Символ fi обозначает особый

константа "неопределенное значение". Конечная упорядоченная последовательность произвольных элементов, называемая tvj) lc.

будет обозначено перечислением элементов в правильном порядке, в скобках "['и

] ". Если 7 - кортеж. T (i) - элемент uh элемента T, а 7" (T7} .. n) - кортеж, состоящий из

элементы Т (м). T {w + \) r (n), если n)> = n и пустой кортеж. []. иначе. Если

T] и T2 - это кортежи, тогда 7 "] + T2 - это кортежи, полученные путем добавления последовательности

элементы в T2 в конце последовательности элементов T- \. Однозначная карта из

конечное множество A (область) к конечному множеству B (диапазон) будет представлено как множество упорядоченных

пары [год]. где i £ A. y e B и каждый элемент .4 отображается не более чем на один элемент

B. Имеется отображение j \ / и элемент x в его области. M {i) представляет собой клемму y в

диапазон М, который связан с х (у, называется изображением х). Если A 'является кортежем, установите или

карта. его длина или кардинальность обозначается #A '.

Отныне предполагается, что данная грамматика G была дополнена новым

Стартовое правило 5 '- S ± \* и G не содержит всех окончаний vse.ksf'. Нетерминал А сказано

быть бесполезным, если он не генерирует какую-либо строку терминалов: т.е. A - / - "^ ic для любого u- T '.

Для заданной контекстно-свободной грамматики.

FIRST;t(Q) = {x \ (o x>[„ x3 and |j| = k) or (o =>' x and |t| < k)].

Таким образом, FIRST ^ fo) состоит из терминальных префиксов U длины k (или меньше, если o выводит

строка termJncd длиной меньше k) строк терминала, чем может быть получена из

о. С функцией FIRSTjt тесно связана первая функция без s, EFFt (o), которая

определяется как все элементы FIRSTfc (Q), вывод которых не включает замену

ведущий нетерминал на £. Более формально,

EFTk(a) = {w\q =>;„ =>;„ wx, ^ Awx VA € A" and {w] = FIRSTjt(u;x)}

Если X и y обозначают произвольные строки, то x.y - это строка, полученная путем объединения

Строка, обозначенная у, в строку, обозначенную х. Пусть A / и N два набора строк,

Операция конкатенации распространяется на наборы строк следующим образом:

M.N = {x.y I x€ M, y £ N)

If M. .\ C T- then

M et N = U{FIRSTt(u-) I w € M.N])

2.1.1 LR (A) парсеры

Элемент LR (A-) представляет собой четверку {A.Q, p.u), написанную [A - o 3.n], где A - qJ € P

и ты? Это предвкушение. А называется стороной Ифи, а называется префиксом. ; 3 вызывается

суффикс Первый символ в 3. сразу после точки. называется символом точки.

Когда 3 = с. элемент называется финальным илемом, а символ точки считается e.

Пусть K - множество LR (A-) предметов. CLOUSE обозначаемое CLOSV RE {K), определено

как наименьшее множество, удовлетворяющее уравнению:

CLOSURE(A-) = A' U {[5 -•-;.. r] | r € FIRSTA-liii),

[A -a •.B^,u]e CLOSrRE(A'), B - - £ P)

Пусть p будет множеством замыканий. Ядро sei из p, обозначаемое KERN'EL (p), является наименьшим подмножеством

элементов LR (A) в p, таких что:

p= CLOSl"RE(KERNEL(p))

Дан набор предметов, с. для каждого символа точки .V, который появляется в элементе р. Goto

Функция: GOTO ^ c- определяется для пары (p, A ') следующим образом:

GOTOk(p.X)= CLOSrRE({[.4-QA'-i3,u] 1 [.4- o-X3.v]e p))

Для данной грамматики G = (. \, T, P, S) канонический набор элементов LR (A ') для G \ обозначается

/ Jp. может быть построен с помощью следующей процедуры с заданной функцией замыкания (для вычисления

CLOSrRE (A ') для некоторого набора предметов A') и GOTO ^.

1. Инициализировать / f =

2. Начните с набора ядра, состоящего исключительно из начального элемента: [S '- -5]: вычислите его

замыкание установить и добавить это замыкание установить в I ^.

3. Выберите замыкающее множество p из I ^. Вычислить его набор точечных символов и применить GOTOt

функция на р и каждый из его точечных символов. Если какие-либо новые замыкающие множества еще не в I ^

полученные таким образом они добавляются в / Jp.

4. Повторяйте предыдущий шаг до тех пор, пока новые наборы замыканий не будут добавлены к I ^

,

Этот алгоритм должен четко завершаться, поскольку набор элементов и набор символов

конечно.

Определение 2.1.1. Пусть G - контекстно-свободная грамматика. Машина LR (k) для G является тройкой:

LRM ^ = (Mf, IS ^, GOTO ^), где jl / f - набор состояний LR (k), по одному для каждого набора

предметы в I ^. IS ^ - начальное состояние, соответствующее набору замыканий исходного элемента.

GOTO ^ - функция GOTO, определенная для A / f x V - A / f

,

Заметим, что состояние p в A / \* характеризуется своим набором ядра, так как полный набор

элементов, составляющих это состояние, можно воспроизвести, учитывая набор ядра и закрытие

функция. Для удобства отныне не будет проводиться никакого различия между государством и

соответствующий ему набор предметов. Также для данной грамматики G. верхний индекс G будет

опущено всякий раз, когда это упущение не вызывает путаницы. Элемент в p, который находится в KER. \ EL (7j)

называется элементом ядра р. Элемент в p, которого нет в KERXEL (p), называется замыканием

вещь.

Также удобно обобщить функцию GOTO ^ t для произвольных строк следующим образом:

GOTO;, (p. £) = p

GOTO; t (p.-Vo) = GOTOi. {GOTOk {p.X) .Q)

Пусть PRED будет инверсией GOTO /; функция. Он определяется на произвольных строках как

следующим образом:

PRED (p.Q) = {q I GOTOt ((y.o) = p]

2.1.2 LALR (il) парсеры  
Понятие парсера L.A.LR (A-) охватывается следующими определениями и теоремами  
представлен в [2-4]. В каждом пусть G CFG с LR (A-) состояниями A / ^. к> 0.  
Определение 2.1.2. Пусть p £ A / \*, thtn  
LRi (p. [. 4-o-; 3]) = {u \ [A-a-3.u] ep]  
Определение 2.1.3. Пусть [A - a S.u] bt LR (k) litm и освещено p £ M ^. thtn  
C0RE ([/ 1 -o-P], xi) = [A-Q-3]  
и  
CORE (p) = {CORE (/) :; € p}  
.Не проводится различие между пунктами [A - o d.s] и [.4 - o 3].  
  
Определение 2.1.4. Пусть p £ Mq, тогда  
URCOREa. (P) = {q € Mk \ C0RE (9) = p]  
URCORE связывает состояние LR (0) p с набором состояний LR (A-) с тем же ядром. Заметка  
что поскольку CORE (ISo) = CORE (IS; t) и что GOTO ^ ip, A '), для alJ A-> 0. зависит только  
в ядре p каждое состояние LR (A-) соответствует состоянию LR (0) с тем же ядром.  
Другими словами. URCORE /; (p) 5 ^ 0, для всех A-> и p e Mq.  
Определение 2.1.5. Пусть p e Mo. Тогда  
LALRk (p, [A - Q / 3]) = | J {LR, (9, [A ^ a 0]) \ g e URCORE, (p)}  
Определение 2.1.6. Грамматика G называется LALR (k), k>, если для всех p £ Mo и для  
все отдельные пункты [A - ^ a 0] и [B - 7] в с.  
EFFt (.; 3) Q) k LALRtlp, [A-Q-d]) r \ LALR ^. (Стр. [B - 7-]) =  
Машина LR (0), построенная для грамматики G, на самом деле является правильным парсером для 6 ': то есть  
Язык, распознаваемый LRMq, точно такой же, как язык, описанный G. Однако он  
может быть недетерминированным из-за наличия одного или нескольких несовместимых состояний. В общем.  
состояние называется несовместимым, если оно предусматривает два разных хода для данной строки предпросмотра.  
В частности, состояние в Мо несовместимо, если оно содержит два или более элементов и один из них  
Предметы - это последний пункт.  
Когда состояние p € Mo не удовлетворяет условию определения 2.1.6. также сказано  
быть непоследовательным (в LALR (A-) смысле). Строки, которые находятся на пересечении  
говорят, что два набора находятся в конфликте, и они называются конфликтными строками. Если ^ 3 ^ 5, то  
результирующие конфликты называются конфликтами с уменьшением сдвига, в противном случае они называются rtdvct-ridvcf  
конфликты.  
Теорема 2.1.1. Пусть p ^ Mk. тогда  
LR;. (P. [. 4 -o-3]) = {u '  
я  
u- € FIRSTA. (y). 5 '^; „' lAy => -jaJy. G0T0a (IS, .7o) = p}  
Теорема 2.1.2. Пусть p e Mq. тогда  
LALRap, [> l-o- / 3]) = {u- | ii € FIRST \* (i /). 5 '=>; ^ -, Ay => -, a3y.  
GOTOo (ISo.7a) = p}  
Теорема 2.1.3. Пусть p £ Mk  
\ / g £ PRED (p.Q): LR / t (p, [/ l-o • /?]) = LI {k {q. [A- -oS])