А. Ахо

Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ. Том 2

Москва
«Книга по Требованию»

УДК 53
ББК 22.3
А11

А. Ахо
А11 Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ. Том 2 / А. Ахо – М.: Книга по Требованию, 2021. – 487 с.

ISBN 978-5-458-27407-4

Второй тип фундаментальной монографии известных американских ученых посвящен методам оптимизации синтаксических анализаторов, теории синтаксически управляемого перевода, а также способам организации памяти при переводе. Большое внимание уделяется методам оптимизации объектной программы. Авторы проделали значительную работу по отбору и систематизации многочисленных результатов, полученных в последние годы; они строят изложение на едином подходе к задачам перевода и задачам оптимизации программы.Книга предназначена тем, кто работает в области системного и теоретического программирования, преподает или изучает эти дисциплиы, а также разработчикам математического обеспечения ЭВМ.

ISBN 978-5-458-27407-4

ПРЕДИСЛОВИЕ

Конструирование компиляторов — один из первых больших разделов системного программирования, которые получают сейчас строгое теоретическое обоснование. В т. 1 этой книги изложены необходимые для такого обоснования сведения из математики и теории языков и основные методы проведения быстрого синтаксического анализа. Том 2 служит продолжением т. 1, но, за исключением гл. 7 и 8, он ориентирован на несинтаксические аспекты в конструировании компиляторов.

Материал в т. 2 излагается в основном так же, как и в т. 1, но доказательства здесь несколько более схематичны. Мы попытались по возможности облегчить чтение, включив в книгу много примеров, каждый из которых иллюстрирует одно или два понятия.

Так как акцент делается на идеи, а не на языковые или машинные подробности, то основанный на этой книге курс должен сопровождаться лабораторными занятиями по программированию, чтобы студент мог получить некоторые навыки в применении обсуждаемых идей к практическим задачам. Для таких занятий можно порекомендовать упражнения на программирование, приведенные в конце разделов. Часть лабораторного курса должна быть посвящена вопросу генерации кода для таких конструкций языка программирования, как рекурсия, пересылка параметров, взаимодействие подпрограмм, адресация массивов, циклы и т. д.

О пользовании книгой

Книга возникла на основе записей лекций, прочитанных на старших курсах Принстонского университета и Стивенсовского технологического института. Материал т. 2 использовался в Стивенсовском институте как семестровый курс конструирования компиляторов, следующий за семестровым курсом, основанным на т. 1.

С точки зрения изучения конструирования компиляторов одни разделы книги, как мы полагаем, важнее других. При первом

чтении можно пропустить все доказательства, а также гл. 8 и разд. 7.4.3, 7.5.3, 9.3.3, 10.2.3 и 10.2.4.

Как и в т. 1, в конце каждого раздела помещены задачи и замечания по литературе. Задачи, не являющиеся ни проблемами для исследования, ни открытыми проблемами, мы приблизительно упорядочили в соответствии с уровнем их сложности. Для этого использовали звездочки: упражнения, не помеченные звездочкой, служат для проверки понимания основных определений; для решения упражнений с одной звездочкой требуется одна существенная догадка, а упражнения с двумя звездочками значительно сложнее.

Благодарности

В дополнение к благодарностям, выраженным в предисловии к т. 1, мы хотели бы поблагодарить Карела Чулика, Амелию Фонг, Майка Хэммера и Стива Джонсона за полезные замечания.

Альфред В. Ахо
Джефри Д. Ульман

7 МЕТОДЫ ОПТИМИЗАЦИИ СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ

В этой главе мы рассмотрим различные методы, с помощью которых можно уменьшать размер и/или повышать скорость работы синтаксических анализаторов.

Сначала исследуем вопрос об уменьшении объема памяти, требующейся для матриц предшествования. Мы увидим, что в определенных случаях, среди которых многие представляют практический интерес, матрицу предшествования размера $m \times n$ удается заменить двумя векторами длины m и n соответственно. Кроме того, покажем, как видоизменять матрицу предшествования, не затрагивая построенный по ней алгоритм разбора типа „перенос—свертка".

Затем мы опишем, как по грамматике слабого предшествования можно механически получить синтаксический анализатор на языке Флойда—Эванса, а потом рассмотрим различные приемы, применяемые для уменьшения размера полученного анализатора.

Наконец, подробно изучим различные преобразования, с помощью которых можно уменьшить размер LR-анализатора без ослабления его способности обнаруживать ошибки. Детально исследуются метод „простых LR" Де Ремера и метод расщепления грамматики Кореньяка.

Методы, описанные в этой главе, показывают, какие типы оптимизации возможны для анализаторов, построенных с помощью приемов гл. 5 тома 1. Возможны и многие другие типы оптимизации, но законченного „каталога" их не существует. Читателям, желающим получить только общее представление о методах оптимизации синтаксических анализаторов, рекомендуем помещенный в конце главы обзор ее содержания.

7.1. ФУНКЦИИ ПРЕДШЕСТВОВАНИЯ

Матрицу с элементами —1, 0, +1 или „пусто" будем называть *матрицей предшествования*. Матрицы предшествования очевидным образом применяются при построении алгоритмов

разбора, использующих технику предшествования. Например, можно применить матрицу предшествования для представления отношений предшествования Вирта—Вебера для грамматики предшествования, сопоставив

-1	с	$\lessdot$
0	с	$\doteq$
$+1$	с	$\gtrdot$
пусто	с	**ошибкой**

Матрицей предшествования можно также воспользоваться для того, чтобы представить шаги алгоритма разбора типа „перенос—свертка". Одно из таких представлений можно получить, сопоставив

-1	с	**переносом**
0	с	**ошибкой**
$+1$	с	**сверткой**

В этом разделе мы покажем, как матрицу предшествования часто удается представить в сжатой форме парой векторов, называемых функциями предшествования.

7.1.1. Теорема о представлении матрицы

Пусть M — матрица предшествования размера $m \times n$. Будем говорить, что пара (f, g) векторов с целочисленными компонентами *представляет* M, если

(1) $f = (f_1, f_2, \ldots, f_m)$,
(2) $g = (g_1, g_2, \ldots, g_n)$,
(3) $f_i < g_j$, если $M_{ij} = -1$,
$f_i = g_j$, если $M_{ij} = 0$,
$f_i > g_j$, если $M_{ij} = +1$.

Можно использовать f и g вместо M следующим образом. Для того чтобы определить M_{ij}, отыщем f_i и g_j. Если $f_i < g_j$, $f_i = g_j$ или $f_i > g_j$, положим $M_{ij} = -1$, 0 или $+1$ соответственно. Заметим, что при таком использовании f и g вместо M у матрицы не будет пустых элементов, потому что между f_i и g_j всегда выполняется одно из отношений $<$, $=$ или $>$.

Векторы f и g будем называть *функциями предшествования* для M. Используя f и g для представления M, можно сократить объем памяти, требующейся для матрицы предшествования, с $m \times n$ до $m + n$ элементов. Следует, однако, отметить, что не для всякой матрицы предшествования функции предшествования существуют.

Пример 7.1. Рассмотрим грамматику простого предшествования G с правилами

$$S \longrightarrow aSc \mid bSc \mid c$$

Отношения предшествования Вирта—Вебера для G образуют матрицу, изображенную на рис. 7.1. Мы будем называть ее *матрицей отношений предшествования Вирта—Вебера*, чтобы избежать путаницы с термином „матрица предшествования". Отношения предшествования, показанные на рис. 7.1, можно представить также в виде матрицы предшествования (рис. 7.2). Можно далее

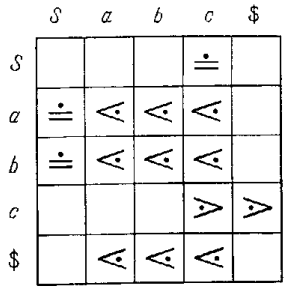

Рис. 7.1. Матрица отношений предшествования Вирта — Вебера.

		1	2	3	4	5
		S	a	b	c	$
1	S				0	
2	a	0	−1	−1	−1	
3	b	0	−1	−1	−1	
4	c				+1	+1
5	$		−1	−1	−1	

Рис. 7.2. Матрица предшествования M.

представить эту матрицу предшествования функциями предшествования

$$f = (1,\ 0,\ 0,\ 2,\ 0)$$
$$g = (0,\ 1,\ 1,\ 1,\ 0)$$

Нетрудно убедиться, что это действительно функции предшествования для M. Например, $f_4 = 2$, $g_5 = 0$ и, значит, поскольку $f_4 > g_5$, f и g действительно представляют элемент M_{45} со значением $+1$.

Элемент M_{41} матрицы предшествования имеет значение „пусто". Однако $f_4 = 2$, а $g_1 = 0$. Таким образом, если мы будем использовать f и g для представления M, надо будет заменить значение M_{11} на $+1$ (так как $f_4 > g_1$). Аналогично пустые элементы M_{11}, M_{15}, M_{42} и M_{43} все будут иметь значение $+1$, а M_{12}, M_{13}, M_{25}, M_{35}, M_{51}, M_{55} получат значение 0.

Пустые элементы в исходной матрице предшествования указывают на ошибочные ситуации. Поэтому, если использовать функции предшествования для представления отношений предшествования, утратится возможность обнаружения ошибки в том случае, когда не выполняется ни одно из трех отношений предшествования. Но эта ошибка все равно выявится при попытке

выполнить свертку, когда окажется, что нет правила с такой правой частью, которая находится в верхушке магазина. Тем не менее такая задержка в обнаружении ошибки может оказаться неприемлемо дорогой платой за удобство использования функций предшествования вместо матриц предшествования; это зависит от того, насколько важно раннее обнаружение ошибки в конкретном компиляторе. □

Пример 7.2. Потерю возможности своевременно обнаруживать ошибки можно в значительной степени компенсировать, построив для грамматики предшествования алгоритм разбора типа „перенос—свертка", в котором с отношениями $\lessdot$ и $\doteq$ будет связана

		1	2	3	4
		a	b	c	\$
1	S			−1	
2	a	−1	−1	−1	
3	b	−1	−1	−1	
4	c			+1	+1
5	\$	−1	−1	−1	

Рис. 7.3. Матрица предшествования M'.

операция **перенос**, а с отношением $\gtrdot$ — операция **свертка**. Кроме того, для построения функции действия алгоритма разбора типа „перенос—свертка" требуются только отношения предшествования с областью определения $N \cup \Sigma \cup \{\$\}$ и множеством значений $\Sigma \cup \{\$\}$. Например, можно сопоставить $\lessdot$ и $\doteq$ с -1, $\gtrdot$ с $+1$ и получить из матрицы, приведенной на рис. 7.1, матрицу предшествования M' (рис. 7.3). Пустые элементы обозначают ошибочные ситуации. Можно показать, что векторы

$$f = (0, 0, 0, 2, 0) \quad \text{и} \quad g = (1, 1, 1, 0)$$

— функции предшествования для M'. Эти функции предшествования обладают тем преимуществом, что для пустых элементов M_{14}, M_{24}, M_{34} и M_{54} обе имеют значение 0 ($f_1 = f_2 = f_3 = f_5 = g_4 = 0$). Поэтому, обозначая нулем ошибочную ситуацию, мы сохраняем возможность обнаружения ошибки, заложенную в исходной матрице M'. Подробнее мы рассмотрим эту проблему в разд. 7.1.3. □

Сначала мы дадим алгоритм, который по данной матрице предшествования M находит функции предшествования для M,

если они существуют. В следующем разделе опишем модификацию этого алгоритма, где по данной матрице предшествования с элементами -1, $+1$ и „пусто" строятся функции предшествования для этой матрицы, которые по возможности чаще представляют пустые элементы нулями.

Прежде всего заметим, что если две строки матрицы предшествования M совпадают, то их можно слить в одну строку, и это не повлияет на факт существования функций предшествования для M. По той же причине можно слить совпадающие столбцы. Матрицу предшествования, в которой все совпадающие строки и столбцы слиты, назовем *приведенной* матрицей предшествования. Ясно, что, если сначала привести матрицу предшествования, легче будет строить функции предшествования.

Алгоритм 7.1. Вычисление функций предшествования.

Вход. Матрица M размера $m \times n$ с элементами -1, 0, $+1$ и „пусто".

Выход. Два целочисленных вектора $f = (f_1, \ldots, f_m)$ и $g = (g_1, \ldots, g_n)$, у которых

$$\begin{aligned} f_i < g_j \quad &\text{при} \quad M_{ij} = -1 \\ f_i = g_j \quad &\text{при} \quad M_{ij} = 0 \\ f_i > g_j \quad &\text{при} \quad M_{ij} = +1 \end{aligned}$$

или „нет", если такие векторы не существуют.

Метод.

(1) Построим ориентированный граф, содержащий не более $m+n$ вершин и называемый *графом линеаризации* для M. Сначала пометим m вершин буквами $F_1, F_2, \ldots, F_m$, а оставшиеся n вершин буквами $G_1, G_2, \ldots, G_n$. В дальнейшем граф будет преобразовываться, причем в каждый момент будут существовать некоторая вершина $\hat{F}_i$, *представляющая* F_i, и некоторая вершина $\hat{G}_j$, *представляющая* G_j. Вначале $\hat{F}_i = F_i$ и $\hat{G}_j = G_j$ для всех i и j. Затем для всех i и j выполним шаг (2) или (3).

(2) Если $M_{ij} = 0$, построим новую вершину N, сливая $\hat{F}_i$ с $\hat{G}_j$. Теперь N представляет все те вершины, которые ранее представлялись вершинами $\hat{F}_i$ и $\hat{G}_j$.

(3) Если $M_{ij} = +1$, проведем дугу из $\hat{F}_i$ в $\hat{G}_j$. Если $M_{ij} = -1$, проведем дугу из $\hat{G}_j$ в $\hat{F}_i$.

(4) Если полученный граф содержит циклы, выдаем сообщение „нет".

(5) Если граф линеаризации ациклический, положим значение f_i равным длине самого длинного пути, начинающегося в $\hat{F}_i$, а g_j — длине самого длинного пути, начинающегося в $\hat{G}_j$. □

На шаге (4) алгоритма 7.1 для выяснения, циклический или нет ориентированный граф G, можно применить следующий общий метод:

(1) Пусть G — исходный граф.

(2) Найти в данном графе вершину N, не имеющую прямых потомков. Если таких вершин нет, граф G — циклический. В противном случае удалить N [1]).

(3) Если полученный граф пуст, то граф G — ациклический. В противном случае повторить шаг (2).

Установив в некоторый момент, что граф ациклический, воспользуемся для нахождения на шаге (5) алгоритма 7.1 длины самого длинного пути, начинающегося в произвольной вершине, следующим методом разметки вершин.

Пусть G — ориентированный ациклический граф.

(1) Сначала пометить все вершины графа G нулями.

(2) Повторять шаг (3) до тех пор, пока метки графа G не перестанут изменяться. Метка при каждой вершине будет равна длине самого длинного пути, начинающегося в этой вершине.

(3) Выбрать в G некоторую вершину N. Пусть N имеет прямых потомков $N_1, N_2, \ldots, N_k$ с метками $l_1, l_2, \ldots, l_k$. Заменить метку вершины N на $\max\{l_1, l_2, \ldots, l_k\}+1$. (Если $k=0$, то меткой вершины N оставить 0.)

Проделать этот шаг для всех вершин графа G.

Ясно, что шаг (3) для каждой вершины повторяется не более l раз, где l — длина самого длинного пути в G.

Пример 7.3. Рассмотрим матрицу предшествования M, изображенную на рис. 7.4.

	1	2	3	4	5
1	−1	−1	0		−1
2			+1	−1	0
3	−1		+1	0	
4		−1	+1		
5				+1	+1

Рис. 7.4. Матрица предшествования.

Граф линеаризации, построенный по M, показан на рис. 7.5. Заметим, что на шаге (2) алгоритма 7.1 сливаются три пары вершин: (F_3, G_4), (F_2, G_5) и (F_1, G_3).

[1]) Строго говоря, надо удалить также и дуги, входящие в N. — *Прим. перев.*

Граф линеаризации — ациклический. В результате выполнения шага (5) алгоритма 7.1 получаются функции предшествования $f=(0, 1, 2, 1, 3)$ и $g=(3, 2, 0, 2, 1)$. Например, $f_5=3$, поскольку длина самого длинного пути, начинающегося в вершине F_5, равна 3. □

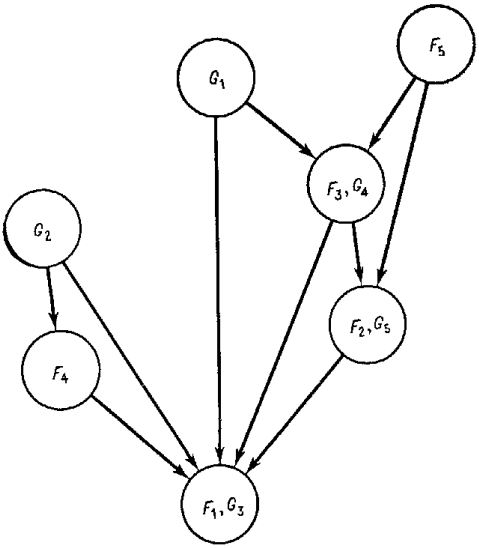

Рис. 7.5. Граф линеаризации.

Теорема 7.1. *Матрица предшествования имеет функции предшествования тогда и только тогда, когда ее граф линеаризации ациклический.*

Доказательство. *Достаточность.* Сначала отметим, что алгоритм 7.1 выдает на выходе функции f и g только тогда, когда граф линеаризации ациклический. Достаточно показать, что если f и g вычисляются при помощи алгоритма 7.1, то

(1) $M_{ij}=0$ означает, что $f_i=g_j$,
(2) $M_{ij}=+1$ означает, что $f_i>g_j$,
(3) $M_{ij}=-1$ означает, что $f_i<g_j$.

Утверждение (1) сразу следует из шага (2) алгоритма 7.1. Для того чтобы доказать утверждение (2), заметим, что при $M_{ij}=+1$ к графу линеаризации добавляется дуга $(\hat{F}_i, \hat{G}_j)$. Следовательно, $f_i>g_j$, так как, если граф линеаризации ацикличе-

ский, длина самого длинного пути, начинающегося в вершине $\hat{F}_i$, должна быть по крайней мере на единицу больше длины самого длинного пути, начинающегося в $\hat{G}_j$. Утверждение (3) доказывается аналогично.

Необходимость. Допустим, что матрица предшествования M имеет функции предшествования f и g, но граф линеаризации содержит цикл, состоящий из последовательности вершин $N_1, \ldots, N_k, N_{k+1}$, где $N_{k+1} = N_1$ и $k \geqslant 1$. Тогда на шаге (3) для всех i, $1 \leqslant i \leqslant k$, можно найти такие вершины H_i и I_{i+1}, что

(1) H_i и I_{i+1} — вершины, вначале помеченные буквами F и G,
(2) H_i и I_{i+1} представляются вершинами N_i и N_{i+1} соответственно,
(3) либо H_i есть F_r, I_{i+1} есть G_s и $M_{rs} = +1$, либо H_i есть G_r, I_{i+1} есть F_s и $M_{sr} = -1$.

Из правила (2) алгоритма 7.1 видно, что если вершины F_r и G_s представляются одной и той же вершиной N_i, то f_r должно равняться g_s, если f и g — функции предшествования для M. Предположим, что f и g — функции предшествования для M. Пусть $h_i = f_r$, если H_i есть F_r, и $h_i = g_r$, если H_i есть G_r. Положим h_i' равным f_r, если I_i есть F_r, и равным g_r, если I_i есть G_r. Тогда

$$h_1 > h_2' = h_2 > h_3' = \ldots = h_k > h_{k+1}'$$

Но так как N_{k+1} совпадает с N_1, то $h_{k+1}' = h_1$. Однако уже было показано, что $h_1 > h_{k+1}'$. Таким образом, матрица предшествования с циклическим графом линеаризации не может иметь функций предшествования. □

Следствие. *Алгоритм 7.1 вычисляет функции предшествования для M, если они существуют, и выдает ответ „нет" в противном случае.* □

7.1.2. Применения к разбору, основанному на операторном предшествовании

Для любой матрицы, элементы которой принимают не более трех значений, можно попытаться найти функции предшествования. Описанный метод применим независимо от того, что именно представляют элементы. Чтобы проиллюстрировать это, покажем сейчас, как применяются функции операторного предшествования для представления отношений операторного предшествования.