Номер 3 **Май - Июнь 1995**

ISSN 0132-3474

РОССИЙСКАЯ АКАДЕМИЯ НАУК

ПРОГРАММИРОВАНИЕ

Главный редактор
В.П. Иванников

МАИК НАУКА

ПРОГРАММИРОВАНИЕ, 1995, N° 3, с.3 - 8

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ ПРОГРАММИРОВАНИЯ

УДК 519.685.3

АЛГОРИТМЫ ВЫЧИСЛЕНИЯ АТРИБУТОВ В АТРИБУТНЫХ ГРАММАТИКАХ *

© 1995 г. **В. М. Курочкин**
Вычислительный Центр Российской академии наук
117967 Москва, ул. Вавилова, 40
Поступила в редакцию 28.12.94 г.

Рассматривается задача вычисления атрибутов на дереве вывода для атрибутных грамматик. Предлагаемые алгоритмы не предполагают никаких ограничений на грамматики и могут выполняться при любых обходах деревьев вывода.

1. ВВЕДЕНИЕ

Атрибутные грамматики (см. [1]) – один из самых универсальных механизмов определения семантики языков программирования. Из существенных его достоинств отметим то, что он:

1. базируется на контекстно свободном описании синтаксиса языка (который широко используется и общепризнан, как наиболее удобный способ определения синтаксиса) и

2. носит компилятивный характер и ориентирован на использование в трансляторах.

Атрибутные грамматики (АГ) особенно удобны для задания и проверки контекстных зависимостей в языке, они могут быть также использованы для оптимизации и генерации кода транслируемых программ (см., например, [2]). На практике применение АГ в трансляторах в какой-то степени затрудняется сложностью процедуры вычисления всех атрибутов на дереве вывода, полученного в результате синтаксического анализа программы. Было предложено несколько алгоритмов вычисления атрибутов, но почти все они выполняются при тех или иных ограничениях на АГ, часто неприемлемых на практике, требуют значительных дополнительных усилий и, что самое главное, дополнительных неформальных преобразований АГ. Из известных универсальных алгоритмов приемлемым можно считать только так называемый "метод визитов" (см. [3]). Однако и он по сравнению с предложенным далее обладает такими недостатками, как необходимость предварительного построения всего дерева вывода, фиксированного порядка обхода этого дерева и вычисления всех, даже порой ненужных, атрибутов.

В настоящей статье излагается подход, который позволяет построить несколько универсальных алгоритмов вычисления атрибутов, учитывающих те или иные особенности АГ, дополнительные требования к памяти ЭВМ, к скорости выполнения алгоритма, специфику выдаваемой информации и др. Можно считать, что подход инспирирован логикой работы машины, управляемой потоком данных, и описываемые нами алгоритмы рассматривать, как формализацию процесса вычисления атрибутов, протекающего в такой машине. Один из вариантов такого алгоритма был описан нами ранее [4], но, по-видимому, в силу сложности алгоритма, отражающей сложность логики функционирования машины, управляемой данными, содержал ошибку и не удовлетворял всем декларированным там свойствам. Поэтому мы сочли необходимым привести здесь и доказательство правильности наиболее сложного варианта алгоритма.

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований, проект 93-01-00573.

2. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И ОБОЗНАЧЕНИЯ

Основой атрибутной грамматики АГ является контекстно-свободная (КС) грамматика $G(V,T,P,S)$, где V – полный алфавит, T – терминальный алфавит, P – совокупность правил грамматики, имеющих вид: $p : a_0 \longrightarrow a_1 \ldots a_l$, S – аксиома.

Атрибутная грамматика $AG(G,X,F)$ – это КС-грамматика G, в которой:

1. каждому символу a из V сопоставлено конечное подмножество X_a множества X переменных $x_1, x_2, \ldots$, называемых далее атрибутами;

2. каждому правилу $p : a_0 \longrightarrow a_1 \ldots a_l$ из P сопоставлено конечное множество F_p функциональных зависимостей $y = f(x \ldots)$, выражающих значения одних из атрибутов символов $a_0, a_1, \ldots, a_l$, образующих данное правило p, через значения других атрибутов $x \ldots$ того же набора $a_0, a_1, \ldots, a_l$ символов.

Основная задача, встающая при работе с АГ – это, коль скоро дано какое-то дерево вывода, вычислить значения всех атрибутов для всех узлов дерева.

В работе [1] приводятся необходимые и достаточные условия, которым должна удовлетворять АГ для того, чтобы на любом дереве вывода можно было вычислить все атрибуты. И хотя формально проверка этих условий весьма трудоемка (сложность задачи – экспоненциальная), применение этой процедуры для установления корректности описания языка программирования с помощью АГ вполне приемлемо.

В процессе выполнения описываемых далее алгоритмов вычисления атрибутов создается и трансформируется некоторый граф. Узлы X, Y, Z, графа соответствуют атрибутам x, y, z, вершин дерева вывода. С каждым узлом X связано несколько величин (узел удобно представлять себе как запись, а эти величины – как поля записи):

x – соответствующий узлу X атрибут;

a – логическая переменная. 0 (или 1) означает, что значение атрибута x еще не (или уже) вычислено;

b – логическая переменная. 0 (или 1) означает, что необходимость в значении атрибута x пока еще не (уже) ясна. Может оказаться, что для данного конкретного дерева вывода нет необходимости вычислять все атрибуты, так как конечной целью является получение значений так называемых "выходных" атрибутов, а другие атрибуты вычисляются лишь постольку, поскольку они необходимы для вычисления "выходных". У ненужных атрибутов величина b так до конца и остается равной 0, а атрибут вычисляться не будет;

c – логическая переменная. $c = 1$ означает, что атрибут является выходным. После завершения работы алгоритма (или в процессе его выполнения, как только выяснится, что какой-либо атрибут сам не является выходным и не нужен для вычисления значений других выходных атрибутов) все узлы с $c = 0$ можно ликвидировать. Если $c = 1$, то, конечно, и $b = 1$;

n – целочисленная переменная, которая принимает значения 0, 1 или 2, указывающие, сколько раз еще придется иметь дело с информацией в данном узле;

f – указатель на функцию, по которой можно вычислить значение данного атрибута, либо *null*, если эта функция пока еще не известна.

Узлы графа связываются ориентированными дугами. Дуга $Y \rightarrow X$ означает, что атрибут y непосредственно зависит от атрибута x, т.е. что $y = f(\ldots x \ldots)$. Дуги могут быть четырех цветов: черные, синие, красные и зеленые. Предполагается, что сначала построен черный граф, содержащий все узлы-атрибуты всех вершин дерева вывода и все зависимости между атрибутами. Черный граф служит исключительно наглядным целям, он как бы является фоном для выполняемых построений. (В дальнейшем черный граф используется также для доказательства корректности алгоритма). Можно считать, что его нет, а в описываемых далее преобразованиях графа слова "черная дуга заменяется на синюю" читать "проводится синяя дуга". Смысл дуг следующий:

синяя дуга $Y \to X$, идущая от Y к X, означает, что есть прямая зависимость атрибута y от x: $y = f(\ldots x \ldots)$, но пока неизвестно, надо ли будет вычислять y, так как $Y.b = 0$;

зеленая дуга $Y \to X$ означает, что есть прямая зависимость y от x: $y = f(\ldots x \ldots)$, причем значение x уже известно ($X.a = 1$), y пока не вычислено ($Y.a = 0$), но y надо будет вычислять ($Y.b = 1$), так как от него прямо или косвенно зависит один из выходных атрибутов;

красная дуга, идущая от Y к X, означает, что есть прямая зависимость y от x, причем значение x еще не вычислено ($X.a = 0$), а y надо будет вычислять ($Y.b = 1$), как только будут известны значения всех его аргументов (атрибутов, от которых непосредственно зависит y).

3. АЛГОРИТМ ПОСТРОЕНИЯ И ПЕРЕСТРОЕНИЯ ГРАФА

Вначале граф пустой (или построен черный граф, который, как было сказано, служит лишь фоном для последующих перестроений графа). Алгоритм не связан с каким-либо фиксированным порядком обхода дерева вывода. На вход алгоритма в произвольном порядке подаются все ветвления $p : U_0 \longrightarrow U_1 U_k$ дерева. Для каждого ветвления выполняется

Ветвление(p):

В1. Если вершина U_j $(j = 0, 1, \ldots, k)$ дерева еще не обрабатывалась, то в графе добавляются узлы X, соответствующие всем атрибутам x вершины U_j. При этом полагается $X.f = null$; $X.a = 0$; если атрибут x выходной, то $X.b = X.c = 1$, иначе $X.b = X.c = 0$. Если U_j – терминальная вершина или аксиома, то $X.n = 1$, иначе $X.n = 2$.

В2. Для каждой связанной с p функциональной зависимостью $y = f(x_1 \ldots x_m)$ выполняется *Зависимость*$(Y, f, X_1, \ldots, X_m)$, в результате чего, если нужно и можно, то вычисляется атрибут y, и происходит необходимое перестроение графа.

В3. Для всех атрибутов x всех вершин U_j $(j = 0, 1, \ldots, k)$ выполняется $X.n := X.n - 1$.

В4. Для всех атрибутов x всех вершин U_j $(j = 0, 1, \ldots, k)$ выполняется *Ли – граф*(X), в результате чего будет ликвидирован узел X, если он уже не нужен для вычисления других атрибутов и сам не является выходным, а также все, что можно ликвидировать "ниже" X. (Здесь и далее "ниже" и "выше" означает движение в графе вдоль или против стрелок, что не может привести к недоразумениям, так как фоновый черный граф не содержал циклов ввиду незацикленности АГ).

Конец *ветвления*.

Зависимость$(Y, f, X_1, \ldots, X_m)$:

З1. В узел Y помещается указатель на функцию f : $Y.f := f$.

З2. Для каждого $i = 1, \ldots, m$ черная дуга $Y \to X_i$ заменяется на синюю (без фонового черного графа – проводится синяя дуга $Y \to X_i$).

З3. Если $Y.b = 0$, то конец *Зависимости* (так как y вычислять пока не надо).

З4. Выполняется *Ниже*(Y), перестраивая граф, расположенный ниже Y, включая Y.

З5. Выполняется *Выше*(Y), перестраивая граф, расположенный выше Y.

Конец *Зависимости*.

Ниже(Y) (принять во внимание, что $Y.b = 1$):

Н1. Для каждой выходящей из Y синей дуги $Y \to X$

Н1.1. Если $X.b = 0$, то $X.b := 1$, иначе переход к Н1.3.

Н1.2. Если $X.f/ = null$ и $X.a = 0$, то выполнить *Ниже*(X).

Н1.3. Если $X.a = 1$, то синяя дуга $Y \to X$ заменяется на зеленую, иначе (если $X.a = 0$) – на красную.

Н2. Если из узла Y не выходят красные дуги (т.е. выходят только зеленые или никаких нет), то вычисляется y, делается $Y.a = 1$, ликвидируются все выходящие из Y дуги $Y \to X$, и для каждой из них выполняется *Лик – уз*(X), т.е. ликвидируется, если можно, узел Y.

Конец *Ниже*.

Выше(Y):

Вы1. Если $Y.a = 1$ и в Y входят красные дуги $Z \to Y$, то

Вы1.1. Красная дуга заменяется на зеленую.

Вы1.2. Если из Z не выходит красных дуг, то вычисляется z, делается $Z.a = 1$, ликвидируются все выходящие из Z дуги $Z \to X$, для них выполняется *Лик – уз*(X).

Вы1.3. Выполняется *Выше*(Z).

Конец *Выше*.

Лик – уз(X) (ликвидация узла, в котором вычислено значение атрибута и который более не нужен):

Если выполнены условия

$$X.a = 1,\ X.c = 0,\ X.n = 0,$$

в X не входят синие и зеленые дуги (красные не могут, ибо $X.a = 1$),

то узел X ликвидируется.

Ли – граф(Y) (ликвидация узла и расположенной ниже его части графа):

Если выполняются условия

$$Y.n = 0,\ Y.c = 0,$$

в узел Y не входят никакие дуги,

то для всех выходящих их него синих (другие не могут) дуг $Y \to X$ ликвидируется эта дуга $Y \to X$, выполняется *Ли – граф*(X) и ликвидируется узел Y.

Выполнение алгоритма завершается после того, как на его вход (*Ветвление*) последовательно будут поданы все ветвления дерева вывода. В результате работы алгоритма от преобразуемого графа атрибутов останутся лишь узлы X, для которых $X.c = 1$, причем для всех этих узлов будет вычислено значение соответствующего атрибута ("выходные атрибуты").

4. ОБОСНОВАНИЕ АЛГОРИТМА

Черный граф содержит все зависимости атрибутов. Поэтому можно было бы как-нибудь вычислить значения всех атрибутов (что возможно в силу предполагаемой корректности атрибутной грамматики), а затем удалить в графе все дуги и все узлы, соответствующие "невыходным" атрибутам (т.е. те узлы X, для которых $X.c = 0$). Нужно показать, что результат работы алгоритма будет такой же.

Одним этапом выполнения алгоритма будем считать полную обработку одного ветвления, а тот вид, который при этом примет граф – каноническим. Слово "граф" употребляется здесь корректно, так как при ликвидации узла X (*Лик – уз* и *Ли – граф*) проверяется, что в X не входят никакие дуги, а все выходящие из X дуги предварительно тоже ликвидируются (*Ниже*, *Выше* и *Ли – граф*).

В каноническом (в частности, в заключительном) графе выполняются следующие легко проверяемые свойства.

1. Если $X.a = 1$, то из X не выходят никакие дуги (Н2 и Вы1.2) и в X не могут входить красные (Н1.3 и Вы1).

2. Если дуга $Y \to X$ красная или зеленая, то $X.b = Y.b = 1$ (З3, Н1.1 - Н1.3).

3. Дуга $Y \to X$ синяя, т.е. $Y.b = 0$ (З3 - З4 и Н1).

4. Любой путь в графе имеет вид

   [синяя дуга...] [красная дуга...] [зеленая дуга],

   где ...означает возможное повторение, а [ ] – возможное отсутствие. Это следует из свойств 1 - 3.

5. В любом узле X рано или поздно станет $X.n = 0$ (ликвидирован узел может быть только если $X.n = 0$; каждый узел, как и каждая вершина дерева, будет обработан соответствующее – 1 или 2 – число раз: В1 и В3).

6. Для узла X, если $X.c = 1$, то $X.b = 1$ (В1).

7. Если $X.a = 1$, то $X.b = 1$ (вычисление значения атрибута x и присваивание $X.a := 1$ выполняются только при $X.b = 1$).

Лемма 1. Не могут сохраниться узлы X со значениями полей $a = b = c = 0$.

Доказательство. Если это не так, то от узла X пройдем навстречу стрелок сколько можно. В силу свойств 2 и 3, это будут только синие дуги. У начала Y этого пути $Y.a = Y.b = Y.c = Y.n =$

0. Рассмотрим тот момент, когда стало $Y.n = 0$ (В3). В4 обращается к $Ли - граф(Y)$. Узел Y при этом не был ликвидирован потому, что в него входили (синие) дуги. В какой-то момент исчезла последняя из них – $Z \to Y$. Это могло быть только при выполнении $Ли - граф(Z)$. Но тогда $Ли - граф$ рекурсивно ликвидировал бы и узел Y. Лемма 1 доказана.

Лемма 2. Если поле $X.b$ узла X стало равно 1, то и поле $X.a$ станет равным 1 (т.е. если возникла необходимость в значении атрибута x, то это значение рано или поздно будет вычислено).

Доказательство. Поле $X.b$ стало равным 1 либо потому, что $X.c = 1$ (В1), либо потому, что была проведена красная дуга $Y \to X$ (Н1.1 и Н1.3). Если бы x не было вычислено, то красная дуга $Y \to X$ и узел X не были бы ликвидированы. Как и в случае $X.c = 1$, они сохранились бы в конечном графе. Проведем из X путь до предела. Он будет весь состоять из красных дуг, а в конце его может быть зеленая дуга (свойство 4) $Y \to Z$. У всех узлов этого пути, кроме Z, поля $a = 0$, $b = 1$, $Z.a = Z.b = 1$. В момент обработки ветвления, содержащего присваивание значения атрибуту y, были проведены синие дуги ко всем аргументам атрибута y. Раз сохранился узел Y, причем $Y.a = 0$, то не была ликвидирована и ни одна из этих дуг. Если же было $Y.b = 1$, то выполнилось $Ниже(Y)$, а если нет, то $Ниже(Y)$ выполнилось позже при присваивании $Y.b = 1$. Если красный путь не был продолжен, то все выходящие из Y дуги были сделаны зелеными, и вычислен y. Если остались красные дуги, то они все были перекрашены в зеленые позже, и $Выше$ все равно вычислила бы y. Таким образом, $Y.a$ в любом случае стало бы равно 1. Все это относится и к случаю, когда красный путь пустой и $X = Y$. Лемма 2 доказана.

Лемма 3. Не могут сохраниться узлы X, у которых $X.c = 0$, $X.a = X.b = 1$.

Доказательство. При вычислении x ($X.a = 1$) ликвидируются все выходящие из X дуги, а все входящие красные превращаются в зеленые (свойство 1). Зеленые дуги $Y \to X$ сохраниться не могут, ибо у них $Y.b = 1$ (свойство 2), а согласно лемме 2, $Y.a$ тоже станет равно 1, но тогда будет ликвидирована и дуга $Y \to X$. Если дуга $Y \to X$ синяя, то $Y.a = Y.b = Y.c = 0$, а такой узел либо будет ликвидирован (со всеми выходящими дугами), либо изменит свой "статус", т.е. значение $Y.b$. Тогда изменит цвет и дуга и, как было показано выше, дуга тоже будет ликвидирована. После ликвидации каждой дуги $Y \to X$ делается попытка ликвидировать и узел X (выполняется $Ли-уз(X)$). Тогда при $X.n = 0$ ликвидация последней дуги вызовет и ликвидацию узла X. Если же $X.n$ становится равным 0 в результате В3 без создания дуг, то ликвидацию узла X вызовет обращение к $Ли - граф(X)$ на шаге В4. Лемма 3 доказана.

Следствие. Из лемм 1 - 3 следует, что сохранятся те и только те узлы X, у которых $X.c = 1$, причем во всех этих узлах будет вычислено значение атрибута; все дуги также будут ликвидированы.

5. СВОЙСТВА И ОСОБЕННОСТИ АЛГОРИТМА

Формальная запись алгоритма во многом, конечно, зависит от применяемого аппарата, в частности, от инструментального языка программирования, от имеющихся в нем структур данных и возможностей. Следует учитывать и то, что алгоритм использует рекурсию. От свойств инструментального языка зависит и скорость работы алгоритма. Скорость работы и объем требуемой оперативной памяти зависят также от используемых вариаций этого алгоритма (о них ниже) и от способа его применения. Дело в том, что, как говорилось во введении, алгоритм допускает обработку ветвлений дерева вывода в любом порядке. Его можно подключать даже одновременно с синтаксическим анализом с учетом, конечно, того, что каждое ветвление дерева вывода должно подаваться на вход алгоритма только один раз (например, при любом детерминированном и без возвратов процессе построения дерева). Однако время работы алгоритма и особенно объем используемой памяти будут зависеть от порядка обработки ветвлений дерева. Правда, время в любом случае будет линейно зависеть от размера дерева, точнее – от числа вершин в нем, с константой, определяемой лишь параметрами атрибутной грамматики. Следует еще отметить, что алгоритм (и этим он существенно отличается от всех известных нам других алгоритмов вычисления атрибутов) будет вычислять значения лишь тех атри-

бутов, от которых фактически зависят значения нужных ("выходных") атрибутов, причем будет вычислять атрибут сразу, как только для этого создались необходимые условия (т.е. как только получены значения всех его аргументов) и будет ликвидировать атрибут (узел), как только минет необходимость в его дальнейшем существовании. Повлиять на скорость и требуемую память можно также, положив с самого начала, т.е. в части *Выполнить*, $b = 1$ для тех атрибутов, значения которых в любом случае необходимо вычислять (предварительный формальный или смысловой анализ атрибутной грамматики легко может обнаружить – пусть даже не все – такие атрибуты, а "ошибиться" здесь нельзя, ибо это связано с "оптимизацией" использования алгоритма и никак не влияет на конечный результат).

В заключение приведем максимально упрощенный и редуцированный вариант алгоритма, получающийся из полного, если в нем положить для всех атрибутов $c = 1$ и исключить из выкладок поле n. Ненужными тогда становятся *Лик – уз* и *Ли – граф*.

6. УПРОЩЕННЫЙ ВАРИАНТ АЛГОРИТМА

Ветвление(p):

В1. Если вершина U_j ($j = 0, \ldots, k$) еще не обрабатывалась, то в граф добавляются узлы X, соответствующие всем атрибутам x вершины U_j. При этом полагается $X.a = 0$, $X.f = null$.

В2. Для каждой связанной с p функциональной зависимостью $y = f(x_1, \ldots, x_m)$ выполняется *Зависимость*($Y, f, X_1, \ldots, X_m$).

Зависимость($Y, f, X_1, \ldots, X_m$):

З1. В узел Y помещается указатель на функцию: $Y.f := f$.

З2. Для каждого $i = 1, \ldots, m$ проводится синяя дуга $Y \to X_i$.

З3. Выполняется *Ниже*(Y).

З4. Выполняется *Выше*(Y).

Ниже(Y):

Н1. Для каждой выходящей из Y синей дуги $Y \to X$

Н1.1. Если $X.f = null$ и $X.a = 0$, то выполнить *Ниже*(X).

Н1.2. Синяя дуга заменяется на зеленую, если $X.a = 1$, либо на красную, если $X.a = 0$.

Н2. Если из узла Y не выходят красные дуги, то вычисляется y и делается $Y.a = 1$.

Выше(Y):

Вы1. Если $Y.a = 1$ и в Y входит красные дуги $Z \to Y$, то

Вы1.1. Красная дуга $Z \to Y$ заменяется на зеленую.

Вы1.2. Если из Z не выходят красные дуги, то вычисляется z и делается $Z.a = 1$.

Вы1.3. Выполняется *Выше*(Z).

Так же, как и в полном алгоритме, на вход в произвольном порядке подаются все ветвления $p : U_0 \longrightarrow U_1 \ldots U_k$ дерева вывода. После окончания работы алгоритма в памяти останутся все узлы-атрибуты, и у всех будут вычислены значения. Как в полном, так и в упрощенном алгоритмах ликвидируются все дуги. Если необходимо сохранить какую-то привязку к дереву вывода, то это, по-видимому, лучше делать на уровне дерева, включив соответствующие дополнения и в часть *Ветвление*.

СПИСОК ЛИТЕРАТУРЫ

1. *Knuth D.E.* Semantics of Context-Free Languages // Math. Systems Theory. 1968. V.2. N° 2.

2. *Курочкин В.М., Серебряков В.А.* Вопросы разработки и использования атрибутных систем построения трансляторов. Сборник "Проблемы прикладной математики и информатики". М. Наука. 1987.

3. *Riis H., Skyum S.* K-Visit Attribute Grammars // Math. Systems Theory. 1981. V.15. PP.17 - 28.

4. *Kurochkin V.M.* A Universal Economical Algorithm of Attribute Evaluation. Informatics'88. Proceedings of the French-Soviet Symposium. INRIA. Nice. 1988.