Стандартные схемы программ

Дисциплина «Теория вычислительных процессов»

Программы и схемы программ

Схема программы – это абстрактная модель программы, которая описывает только структуру программы и не задает никаких типов данных, никаких конкретных функций и предикатов, и никаких конкретных значений переменных и констант.

Пример:

```
begin integer x, y;
  ввод(x);
   y:=1;
L: if x=0 then goto L1
  y:=x*y;
  x:=x-1;
   goto L;
L1: вывод(y);
   end
```

```
begin integer x, y;
    ввод(x);
     y:=e;
L: if x=0 then goto
    y:=CONSCAR(x, y);
    x:=CDR(x);
     goto
L1: вывод(y);
     end
```

Базис класса стандартных схем программ

Базис В класса стандартных схем состоит:

1)Множества символов полного базиса:

1. $X = \{x, x_1, x_2..., y, y_1 y_2..., z, z_1, z_2...\}$ - множество символов, называемых *переменными*;

2. $F = \{f^{(0)}, f^{(1)}, f^{(2)}..., g^{(0)}, g^{(1)}, g^{(2)}..., h^{(0)}, h^{(1)}, h^{(2)}...\}$ - множество *функциональных символов*; верхний символ задает *местность символа*; нульместные символы называют константами и обозначают начальными буквами латинского алфавита *a, b, c*...;

3. $P = \{p^{(0)}, p^{(1)}, p^{(2)}...; q^{(0)}, q^{(1)}, q^{(2)}...; \}$ - множество *предикатных символов*; $p^{(0)}, q^{(0)}$ - ; нульместные символы называют логическими константами;

4. {**start, stop, ..., :=** и т. д.} - множество специальных символов.

Базис класса стандартных схем программ

2)Термами (функциональными выражениями) :

1. односимвольные слова, состоящие из переменных или констант;

2. слово τ вида $f^{(n)}(\tau_1, \tau_2, \ldots, \tau_n)$, где $\tau_1, \tau_2, \ldots, \tau_n$ - термы;

Примеры термов: x, $f^{(0)}$, a, $f^{(1)}(x)$, $g^{(2)}(x, h^{(3)}(y, a))$.

Базис класса стандартных схем программ

3)Тестами (логическими выражениями) называются логические константы и слова вида $p^{(n)}(\tau_1, \tau_2, \ldots, \tau_n)$. Примеры: $p^{(0)}$, $p^{(0)}(x)$, $g^{(3)}(x, y, z)$, $p^{(2)}(f^{(2)}(x, y))$.

4)Множество операторов включает пять типов:

1. ***начальный оператор*** - слово вида **start**$(x_1, x_2 \ldots x_k)$, где $k \geq 0$, а $x_1, x_2 \ldots x_k$ - переменные, называемые результатом этого оператора;

2. ***заключительный оператор*** - слово вида **stop**$(\tau_1, \tau_2, \ldots, \tau_n)$, где $n \geq 0$, а $\tau_1, \tau_2, \ldots, \tau_n$ - термы; вхождения переменных в термы τ называются ***аргументами этого оператора;***

3. ***оператор присваивания*** - слово вида

x := τ, где *x* – переменная (*результат оператора*),

а τ - терм;

вхождения переменных в термы называются ***аргументами*** этого *оператора*;

4. ***условный оператор*** **(тест)** - логическое выражение; вхождения переменных в логическое выражение называются *аргументами* этого *оператора*;

5. ***оператор петли*** - односимвольное слово **loop**.

Способы представления ССП

Линейная форма стандартной схемы

Для использования линейной формы СПП множество специальных символов расширим дополнительными символами

```
0:   start(x) goto 1,
1:   y:=a goto 2,
2:   if p(x) then 5 else 3,
3:   y:=g(x,y) goto 4,
4:   x:=h(x) goto 2,
5:   stop(y).
```

Способы представления ССП

Графовая форма

Стандартной схемой в базисе *B* называется конечный (размеченный ориентированный) граф без свободных дуг и с вершинами следующих пяти видов:

1. *Начальная вершина* (ровно одна).
2. *Заключительная вершина* (может быть несколько).
3. *Вершина-преобразователь.*
4. *Вершина-распознаватель.*
5. *Вершина-петля.*

Конечное множество переменных схемы *S* составляют ее память X_S.

0: start(x) goto 1,

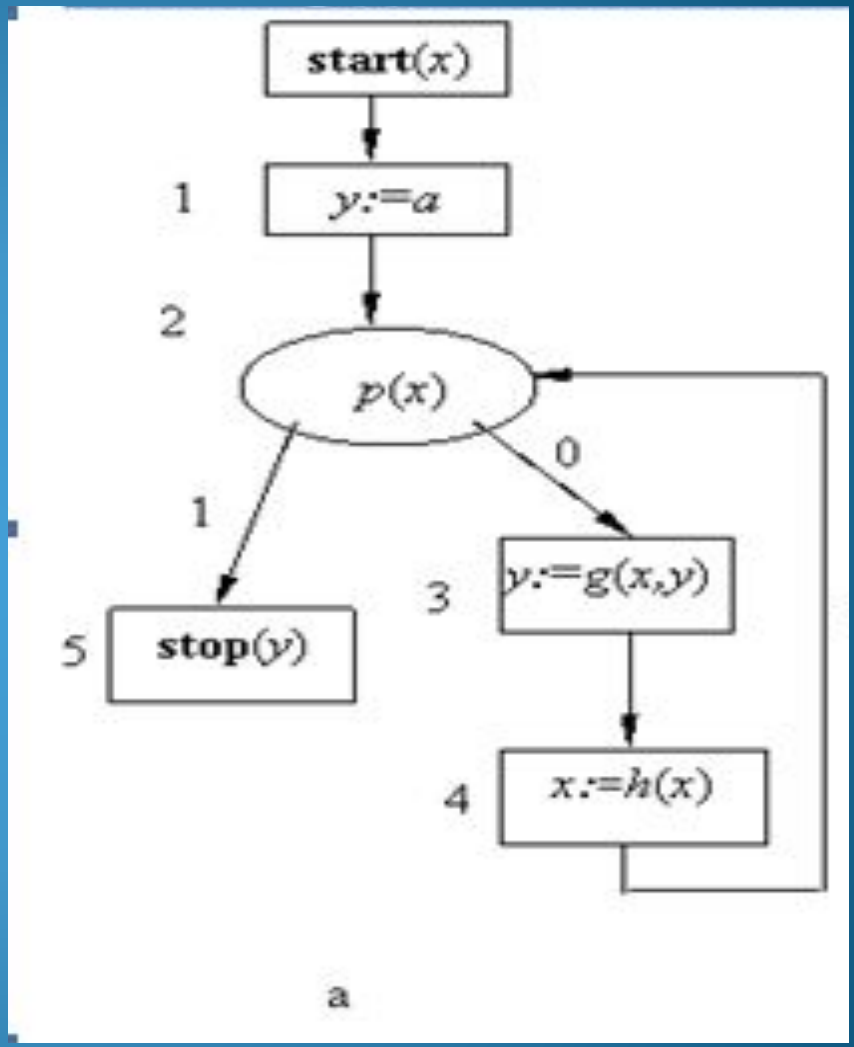
1: y:=a goto 2,

2: if p(x) then 5 else 3,

3: y:=g(x,y) goto 4,

4: x:=h(x) goto 2,

5: stop(y).



a

Способы представления ССП

Интерпретация стандартных схем программ

Состоянием памяти программы (S,I) называют функцию W: XS D, которая каждой переменной x из памяти схемы S сопоставляет элемент W(x) из области интерпретации D.

Значение терма τ при интерпретации I и состоянии памяти W (обозначим $\tau I(W)$) определяется следующим образом:

1) если $\tau = x$, x – переменная, то $\tau I(W) = W(x)$;

2) если $\tau = a$, a – константа, то $\tau I(W) = I(a)$;

3) если $\tau = f(n)(\tau_1, \tau_2..., \tau_n)$, то $\tau I(W) = I(f(n))(\tau_1 I(W), \tau_2 I(W),..., \tau_n I(W))$.

Аналогично определяется значение теста p при интерпретации I и состоянии памяти W или $pI(W)$:

если $p = p(n)(\tau_1, \tau_2,..., \tau_n)$, то $pI(W) = I(p(n))(\tau_1 I(W), \tau_2 I(W),... \tau_n I(W))$, $n \geq 0$.

Конфигурацией программы называют пару

U = (L,W), где L - метка вершины схемы S, а W - состояние ее памяти.

Выполнение программы описывается конечной или бесконечной последовательностей конфигураций, которую называют **протоколом выполнения программы (ПВП).**

Пример (вычисление n!):

Интерпретация (S1, I1) задана так:

1. область интерпретации D1 Nat - подмножество множества Nat целых неотрицательных чисел;

2. I1(x)=4; I1(y)=0; I1(a)=1;

3. I1(g)=G, где G - функция умножения чисел, т. е. G(d1,d2)= d1*d2;

4. I1(h)=H, где H - функция вычитания единицы, т. е. H(d)= d - 1;

5. I1(p)=P1, где P1 - предикат «равно 0», т.е. P1(d)=1, если d=0.

Вычислительные процессы

Дисциплина
«Теория вычислительных процессов»

модель

Модель *– это объект или описание объекта, системы для замещения одной системы (оригинала) другой системой для лучшего изучения оригинала или воспроизведения каких-либо его свойств.*

Любая модель строится и исследуется при определенных допущениях, гипотезах!!!

Задачи моделирования

- *Построение модели*
- *Исследование модели*
- *Использование модели*

***Моделирование** – универсальный метод получения описания функционирования объекта и использования знаний о нем.*

Виды моделей

1. ***Статическая*** (не учитывается временной параметр)

2. ***Динамическая*** (отображает систему во времени)

3. ***Дискретная*** (отображает поведение системы в дискретные моменты времени)

4. ***Непрерывная*** (описывает поведение системы для всех моментов времени некоторого промежутка)

Виды моделей

5. ***Имитационная*** (изучение возможных путей развития путем варьирования параметров модели)

6. ***Детерминированная*** (каждому входному набору параметров соответствует определенный набор выходных параметров)

7. ***Стохастическая*** (вероятностная)

Вычислительный Процесс

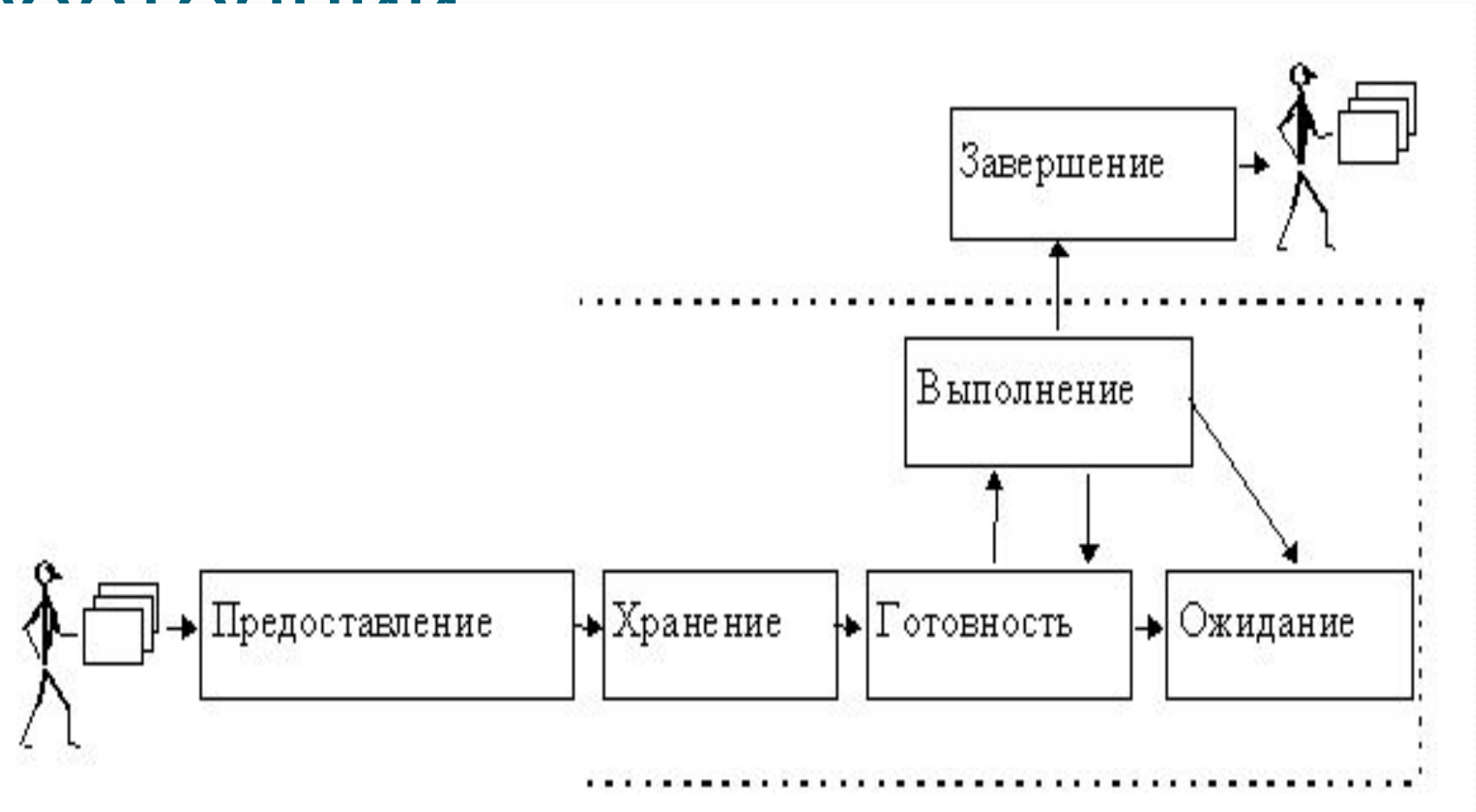
Под ***процессом*** понимается программа в стадии выполнения.

Процесс есть тройка (Q, f, g), где Q – множество состояний процесса; f – функция действия $f : Q \to Q$; $g \subseteq Q$ – начальное состояние процесса.

Свойства процесса

- Действия, реализуемые процессом - результат выполнения некоторой программы на реальном (виртуальном) процессоре
- Процесс не является закрытой системой и может взаимодействовать с другими процессами, воспринимая или изменяя часть среды, которую он с ними разделяет
- Каждый процесс живет временно
- В любой момент процесс может быть описан его состоянием. Все параметры (переменные), характеризующие текущее состояние процесса, объединяются в "вектор состояний»

Переход процессов из разных состояний



Параллельные и последовательные процессы

Параллельные процессы выполняют части задачи, так чтобы не были задействованы одновременно одни и те же ресурсы.

Последовательные процессы используют одни и те же ресурсы последовательно друг за другом.

Законы, управляющие поведением (P || Q), параллельно развивающихся процессов P и Q.

Первый закон выражает логическую симметрию между процессом и его окружением:

L1. P || Q = Q || P.

Следующий закон показывает, что при совместной работе трех процессов неважно, в каком порядке они объединены оператором параллельной композиции ||:

L2. P || (Q || R) = (P || Q) || R.

Процесс, находящийся в тупиковой ситуации, приводит к тупику всей системы.

L3. P || СТОПаР = СТОПаР.

Модели параллельных вычислений

Процесс/канал

Параллельные вычисления состоят из одного или более одновременно выполняющихся процессов, число которых может изменяться с течением времени выполнения программы

Обмен сообщениями

Модель не накладывает ограничений на динамическое создание процессов, на выполнение нескольких процессов одним процессором, на использование разных программ разными процессорами

Модели параллельных вычислений

Параллельность данных

Данные и операции централизованно распределяются между всеми процессорами. Одна и та же операция может быть применена к множеству элементов и структур данных

Модель общей памяти

Все процессы совместно используют общее адресное пространство . Для синхронизации используются механизмы блокировки

Трансляция. Компиляция и интерпретация

ДИСЦИПЛИНА «ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ»

Трансляция

Трансляция - процесс восприятия компьютером программы, написанной на некотором формальном языке.

При всем своем различии формальные языки имеют много общего и эквиваленты с точки зрения потенциальной возможности написать одну и ту же программу на любом из них.

Фазы трансляции и выполнения программы

Подготовка программы начинается **с редактирования файла**, содержащего текст этой программы, который имеет стандартное расширение для данного языка.

Затем выполняется его трансляция, которая включает в себя несколько фаз:

- препроцессор,
- лексический,
- синтаксический,
- семантический анализ,
- генерация кода
- оптимизация кода

В результате трансляции получается объектный модуль – некий «полуфабрикат» готовой программы, который потом участвует в ее сборке. Файл объектного модуля имеет стандартное расширение **obj**.

Фазы трансляции и выполнения программы

Компоновка (сборка) программы заключается в объединении одного или нескольких объектных модулей программы и объектных модулей, взятых из библиотечных файлов и содержащих стандартные функции и другие полезные вещи.

В результате получается исполняемая программа в виде отдельного файла (загрузочный модуль, программный файл) со стандартным расширением -".exe", который затем загружается в память и выполняется.

Препроцессор

Это предварительная фаза трансляции, которая выполняет обработку текста программы, не вдаваясь глубоко в ее содержание. Он производит замену одних частей текста на другие, при этом сама программа так и остается в исходном виде.

Препроцессор - предварительная фаза трансляции на уровне преобразования исходного текста программы.

Например, в языке Си директивы препроцессора оформлены отдельными строками программы, которые начинаются с символа "#"

```
#define      идентификатор  строка_текста
#define      SIZE      100
```

Аналогичные средства в других языках программирования носят название **макропроцессор, макросредства**.

Трансляция и ее фазы
1. Лексический анализ

Собственно трансляция начинается с лексического анализа программы.

Лексика языка программирования - это правила правописания слов» программы, таких как идентификаторы,
константы, служебные слова, комментарии.

Лексический анализ разбивает текст программы на указанные элементы. Особенность любой лексики –ее элементы представляют собой регулярные линейные последовательности символов.

Например, **Идентификатор** - это произвольная последовательность букв, цифр и символа "_", начинающаяся с буквы или "_".

Трансляция и ее фазы
2. синтаксис

Синтаксис языка программирования - это правила составления предложений языка из отдельных слов. Такими предложениями являются операции, операторы, определения функций и переменных.

Особенностью синтаксиса является принцип вложенности (рекурсивность) правил построения предложений. Это значит, что элемент синтаксиса языка в своем определении прямо или косвенно в одной из его частей содержит сам себя.

Например, в определении оператора цикла телом цикла является оператор, частным случаем которого является все тот же оператор цикла.

Трансляция и ее фазы
3. семантика

Семантика языка программирования - это смысл, который закладывается в каждую конструкцию языка.

Семантический анализ - это проверка смысловой правильности конструкции.

Например, если мы в выражении используем переменную, то она должна быть определена ранее по тексту программы, а из этого определения может быть получен ее тип. Исходя из типа переменной, можно говорит о допустимости операции с данной переменной.

Трансляция и ее фазы
4. генерация кода
5. оптимизация

Генерация кода - это преобразование элементарных действий, полученных в результате лексического, синтаксического и семантического анализа программы, в некоторое внутреннее представление.

Это могут быть коды команд, адреса и содержимое памяти данных, либо текст программы на языке Ассемблера, либо стандартизованный промежуточный код (например, Р-код). В процессе генерации кода производится и его оптимизация.

программирование, компоновка

Полученный в результате трансляции **объектный модуль** включает в себя:

готовые к выполнению коды команд

адреса и содержимое памяти данных

Но это касается только собственных внутренних объектов программы (функций и переменных). Обращение к внешним функциям и переменным, отсутствующим в данном фрагменте программы, не может быть полностью переведено во внутреннее представление и остается в объектном модуле в исходном (текстовом) виде.

Но если эти функции и переменные отсутствуют, значит, они должны быть каким-то образом получены в других объектных модулях.

Можно написать их на Си и оттранслировать.

Модульное программирование

Принцип **модульного программирования** - представление текста программы в виде нескольких файлов, каждый из которых транслируется отдельно.

С модульным программированием можно столкнуться в двух случаях:

- когда сами пишем модульную программу;
- когда используем стандартные библиотечные функции.

Библиотека объектных модулей

Библиотека объектных модулей - это файл (библиотечный файл), содержащий набор объектных модулей и собственный внутренний каталог.

Объектные модули библиотеки извлекаются из нее целиком при наличии в них требуемых внешних функций и переменных и используются в процессе компоновки программы.

Формальные языки и грамматики

- ***Формальный язык*** – множество всех слов в алфавите A

- ***Конкатенация слов***— двухместная операция над словами, заключающаяся в приписывании второго слова к первому. Результат конкатенации слов U и V обозначается UV.

Операции над формальными языками

- объединение
- пересечение
- дополнение (до множества всех слов в рассматриваемом алфавите)

Основные определения

Алфавит – конечное непустое множество символов (Σ).

Символ – любой знак, рассматриваемый как нечто неделимое - служебное слово языка программирования.

Примеры:

$\Sigma_1 = \{a,b,c\}$

$\Sigma_2 = \{0, 1\}$

Основные определения

Цепочка над алфавитом Σ– произвольная конечная последовательность символов из Σ.

Пример:

α=abbc

β=ab

Пустая цепочка – цепочка, не содержащая символов.(ε)

Основные определения

Σ^* - **бесконечное множество** всех цепочек над алфавитом Σ.

Язык над алфавитом Σ – произвольное множество цепочек, составленных из символов Σ.

Обозначается (L (Σ)

Примечание:

- Множество цепочек всегда бесконечно.
- Множество цепочек, образующих язык может быть конечным.
- Языки программирования содержат бесконечное множество цепочек.

Пример языка:

Язык L_2 $=\{a^n b^n c^n \mid n \geq 0\}$ – множество всех цепочек, содержащих вначале некоторое количество символов a, затем такое же количество символов b, затем столько же символов c.

aaabbbccc $\in L_2$

aaabbbcc $\notin L_2$

Порождающие грамматики

Порождающие грамматики являются механизмом, который позволяет задать обширный класс языков, содержащих бесконечное множество цепочек.

Порождающие грамматики используются при описании синтаксиса языков программирования.

Порождающие грамматики

Порождающей грамматикой называется четверка

G=(T, N, P, S), где

T – конечное множество терминальных символов – основной алфавит, порождаемого языка.

Элементы множества T – символы, из которых состоят цепочки языка, порождаемого данной грамматикой.

Терминальный символ («конечный»).

Терминалы обозначаются *a, b, c.*

Порождающие грамматики

N – конечное множеств нетерминальных (вспомогательных) символов – вспомогательный алфавит.

Нетерминалы – это понятия грамматики (языка), которые используются при его описании.

Обозначаются A, B, C.

Порождающие грамматики

P – конечное множество правил вывода, называемых продукциями.

Каждое правило имеет вид $\alpha \rightarrow \beta$,

где α и β – цепочки нетерминальных символов.

Цепочка α не может быть пустой, β – может быть пустой.

Правило $\alpha \rightarrow \beta$ определяет возможность подстановки β вместо α в процессе вывода(порождения) цепочек языка.

Порождающие грамматики

S (S принадлежит N) – начальный символ грамматики – одни из множества нетерминальных символов, начальный (стартовый) нетерминал.

Начальный нетерминал – правило соотвествующее правильному предложению языка.

Например, начальный нетерминал грамматики выражений обозначает «выражение».

Начальный нетерминал грамматики языка Паскаль – «программа».

Пример:

Грамматика

G: $S \rightarrow aSBc$ (1)

$S \rightarrow abc$ (2)

$cB \rightarrow Bc$ (3)

$bB \rightarrow bb$ (4)

$S \rightarrow \varepsilon$ (5)

Формальные методы описания перевода

Дисциплина

«Теория языков программирования и методы трансляции»

Синтаксический анализ - это процесс, который определяет, принадлежит ли некоторая последовательность лексем языку, порождаемому грамматикой.

Синтаксический анализ

Синтаксический разбор (распознавание) является первым этапом синтаксического анализа.

При его выполнении осуществляется подтверждение того, что входная цепочка символов является программой, а отдельные подцепочки составляют синтаксически правильные программные объекты.

Синтаксический анализ

Вслед за распознаванием отдельных подцепочек осуществляется анализ их семантической корректности на основе накопленной информации.

Затем проводится добавление новых объектов в объектную модель программы или в промежуточное представление.

Разбор предназначен для доказательства того, что анализируемая входная цепочка, записанная на входной ленте, принадлежит или не принадлежит множеству цепочек порождаемых грамматикой данного языка.

Синтаксический анализ

Выполнение синтаксического разбора осуществляется распознавателями, являющимися автоматами.

Цель доказательства в том, чтобы ответить на вопрос: ***«Принадлежит ли анализируемая цепочка множеству правильных цепочек заданного языка?»***

- Ответ «да» дается, если такая принадлежность установлена.
- В противном случае дается ответ «нет».

Единственный отказ на любом уровне ведет к общему отказу.

Классификация методов синтаксического разбора

Методы семантического анализа

Нисходящий разбор заключается в построении дерева разбора, начиная от корневой вершины.

Пример нисходящего разбора

Дана грамматика G

$\mathbf{G_8}$ = ({**S**}, {**a**, +, *}, **P**, **S**),

где **P** определяется как:

1. **S** (ρ) **a**
2. **S** (ρ) **S** + **S**
3. **S** (ρ) **S** * **S**

Пример нисходящего разбора слева направо

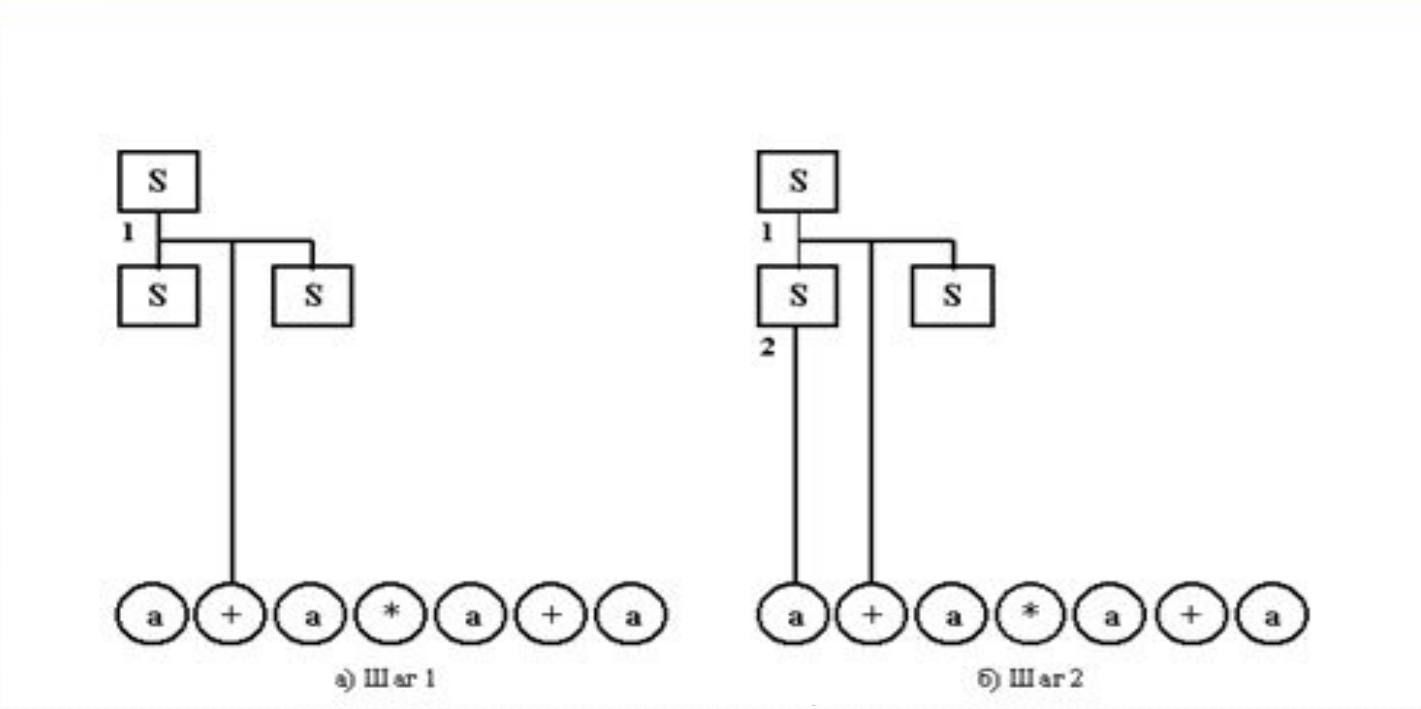
Например, выражение «**a+a*a+a**» можно получить следующими способами:

1. **S ⇒ S+S ⇒ a+S ⇒ a+S*S ⇒ a+ a*S ⇒ a+a*S+S ⇒ a+a*a+S ⇒ a+a*a+a**

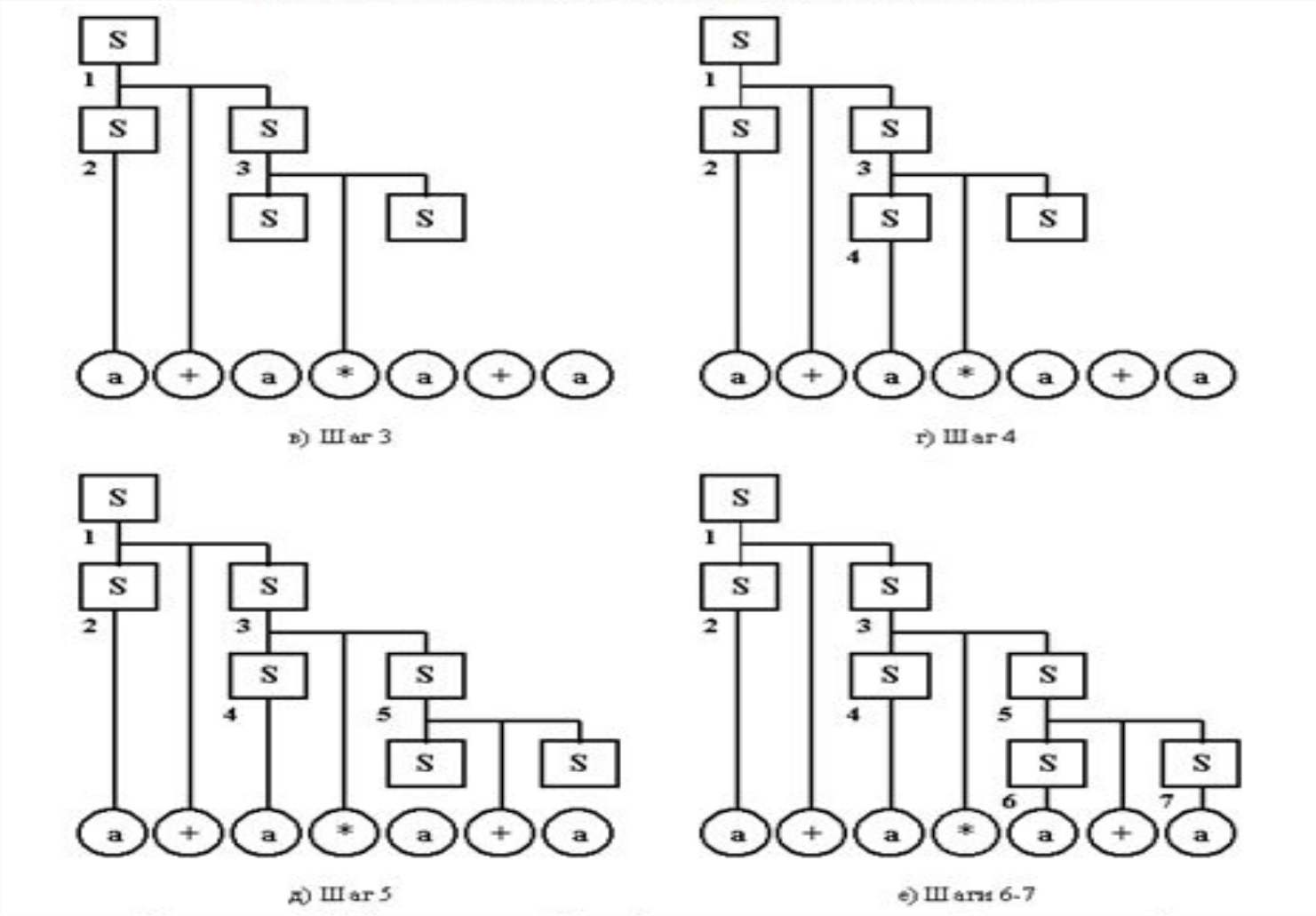
2. **S ⇒ S+S ⇒ S+a ⇒ S*S+a ⇒ S*a+a ⇒ S+S*a+a ⇒ S+a*a+a ⇒ a+a*a+a**

3. **S ⇒ S*S ⇒ S+S*S ⇒ S+S*S+S ⇒ a+ S*S+S ⇒a+a*S+S ⇒ a+a*S+a ⇒ a+a*a+a**

Пример нисходящего разбора слева направо



Пример нисходящего разбора слева направо



Восходящий разбор

При **восходящем разборе** дерево начинает строиться от терминальных листьев путем подстановки правил, применимых к входной цепочке, опять таки, в общем случае, в произвольном порядке.

Комбинированный разбор

Комбинированный разбор может быть реализован тогда, когда процесс распознавания разбивается на два этапа. На одном из них осуществляется нисходящий, а на другом – восходящий разбор.

Этапов может быть и больше, а порядок их применения – произвольным. Комбинированным можно считать разбор в любом трансляторе, если фазу лексического анализа принять за первый этап, а синтаксического – за второй.

эквивалентности МП-автоматов и КС-грамматик

Теорема

Язык является контекстно-свободным тогда и только тогда, когда он допускается МП-автоматом.

Преобразование КС-грамматик

Алгоритм 1 Устранение недостижимых символов.

Вход. КС-грамматика $G = (N, T, P, S)$.

Выход. КС-грамматика $G' = (N', T', P', S)$ без недостижимых символов, такая, что $L(G') = L(G)$.

Метод. Выполнить шаги 1-4:

(1) Положить $V_0 = \{S\}$ и $i = 1$,

(2) Положить $V_i = \{X \mid$ в P есть $A \to \alpha X \beta$ и $A \in V_{i-1}\} \cup V_{i-1}$,

(3) Если $V_i \neq V_{i-1}$, положить $i = i + 1$ и перейти к шагу 2, в противном случае перейти к шагу 4,

(4) Положить $N' = V_i \cap N, T' = V_i \cap T$. Включить в P' все правила из P, содержащие только символы из V_i.

Преобразование кс-грамматик

Алгоритм 2 Устранение несводимых символов

Вход. КС-грамматика G = (N, T, P, S).

Выход. КС-грамматика G' = (N', T', P', S) без несводимых символов, такая, что L(G') = L(G).

Метод. Выполнить шаги 1-4:

(1) Положить N' = T и i = 1,

(2) Положить $N_i = \{A | A \to \alpha \in P \text{ и } \alpha \in (N_{i-1})^*\} \cup N_{i-1}$,

(3) Если $N_i \neq N_{i-1}$, положить i = i + 1 и перейти к шагу 2, в противном случае положить $N_e = N_i$ и перейти к шагу 4,

(4) Положить $G_1 = ((N \cap N_e) \cup \{S\}, T, P_1, S)$, где P_1 состоит из правил множества P, содержащих только символы из $N_e \cup T$,

Устранение бесполезных символов

Чтобы устранить все бесполезные символы, необходимо применить к исходной грамматике сначала Алгоритм 2, а затем Алгоритм 1.

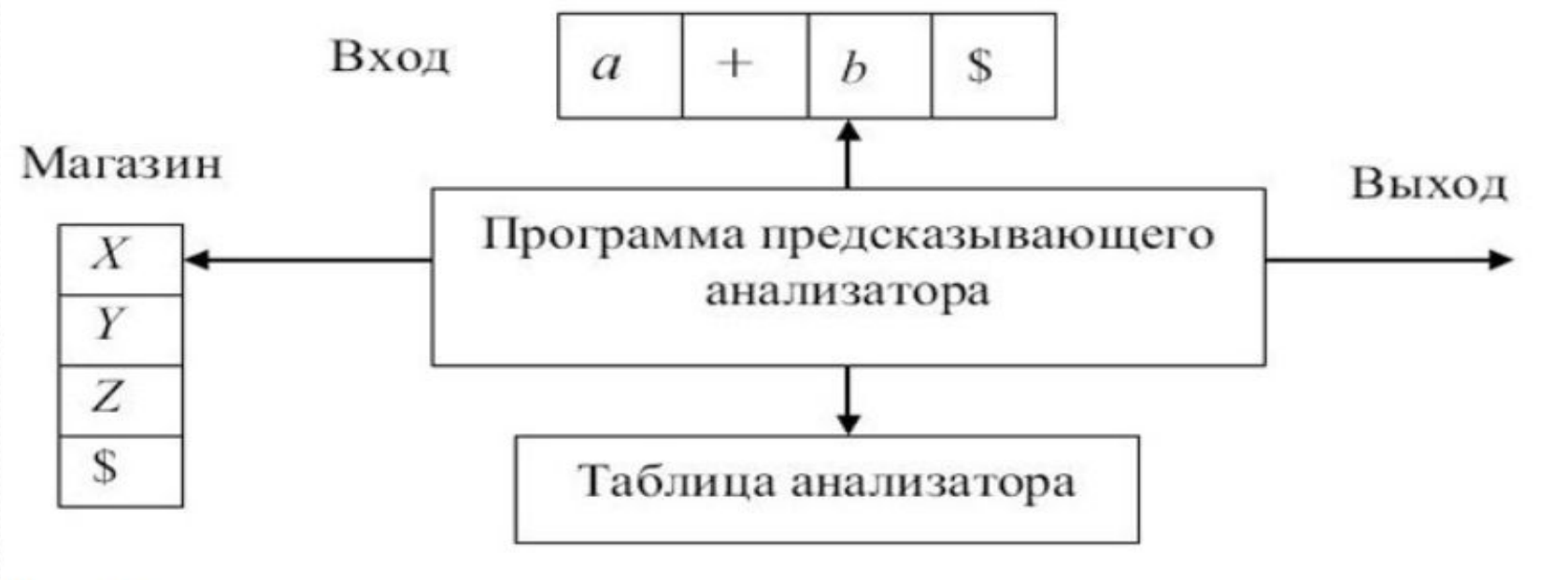
Разбор сверху-вниз (предсказывающий разбор)

Пусть дана КС-грамматика G = (N; T; P; S).

Главная задача предсказывающего разбора - определение правила вывода, которое нужно применить к нетерминалу.

Предсказывающий разбор

Предсказывающий анализатор имеет входную ленту, управляющее устройство (программу),таблицу анализа, магазин (стек) и выходную ленту. Входная лента содержит анализируемую строку, заканчивающуюся символом $-маркером конца строки. Выходная лента содержит последовательность примененных правил вывода.



Предсказывающий разбор

Таблица анализа - это двумерный массив M[A; a], где A- нетерминал, и a - терминал или символ $. Значением M[A; a]может быть некоторое правило грамматики или элемент"ошибка".

Магазин может содержать последовательность символов грамматики с $на дне. В начальный момент магазин содержит только начальный символ грамматики на верхушке и $на дне.

Предсказывающий разбор

Анализатор работает следующим образом:

Вначале анализатор находится в конфигурации, в которой магазин содержит S$, на входной ленте w$ (w- анализируемая цепочка), выходная лента пуста.

На каждом такте анализатор рассматривает X- символ на верхушке магазина и a-текущий входной символ.

Эти два символа определяют действия анализатора.

Предсказывающий разбор

Варианты действия анализатора:

1. Если X=a=$, анализатор останавливается, сообщает об успешном окончании разбора и выдает содержимое выходной ленты.
2. Если $X = a \neq$, анализатор удаляет X из магазина и продвигает указатель входа на следующий входной символ.
3. Если X - терминал, и $X \neq a$, то анализатор останавливается и сообщает о том, что входная цепочка не принадлежит языку.
4. Если X - нетерминал, анализатор заглядывает в таблицу M[X; a]. Возможны два случая:
   1. Значением M[X; a] является правило для X. В этом случае анализатор заменяет X на верхушке магазина на правую часть данного правила, а само правило помещает на выходную ленту. Указатель входа не передвигается.
   2. Значением M[X; a] является "ошибка". В этом случае анализатор останавливается и сообщает о том, что входная цепочка не принадлежит языку. Работа анализатора может быть задана следующей программой:

Синтаксически управляемый перевод

Фактически, такая схема представляет собой КС-грамматику, в которой к каждому правилу добавлен элемент перевода.

Всякий раз, когда правило участвует в выводе входной цепочки, с помощью элемента перевода вычисляется часть выходной цепочки, соответствующая части входной цепочки, порожденной этим правилом.

Схемы синтаксически управляемого перевода

Определение. Схемой синтаксически управляемого перевода (или трансляции, сокращенно: СУ-схемой) называется пятерка Tr = (N, T, П, R, S), где

- N - конечное множество нетерминальных символов;
- T - конечный входной алфавит;
- П - конечный выходной алфавит;
- R - конечное множество правил перевода вида

$$A \to u, v$$

Схема синтаксически управляемого перевода

- где u $(NT)^*$, v $(N)^*$ и вхождения нетерминалов в цепочку v образуют перестановку вхождений нетерминалов в цепочку u, так что каждому вхождению нетерминала B в цепочку u соответствует некоторое вхождение этого же нетерминала в цепочку v; если нетерминал B встречается более одного раза, для указания соответствия используются верхние целочисленные индексы;

- S - начальный символ, выделенный нетерминал из N.

Схема синтаксически управляемого перевода

Определим выводимую пару в схеме Tr следующим образом:

- (S, S) - выводимая пара, в которой символы S соответствуют друг другу;
- если (xAy, x'Ay') - выводимая пара, в цепочках которой вхождения A соответствуют друг другу, и A $\rightarrow$ u, v - правило из R, то (xuy, x'vy') - выводимая пара. Для обозначения такого вывода одной пары из другой будем пользоваться обозначением $\Rightarrow$: (xAy, x'Ay') $\Rightarrow$ (xuy, x'vy'). Рефлексивно-транзитивное замыкание отношение $\Rightarrow$ обозначим $\Rightarrow^*$.

Схема синтаксически управляемого перевода

Переводом $\tau(\mathrm{Tr})$, определяемым СУ-схемой Tr, назовем множество пар

$$\{(x, y) \mid (S, S) \Rightarrow^* (x, y),\ x \in T^*,\ y \in \Pi^*\}$$

Если через P обозначить множество входных правил вывода всех правил перевода, то G = (N, T, P, S) будет входной грамматикой для Tr.

СУ-схема Tr = (N, T, Π, R, S) называется простой, если для каждого правила A $\rightarrow$ u, v из R соответствующие друг другу вхождения нетерминалов встречаются в u и v в одном и том же порядке.

Перевод, определяемый простой СУ-схемой, называется простым синтаксически управляемым переводом (простым СУ-переводом).

Схема синтаксически управляемого перевода

- Класс переводов, определяемых магазинными преобразователями, совпадает с классом простых СУ-переводов.

- Существуют простые СУ-схемы, имеющие в качестве входных грамматик LR(1)-грамматики и не реализуемые ни на каком детерминированном преобразователе с магазинной памятью.

Обобщенная СУ-схема

Определение. Обобщенной схемой синтаксически управляемого перевода (или трансляции, сокращенно: ОСУ-схемой) называется шестерка $Tr = (N, T, \Pi, \Gamma, R, S)$, где все символы имеют тот же смысл, что и для СУ-схемы, за исключением того, что

- Γ - конечное множество символов перевода вида A_i, где $A \in N$ и i - целое число;
- R - конечное множество правил перевода вида

  $A \rightarrow u, A_1 = v_1, ..., A_m = v_m,$

  удовлетворяющих следующим условиям:

  - $A_j \in \Gamma$ для $1 \leqslant j \leqslant m$,
  - каждый символ, входящий в $v_1, ..., v_m$, либо принадлежит Π, либо является $B_k \in \Gamma$, где B входит в u,
  - если u имеет более одного вхождения символа B, то каждый символ B_k во всех v соотнесен (верхним индексом) с конкретным вхождением B.

$A \rightarrow u$ называют входным правилом вывода, A_i - переводом нетерминала A, $A_i = v_i$ - элементом перевода, связанным с этим правилом перевода. Если в ОСУ-схеме нет двух правил перевода с одинаковым входным правилом вывода, то ее называют семантически однозначной.

Транслирующие грамматики

Построение транслирующих грамматик предполагает применение подхода, который предусматривает использование одной грамматики и разрешает включение как входных, так и выходных
символов в каждое правило такой грамматики.

Транслирующие грамматики

Назначение:

Позволяют решать задачу перевода в более сложных случаях, чем СУ-схемы.

Транслирующие грамматики это разновидность КС-грамматик, где символы (терминалы) разделены на два множества, Σ_i и Σ_a (a от action), называемые „входными “ и „операционными“ соответственно.

При использовании ТГ, чтобы различать элементы Σ_i и Σ_a, чтобы различать последние, они заключаются в фигурные скобки, ‘{’, ‘}’, считая получившиеся на письме три символа одним символом алфавита.

Транслирующие грамматики

Определение.

Транслирующей грамматикой (Т - грамматикой)

называется КС-грамматика, множество терминальных

символов которой разбито на множество входных

символов и множество выходных символов, которые

называются также символами действия.

Пример(Т – грамматика):

$$\Gamma_{4.1}: V_{\text{твх}} = \{a,b,c\}, V_{\text{твых}} = \{x,y,z\}, V_a = \{I,A\}$$

$$R = \{\langle I\rangle - a\langle I\rangle x\langle A\rangle,$$

$$\langle I\rangle - z,$$

$$\langle A\rangle - \langle A\rangle\langle C\rangle,$$

$$\langle A\rangle - by$$

$$\}.$$

Выходные символы обозначим фигурными скобками.

С использованием таких обозначений правила грамматики $Г_{..}$ имеют вид:

$$
\begin{aligned}
R = \{ & \langle I \rangle - a \langle I \rangle \{x\} \langle A \rangle, \\
& \langle I \rangle - \{z\}, \\
& \langle A \rangle - \langle A \rangle c, \\
& \langle A \rangle - b\{y\} \\
\}.
\end{aligned}
$$

Вывод в транслирующих грамматиках выполняется по тем же правилам, что и в обычных КС - грамматиках.

Например, в рассматриваемой грамматике из начального символа может быть выведена следующая цепочка:

<I> ==> a<I>{x}<A> ==> a{z}{x}<A> ==> a{z}{x]b{y}

Каждый символ или цепочка символов, заключенные в фигурные скобки, должны рассматриваться как единый символ, называемый символом действия.

В общем случае
цепочки символов, заключенные в фигурные скобки, можно интерпретировать как имена процедур, выполнение которых производит требуемый эффект на выходе.

При описании перевода предусматривается, что каждый символ действия представляет собой процедуру, осуществляющую передачу символа, заключенного в фигурные скобки, на выход.

Когда нужно подчеркнуть, что используется такая интерпретация символов действия, то Т - грамматику называют **грамматикой цепочного перевода.**

Атрибутные грамматики

В Атрибутной грамматике с каждым символом грамматики может быть связан один или несколько *атрибутов*.

Для каждого синтаксического правила вводятся *семантические правила*, устанавливающие функциональные зависимости между атрибутами.

Дерево, содержащее атрибуты, называется *аннотированным*. Атрибуты придают смысл синтаксической структуре, описываемой деревом, и потому аннотированное дерево называется также **семантическим деревом**.

Атрибут $a(X_o)$ синтезируемый, если одному из правил вывода $p: X_o \to X_1 \ldots X_{np}$ сопоставлено семантическое правило $a^{<o>} = fa^{<o>}(\ldots)$

Атрибут a(Xi) наследуемый, если одному из правил вывода $p: X_o \to X_1 \ldots X_i \ldots X_{np}$ сопоставлено семантическое правило $a = fa(\ldots),\ i[1,np]$

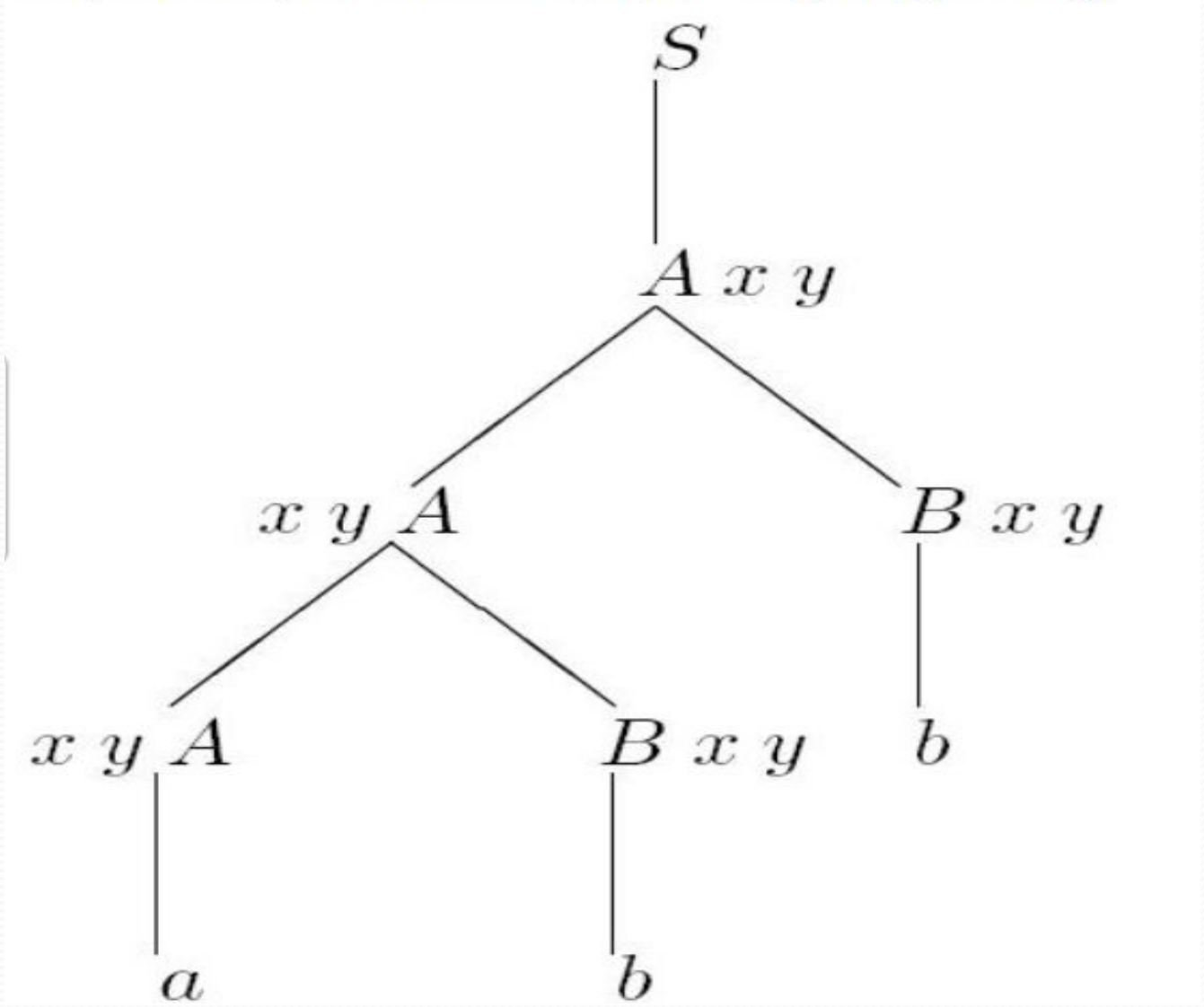
Множество синтезируемых атрибутов символа X - **S(X)**

Множество наследуемых атрибутов символа X - **I(X)**

Значение атрибутов терминальных символов-константы

(т.е. их значения определены, но для них нет семантических правил, определяющих их значения)

Атрибутная грамматика



Атрибутные грамматики

Атрибутная грамматика называется ***незациклленной,*** если графы зависимостей деревьев всех цепочек, принадлежащих языку, определяемому грамматикой G, не содержат циклов,

Атрибутная грамматика называется ***зацикленной***, если существует хотя бы одна цепочка, принадлежащая языку, для дерева разбора которой граф D(t) содержит ориентированный цикл.

Атрибутные грамматики

Теорема 1

Задача определения того, является ли данная атрибутная грамматика зацикленной, имеет экспоненциальную временную сложность, то есть существует константа c > 0 такая, что любой алгоритм, проверяющий на зацикленность произвольную атрибутную грамматику размера n, должен работать более, чем 2cn/log nшагов на бесконечно большом числе грамматик

Алгоритм кнута (Проверка атрибутной грамматики на зацикленность)

begin
for $X \in N$ do $\Gamma_x := 0$ end;
for $T \in N$ do $\Gamma_x := \{A(X)\}$ end;
{$A(X)$ - граф со множеством вершин-множеством атрибутов символа X и пустым множеством дуг}
finish := false; cycle := false;
while (not finish) and (not cycle) do
if $(\exists\, p : X_0 \to X_1 \dots X_{np})$ & $(\exists\, G_i \in \Gamma_{x_i}, i \in [0, n_p])$
такие, что $D_p[G_1 \dots Gn_p]$ содержит цикл
then cycle := true
else if $(\exists\, p : X_0 \to X_1 \dots X_{np})$ & $(\exists\, G_i(\Gamma_{x_i}, i \in [0, n_p])$
такие, что $D_p[G_1 \dots Gn_p] \in \Gamma_{x0}$
then $\Gamma_{x0} := \Gamma_{x0}\ \{D_p[G_1 \dots Gn_p]\}$
else finish := true
end end
end end.

Атрибутные грамматики

Теорема 2

Атрибутная грамматикаAGнезациклена тогда и только тогда, когда ни один из графов$D_p[G_1 \ldots Gn_p]$не содержит ориентированных циклов, то есть когда алгоритм B.1. заканчивается со значениемcycle = false.

Теорема 3

Алгоритм Кнута проверки на зацикленность атрибутной грамматики размераптребует в общем случаеexp(cn^2)шагов.

Верификация программ

Верификация - это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы.

Основная цель верификации состоит в подтверждении того, что программное обеспечение соответствует требованиям. Дополнительной целью является выявление и регистрация дефектов и ошибок, которые внесены во время разработки или модификации программы.
Верификация является неотъемлемой частью работ при коллективной разработке программных систем.

Правила верификации программ

Основа для исчисления выводов программ - правила К.Хоара (правила верификации) для интерпретации программных конструкций. Правила (аксиомы) К.Хоара определяют предусловия, как достаточные предусловия, гарантирующие, что исполнение соответствующего оператора при успешном завершении приведет к желаемым постусловиям.

- A1. Аксиома присваивания: { Ro } x := e { R }
Неформальное объяснение аксиомы: так как x после выполнения будет содержать значение e, то R будет истинно после выполнения, если результат подстановки e вместо x в R истинен перед выполнением.
Таким образом Ro = R(x) при x = e.
Для Ro вводится обозначение: Ro=Rxe (у Вирта)
или Rx->e (у Дейкстры)
что означает, что x заменяется на e.
Аксиома присваивания будет иметь вид:
{ Rxe} x:=e {R}

- Сформулируем два очевидных правила.
  A2. Если известно: { Q } S { P } и { P } => { R }, то { Q } S { R }
  A3. Если известно: { Q } S { P } и { R } => { Q }, то { R } S { P }
  Пусть S - это последовательность из двух операторов S1;S2 (составной оператор).
  A4. Если известно:
  { Q } S1 { P1 } и { P1 } S2 { R }, то { Q } S { R }.
  Очевидно, что это правило можно сформулировать для последовательности, состоящей из n операторов.
  Сформулируем правило для условного оператора (краткая форма).
  A5. Если известно:
  { Q and B } S1 { R } и
  { Q not B } => { R },то
  { Q } if B then S1 { R }.
  Правило A5 соответствует интерпретации условного оператора в языке программирования.

Сформулируем правило для альтернативного оператора (полная форма условного оператора).

A6. Если известно:

{ Q and B } S1 { R } и

{ Q not B } S2 { R },то

{ Q } if B then S1 else S2 { R }.

Сформулируем правила для операторов цикла. Предусловия и постусловия цикла until (до) удовлетворяют правилу:

A7. Если известно:

{ Q and not B } S1 { Q } , то

{ Q } repeat S1 until B { Q and not B }

Правило вводит важное понятие инварианта цикла. Предикат Q, истинный перед выполнением и после выполнения каждого шага цикла, называется инвариантным отношением или просто инвариантом цикла. В математике термин "инвариантный" означает не изменяющийся под воздействием совокупности рассматриваемых математических операций. В данном случае единственная операция - это выполнение шага цикла при условии истинности Q вначале.

Предусловия и постусловия цикла while (пока) удовлетворяют правилу:

A8. Если известно: { Q and B } S1 { Q } ,
то { Q } while B do S1 { Q and not B }

Правила A1 - A8 можно использовать для проверки согласованности передачи данных от оператора к оператору, для анализа структурных свойств текстов программ, для установления условий окончания цикла. Кроме того, правила можно использовать для анализа результатов выполнения программы, что связано с семантикой программы.

Абстрактная интерпретация

Абстрактный синтаксис программ является утончением синтаксиса данных, а именно - выделением подкласса **вычислимых выражений** (форм), т.е. данных, имеющих смысл как выражения языка и приспособленных к вычислению. Внешне это выглядит как объявление объектов, заранее известных в языке, и представление разных форм, вычисление которых обладает определенной спецификой.

Операционная семантика языка определяется как **интерпретация абстрактного синтаксиса**, представляющего выражения, имеющие значение.

интерпретатор

- **Интерпретатор** (interpreter) — программа, выполняющая интерпретацию, а также вид транслятора, осуществляющего пооперационную (покомандную) обработку и выполнение исходной программы или запроса.
- В отличие от компилятора, который осуществляет трансляцию всей программы высокого уровня в машинные коды один раз без ее выполнения (создает объектную программу), интерпретатор транслирует исходную программу команда за командой каждый раз при выполнении и не создает объектного модуля. За счет такого режима выполнение программы происходит медленнее, чем в случае ее обработки транслятором, однако при обработке интерпретатором программы выполняются сразу, без промежуточной стадии трансляции.

Абстрактный интерпретатор

Абстрактный интерпретатор – это сложный интерпретатор, компилирующего типа, перед выполнением производящий компиляцию исходного кода программы в машинный или «промежуточный код».

Они быстрее выполняют большие и циклические программы, не занимаются анализом исходного кода в реальном времени.