Автоматы и формальные языки

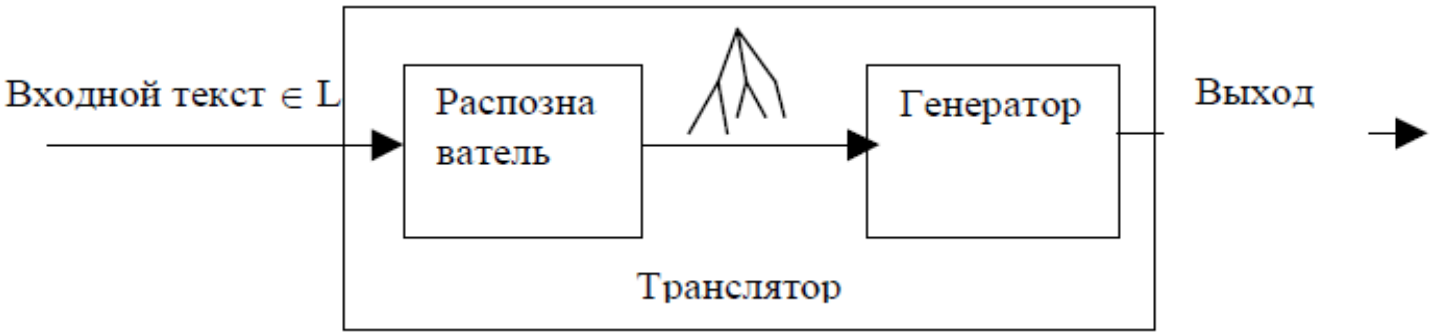
Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского Политехнического университета

karpov@dcn.infos.ru

Структура курса

- Конечные автоматы-распознаватели – 4 л
- Порождающие грамматики Хомского – 3 л
  - Лекция 5. Синтаксически-ориентированная трансляция и грамматики Хомского
  - Лекция 6. Иерархия грамматик Хомского
  - Лекция 7. Абстрактные распознающие автоматы
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Распознаватели КС-языков и трансляция – 6 л
- Дополнительные лекции - 2 л

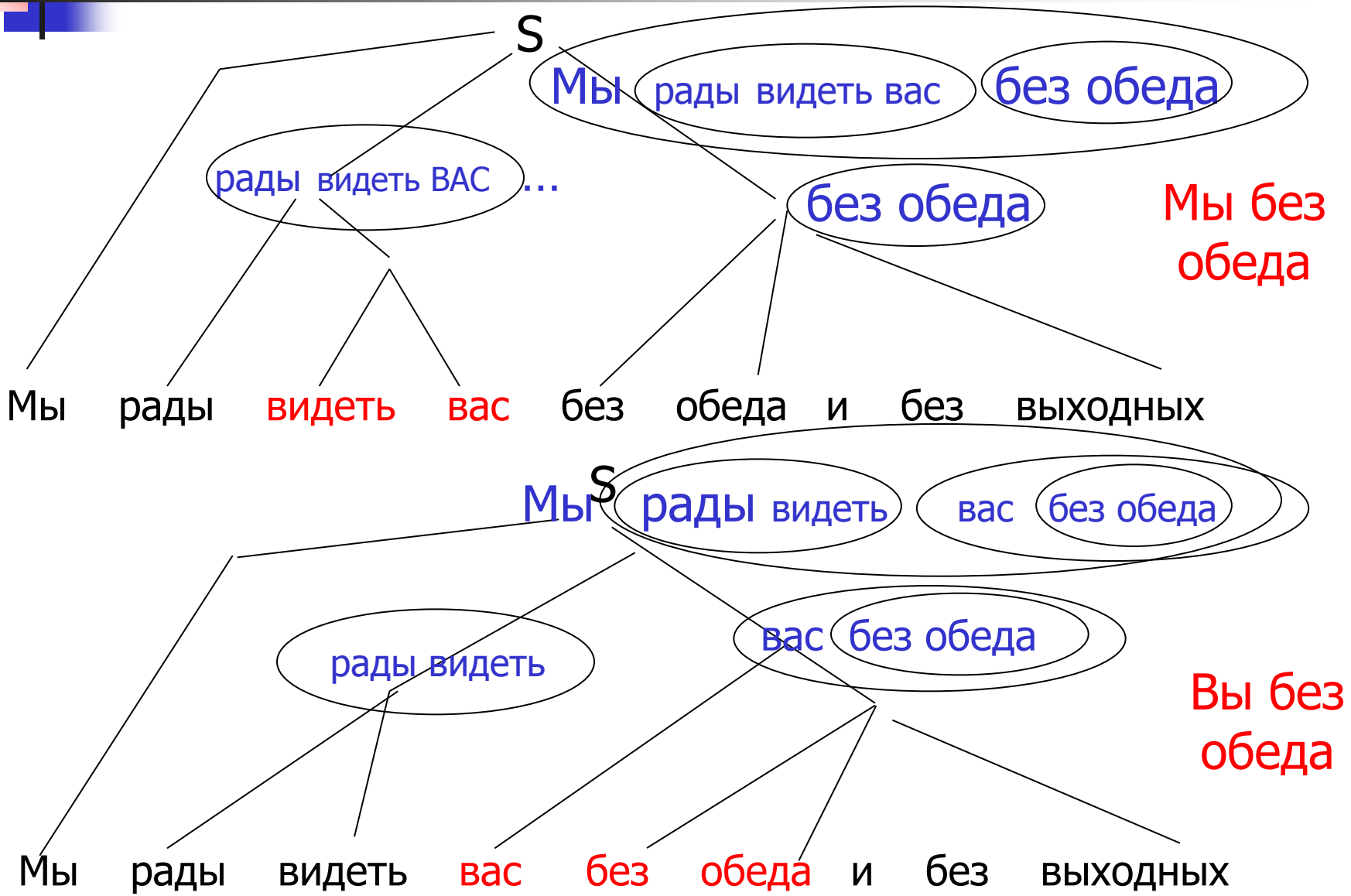
Грамматики Хомского и проблемы трансляции



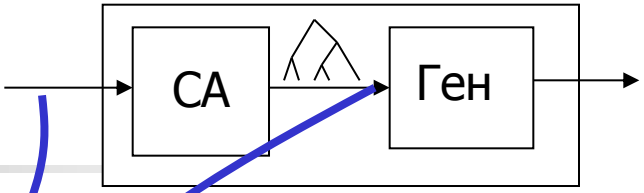
- Основной этап при построении транслятора – распознавание структуры входного текста, т.е. восстановления его вывода из начального символа грамматики – либо дерева вывода, либо последовательности шагов вывода

В соответствии с идеей Хомского о механизме понимания фраз естественного языка, структура входной цепочки должна помочь в трансляции программ, т.е. в построении выхода, отражающего “смысл” входной цепочки любого формального языка

Реклама магазина: смысл можно приписать при известной структуре

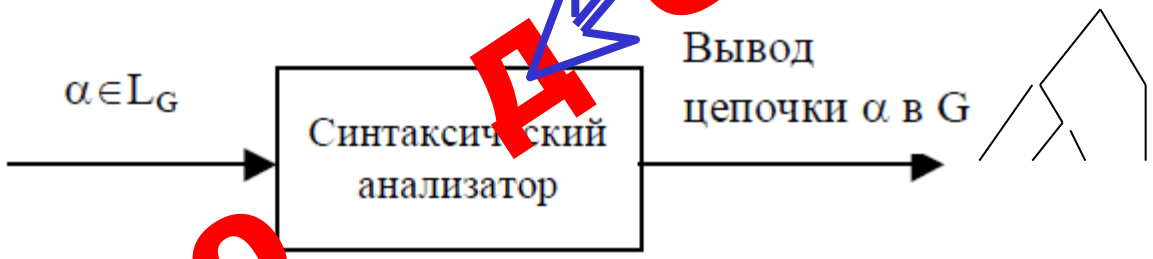


Синтаксический анализ



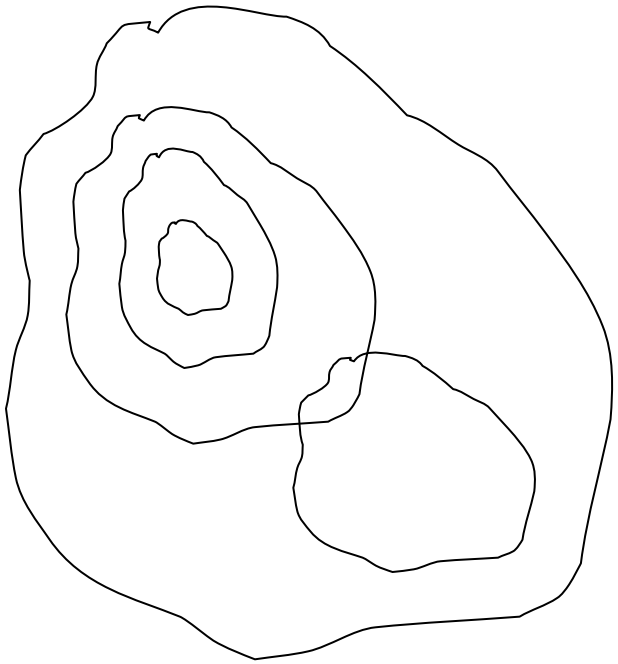
- Распознавание структуры цепочки языка происходит при восстановлении последовательности шагов вывода этой цепочки из начального символа грамматики. Такое восстановление называется синтаксическим анализом (парсингом, от английского parse-разбор)
- Существует ли общий алгоритм синтаксического анализа, подходящий для любых грамматик Хомского, и если существует, то какова его сложность?

Что делать?

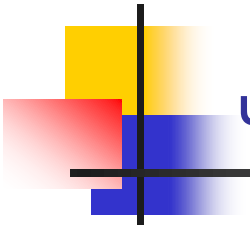


- Утверждение. Не существует общего алгоритма, позволяющего для произвольной цепочки над входным словарем восстановить вывод этой цепочки в произвольной грамматике Хомского с правилами вида $\alpha \to \beta$

Подклассы проблем



- Если для решения любой проблемы некоторого класса не существует общего алгоритма решения, или такой алгоритм очень сложен, то нужно выделить подклассы проблемы, для которых эффективные алгоритмы существуют. Иногда можно выделить некую иерархию классов проблем по их спецификации (по какому-нибудь параметру или свойству)



Что делать?

- Всегда сужение постановки задачи приводит к более содержательным моделям, в которых многие проблемы разрешимы. Иными словами, если мы будем упрощать модель, то сможем получить интересные результаты
- Другой вопрос, а будут ли эти суженные модели практически интересны (применимы?)
- Очень часто многие практические проблемы попадают именно в эти частные подклассы легко разрешимых проблем. Но, конечно, не всегда

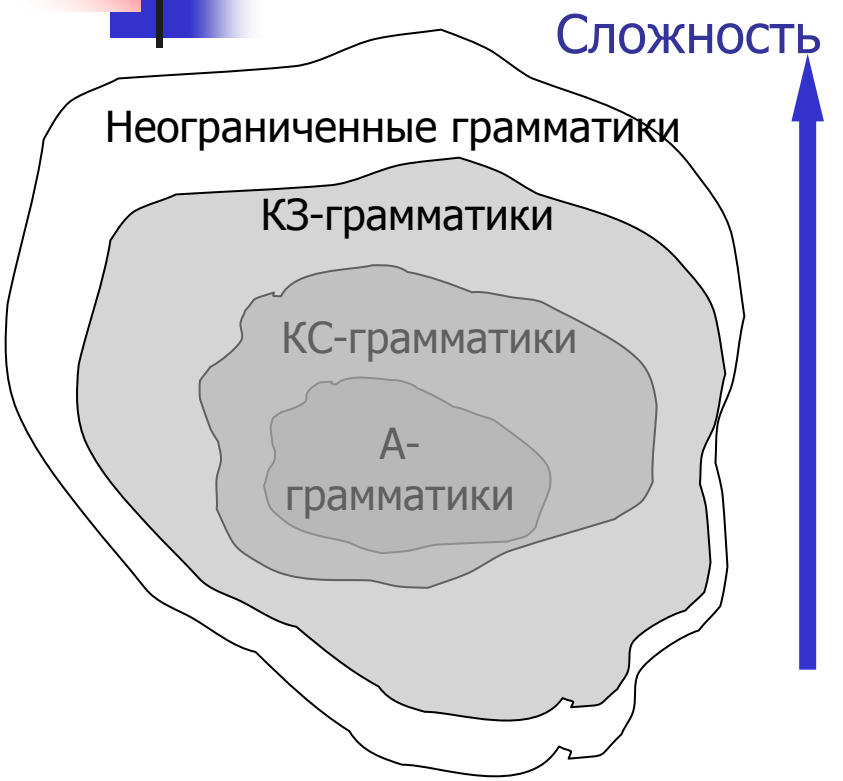
Подклассы грамматик иерархии Хомского

Тип	Вид правил	Название
0	$\alpha \to \beta$	Неограниченные грамматики
1	$\alpha A \beta \to \alpha \gamma \beta$	Контекстно-зависимые грамматики (КЗ-грамматики)
2	$A \to \gamma$	Контекстно-свободные грамматики (КС-граммаатики)
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики (А- грамматики)

Вложения грамматик синтаксические (по виду правил):

любая автоматная грамматика является контекстно-свободной,
любая КС-грамматика является контекстно-зависимой,
любая КЗ-грамматика является общей, неограниченной

Сложность распознавания цепочек, порождаемых грамматиками Хомского



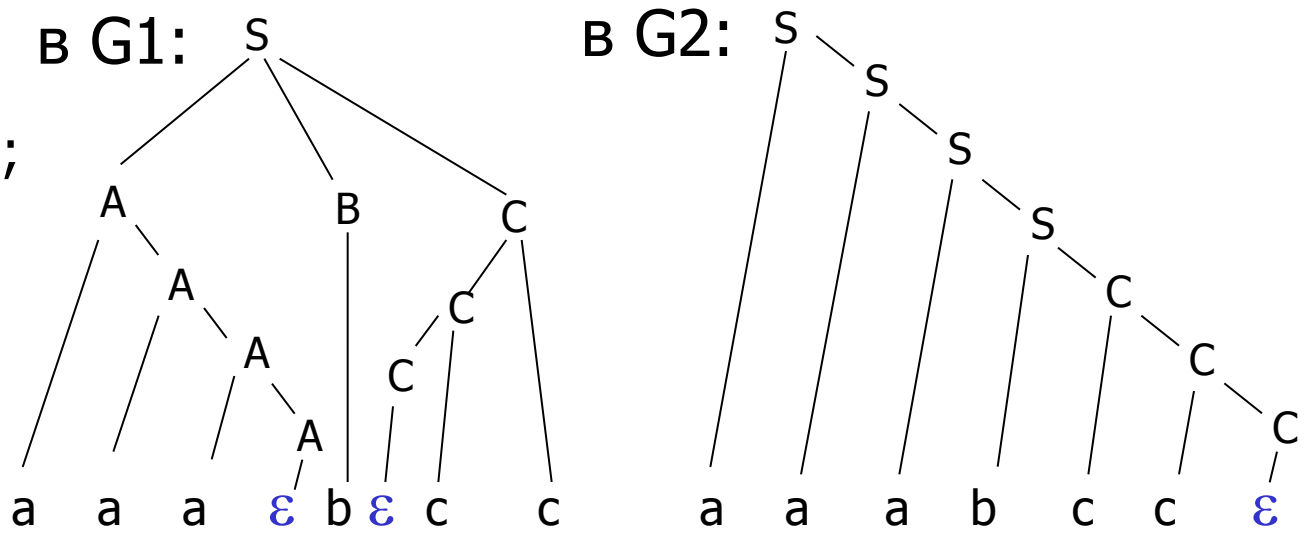
Тип	Вид правил	Название
0	$\alpha \rightarrow \beta$	Неограниченные грамматики
1	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Контекстно-зависимые грамматики (КЗ-грамматики)
2	$A \rightarrow \gamma$	Контекстно-свободные грамматики (КС-граммаатики)
3	$A \rightarrow aB$ $A \rightarrow a$ $A \rightarrow \varepsilon$	Автоматные грамматики (А- грамматики)

Сложность распознавания:

не существует алгоритма, распознающего любую цепочку в грамматике типа 0
распознавание цепочек для грамматики типа 1 очень сложно
сложность распознавания цепочек для грамматик типа 2 полиномиальна
распознавание цепочек для грамматик типа 3 очень простое (линейное)

Один язык – много грамматик

Σ={a, b, c},
L={$a^n b c^m$| n, m≥0};
α=aaabcc



G1:: S → ABC
A → aA
A → ε
B → b
C → Cc
C → ε
КС-грамматика

G2:: S → a S
S → b C
C → c C
C → ε
А-грамматика

Две грамматики называются эквивалентными, если они порождают один и тот же язык

Любой язык может быть порожден бесконечным числом грамматик.

Грамматика G1 является контекстно-свободной, и для любой цепочки языка L можно построить дерево вывода ее в этой грамматике. Эквивалентная ей грамматика G2 является автоматной, с более простым алгоритмом распознавания.

Можно ли любой язык породить простой грамматикой? Например, автоматной?

Иерархия Хомского

- Язык L называется языком типа k, если для L существует порождающая грамматика типа k И не существует никакой грамматики типа k-1, порождающей L.
- Может ли любой язык быть порожденным очень простой, например, автоматной грамматикой? НЕТ!!
  - язык $L=\{a^n b^n \mid n\geq 0\}$ является КС-языком, но не может быть порожден никакой автоматной грамматикой
  - язык $L=\{a^n b^n c^n \mid n>0\}$ является контекстно-зависимым, и не существует никакой КС-грамматики, его порождающей, и т.д.
- Отсюда следует справедливость строгого вложения как классов грамматик, так и классов языков
- Пусть $\mathfrak{I}_N$ – класс грамматик типа N, Λ_N – класс языков, порождаемых грамматиками типа N

Теорема. Строгое вложение существует как для классов грамматик Хомского:

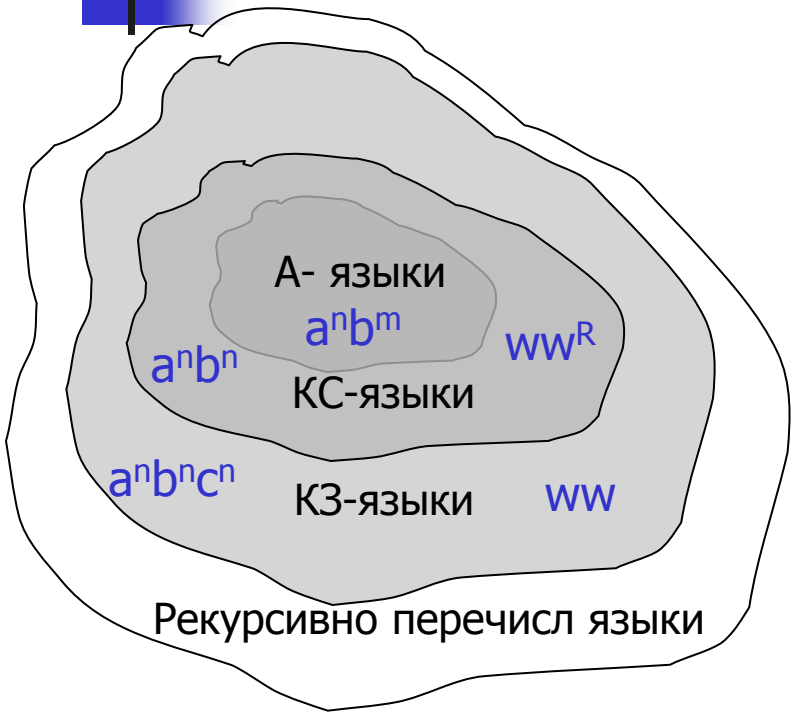
$$\mathfrak{I}_3 \subset \mathfrak{I}_2 \subset \mathfrak{I}_1 \subset \mathfrak{I}_0$$

так и для классов формальных языков:

$$\Lambda_3 \subset \Lambda_2 \subset \Lambda_1 \subset \Lambda_0$$

Эта иерархия языков и грамматик называется иерархией Хомского

Иерархия формальных языков



$\alpha \to \beta$	Рекурсивно перечислимые языки
$\alpha A \beta \to \alpha \gamma \beta$	Контекстно-зависимые языки (КЗ-языки)
$A \to \gamma$	Контекстно-свободные языки (КС-язык)
$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные языки (А- языки)

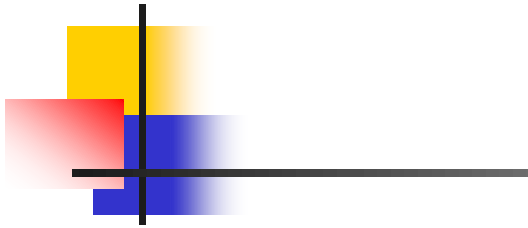
Вложения языков (по типам порождающих грамматик):

$\{a^n b^m \mid n,m \geq 0\}$ - *Регулярные, автоматные языки порождаются А-грамматиками*

$\{a^n b^n \mid n \geq 0\}$, $\{ww^R \mid w \in T^*\}$ - *КС-языки, они не порождаются А-грамматиками*

$\{a^n b^n c^n \mid n \geq 0\}$, $\{ww \mid w \in T^*\}$ - *КЗ-языки, они не порождаются КС-грамматиками,*

Существуют языки, порождаемые грамматикой типа 0, и не порождаемые никакой грамматикой типа 1



Автоматные грамматики и языки

Подклассы грамматик иерархии Хомского

Тип	Вид правил	Название
0	$\alpha \to \beta$	Неограниченные грамматики
1	$\alpha A \beta \to \alpha \gamma \beta$	Контекстно-зависимые грамматики (КЗ-грамматики)
2	$A \to \gamma$	Контекстно-свободные грамматики (КС-граммаатики)
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики (А- грамматики)

Грамматики типа 3

aabbce

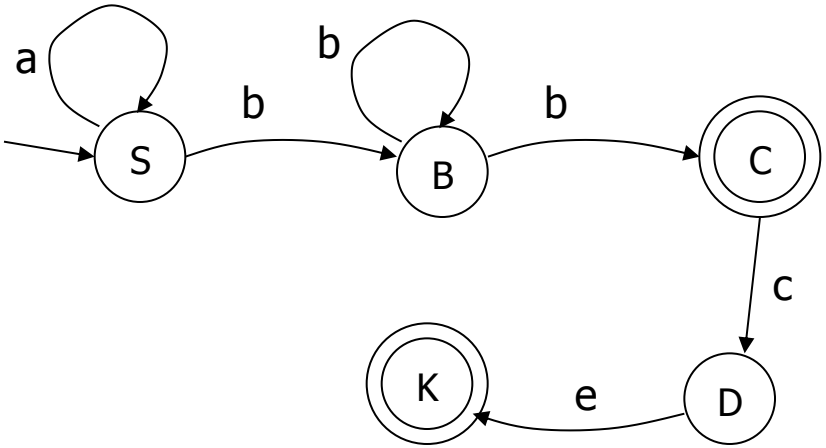
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbC \Rightarrow aabbcD \Rightarrow aabbce$

Пример:

$S \to aS$
$S \to bB$
$B \to bB$
$B \to bC$
$C \to cD$
$C \to \varepsilon$
$D \to e$

Грамматика типа 3 Хомского -> КА

Граф автоматной грамматики:

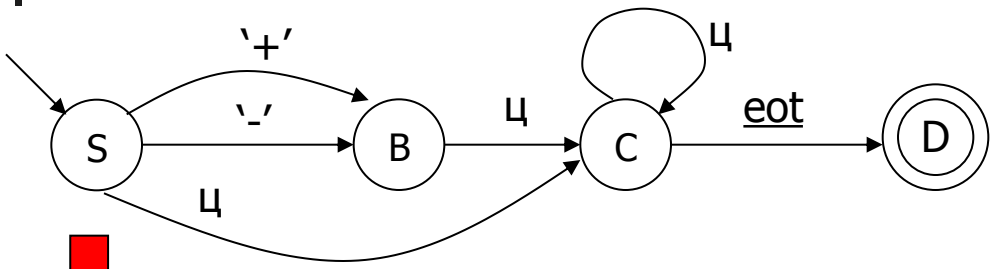


Все конечные языки являются автоматными!

Каждой автоматной грамматике соответствует распознающий конечный автомат (в общем случае недетерминированный)

Автоматные языки называют иногда регулярными языками

КА -> грамматика Хомского



- 9336*eot*

G:: S → '+' B
S → '-' B
S → ц C
B → ц C
C → ц C
C → *eot* D
D → ε

Порождающая грамматика Хомского для автоматных языков строится тривиально. Пусть дан автомат:

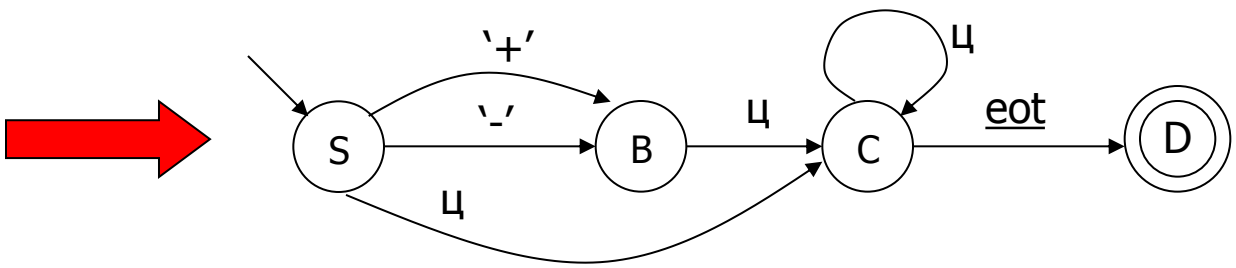
1. В качестве нетерминалов грамматики - состояния
2. Начальное состояние является начальным нетерминалом
3. Переходу A -a->B в автомате соответствует правило A→aB в грамматике
4. Финальному состоянию D соответствует правило D → ε

S ⇒ - B ⇒ - цC ⇒ - ццC ⇒ - цццC ⇒ - ццццC ⇒ - цццц*eot* D ⇒ - цццц*eot*

A-грамматика -> конечный автомат-распознаватель

- Обратное тоже верно: для любой автоматной грамматики, порождающей язык L, существует конечный автомат, допускающий L, который может быть построен непосредственно по продукциям грамматики. Фактически, соответствующий конечный автомат – это граф автоматной грамматики
- Построенный по грамматике Хомского конечный автомат осуществляет распознавание цепочек языка, и в этой роли используется как основа трансляторов автоматных языков.

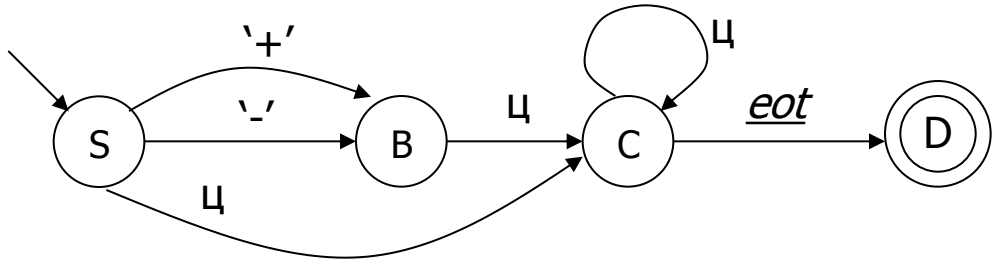
$S \rightarrow$ '+' B
$S \rightarrow$ '-' B
$S \rightarrow$ ц C
$B \rightarrow$ ц C
$C \rightarrow$ ц C
$C \rightarrow$ *eot* D
$D \rightarrow \varepsilon$

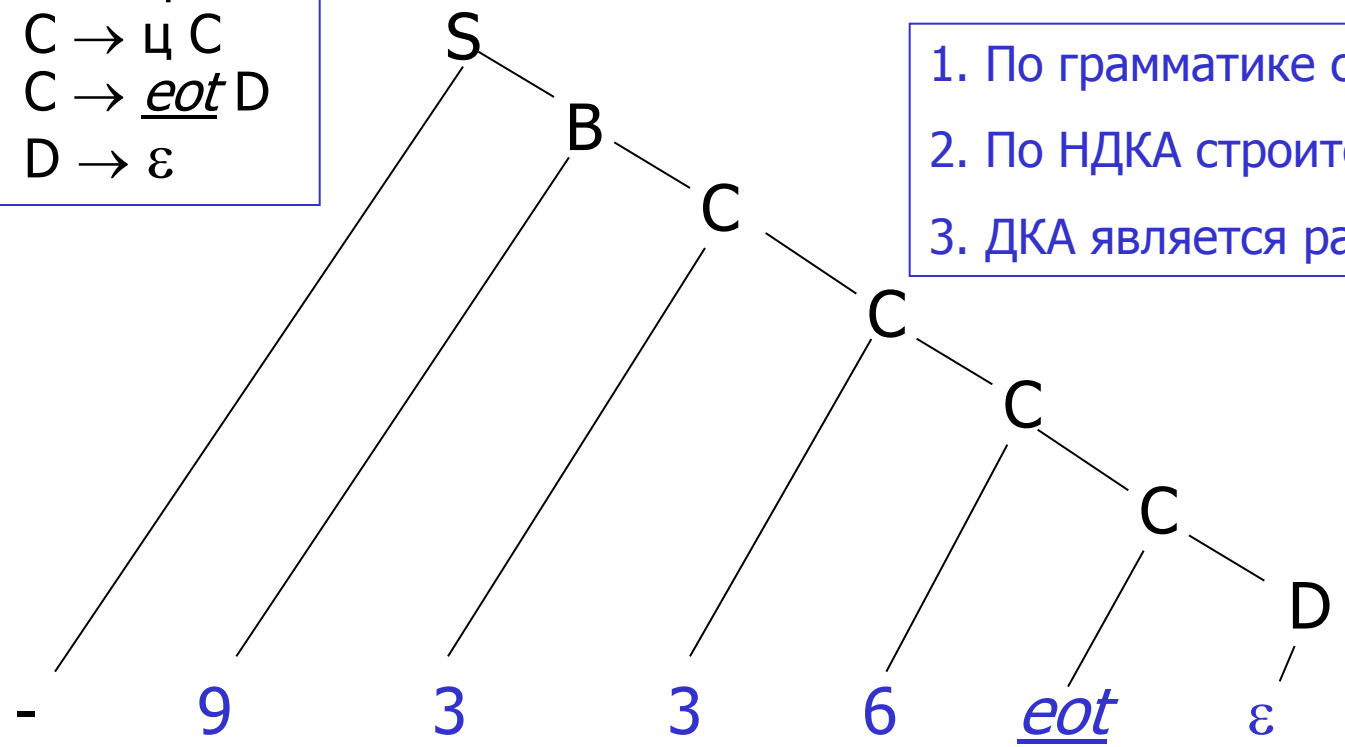


Синтаксический анализ языков типа 3

- 9336*eot*

G::
- S → ‘+’ B
- S → ‘-’ B
- S → ц C
- B → ц C
- C → ц C
- C → *eot* D
- D → ε





1. По грамматике строится НДКА
2. По НДКА строится ДКА
3. ДКА является распознавателем

S ⇒ - B ⇒ - ц C ⇒ - ццC ⇒ - цццC ⇒ - ццццC ⇒ - ццццц *eot* D ⇒ - ццццц *eot*

Пример: Регулярный язык L, определяемый ρ = 11*0*

- Автоматная грамматика, порождающая язык L:

G::

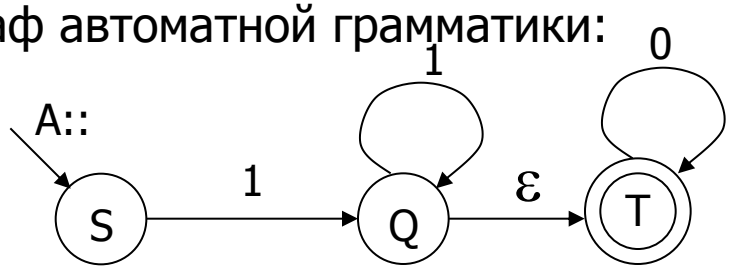
$S \to 1Q$
$Q \to 1Q$
$Q \to T$
$T \to 0T$
$T \to \varepsilon$

Эквивалентная грамматика:

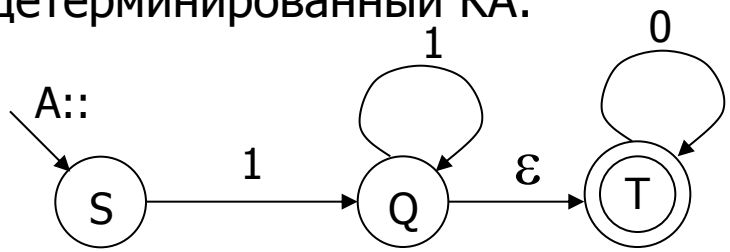
G1::

$S \to 1Q$
$Q \to 1Q$
$Q \to 0T$
$Q \to \varepsilon$
$T \to 0T$
$T \to \varepsilon$

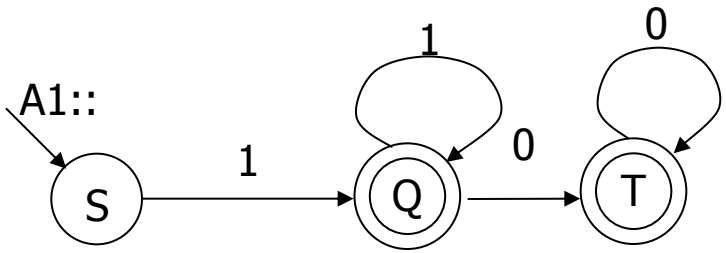
Граф автоматной грамматики:

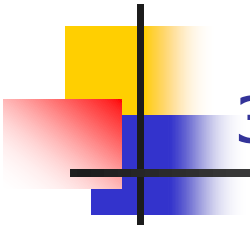


Недетерминированный КА:



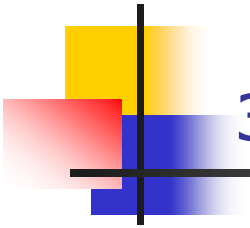
Детерминированный КА:





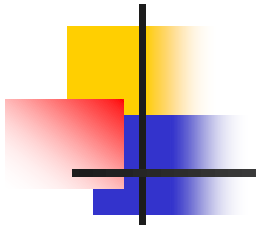
Заключение: грамматики Хомского типа 3

- В порождающих грамматиках Хомского типа 3 на правую часть продукций накладываются жесткие ограничения. Эти ограничения приводят к тому, что порождаемые языки этого типа являются автоматными, и распознающее их автоматическое устройство – это конечный автомат
- Все результаты, относящиеся к конечным автоматам, справедливы для грамматик типа 3
- Отсюда следует, что для языков типа 3 существует очень эффективный (линейный) алгоритм синтаксического анализа, основанный на модели конечного автомата.
- Языки типа 3 имеют сравнительно узкое применение в трансляции языков программирования из-за их очень ограниченных возможностей. Такими языками, например, нельзя описать даже скобочные структуры



Заключение: свойства автоматных грамматик

- Большинство свойств грамматик типа 3 разрешимы. Это означает, что можно получить ответы на такие вопросы:
  - порождает ли данная грамматика любые цепочки?
  - порождает ли грамматика пустой язык?
  - порождается ли этой грамматикой заданная цепочка символов языка?
  - ограничено ли количество цепочек в языке?
- Грамматики типа 3 используются в сканерах (лексических анализаторах) компиляторов для распознавания лексем (идентификаторов, литералов и строк, а также ключевых слов данного языка,например, if, begin, while)



Контекстно-свободные грамматики и языки

Грамматики типа 2 – КС-грамматики

Тип	Вид правил	Название
0	$\alpha \to \beta$	Неограниченные грамматики
1	$\alpha A \beta \to \alpha \gamma \beta$	Контекстно-зависимые грамматики (КЗ-грамматики)
2	$A \to \gamma$	Контекстно-свободные грамматики (КС-грамматики)
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики (А - грамматики)

- По сравнению с грамматиками типа 3, правила замены здесь более сложны: нетерминал А может быть заменен произвольной цепочкой α в пустом контексте (частный случай контекста), поэтому грамматика и называется контекстно-свободной (или бесконтекстной)
- По английски- Context Free Grammar, CFG
- Нетерминал, стоящий в левой части правила, иногда называют LHS – “Left Hand Side” символом правила

Грамматики типа 2: правила вида A→β

- A – нетерминал, он обозначает некоторую конструкцию
- β - произвольная цепочка из терминалов и нетерминалов

- Правила КС-грамматики определяют, как конструкция может быть построена из подконструкций и символов цепочек языка в любом контексте. Замена конструкции на подконструкции возможна всегда, в любом контексте по любой альтернативе для этого нетерминала
- Это естественный подход к заданию сложных структур. Например, дом:

<дом> → <этаж> |<дом><этаж>
<этаж> → <комната> |<этаж><комната>
<комната>→ дверь <стены> пол потолок
<стены> → стена стена стена стена

- Для цепочек КС-языков вывод их из начального символа удобно представляется деревом, называемым '*деревом синтаксического анализа*' (или '*деревом вывода*')

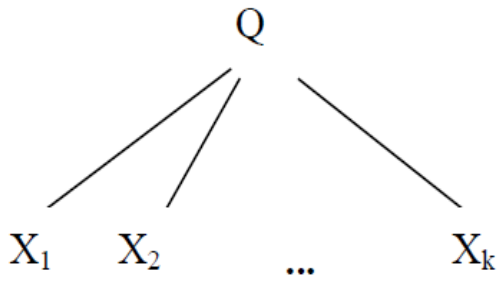
Одно из базовых свойств КС-грамматик – то, что они легко описывают вложенные структуры

Структура дерева вывода в КС-грамматиках

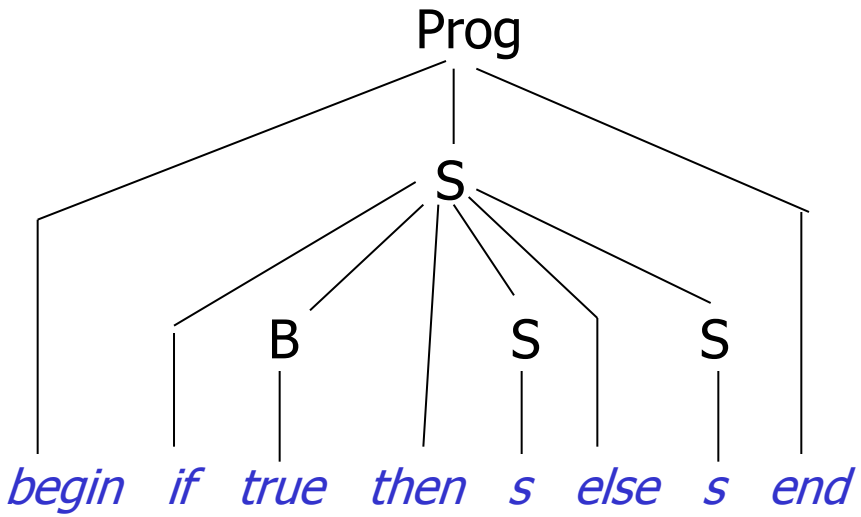
Деревья вывода цепочек языка, порождаемого грамматикой с правилами

$$Q \rightarrow X_1 X_2 \ldots X_к,$$

имеют следующую структуру. Корень дерева помечен начальным символом грамматики, в каждом внутреннем узле дерева стоит нетерминал, листья помечены терминальными символами так, что читаемые слева направо, они представляют терминальную цепочку, выведенную из начального символа. Из любого узла дерева, помеченного нетерминальным символом Q, идут вниз ветви к узлам, помеченным $X_1 X_2 \ldots X_к$, если $Q \rightarrow X_1 X_2 \ldots X_к$ – это одно из правил грамматики



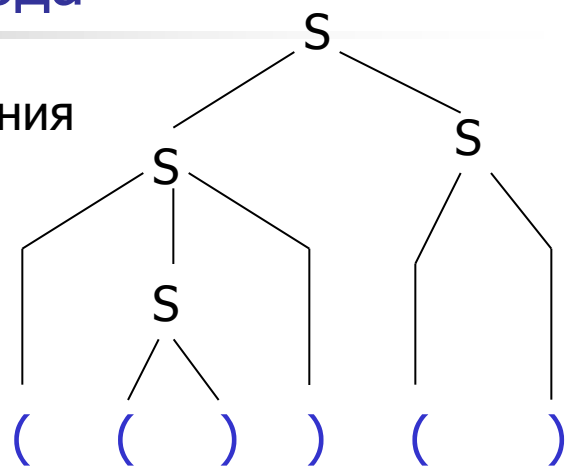
G:: Prog → *begin* S *end*
S → *if* B *then* S *else* S
S → *s*
B → *true*
B → *false*



Примеры КС-языков. Дерево вывода

- Σ={ '(' , ')' } L = правильные скобочные выражения

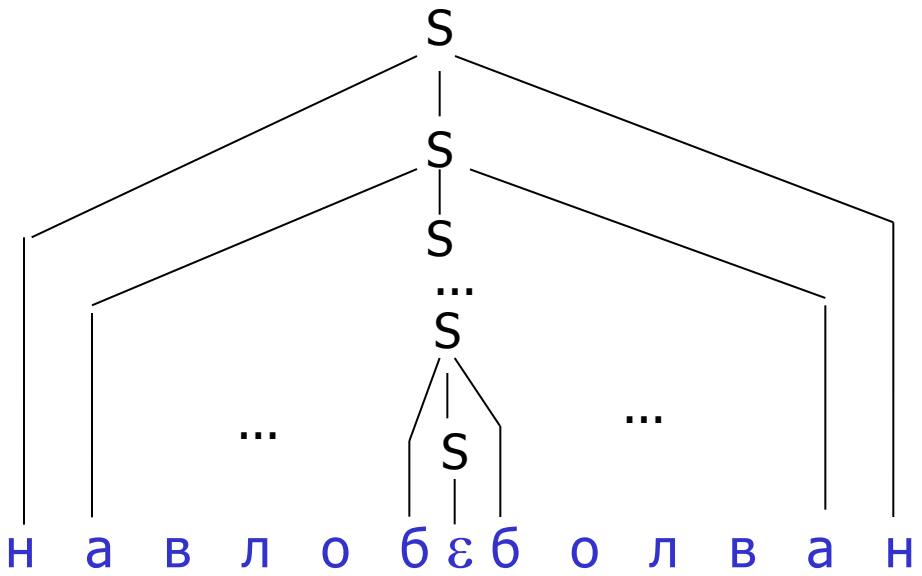
  (( ))( ) $\in$ L))(( ) $\notin$ L - язык неавтоматный!



G:: $S \to (S)$
$S \to S S$
$S \to ( )$

- Σ={ ... } Язык палиндромов $\{ww^R \mid w \in \Sigma^*\}$: навлобболван

G:: $S \to a S a$
$S \to б S б$
$S \to в S в$
...
$S \to \varepsilon$



Строим КС-грамматику языка Милан

- Как построить грамматику языка Милан (Mini Language)?
- Программа:

```
begin
    m:=read;
    n:=read;
    while m≠n do
      if m>n then m:=m-n else n:=n-m;
    write(m)
end
```

Будем считать, что лексемы уже выделены лексическим анализатором

Терминалы – это символы, из которых строятся цепочки языка

Σ={'+', '-', ... , '0',..., '9', ..., begin, end, if, ..., };

Что здесь является нетерминалами? Мы их выбираем сами при построении грамматики!

Что необходимо, когда мы говорим о языке, но в языке не встречается? Названия конструкций языка: <программа>, <оператор> и т.д.

Правила: <программа> → begin <тело программы> end

КС-грамматика языка Милан

- Σ={'+', '-', ... , '0',..., '9', ..., begin, end, if, ..., };
  L_{14} = язык MiLan

- ***begin***
  *m:=**read**;*
  *n:=**read**;*
  while** m≠n **do
  ***if** m>n **then** m:=m-n **else** n:=n-m;*
  ***write**(m)*
  end $\in L_{14}$

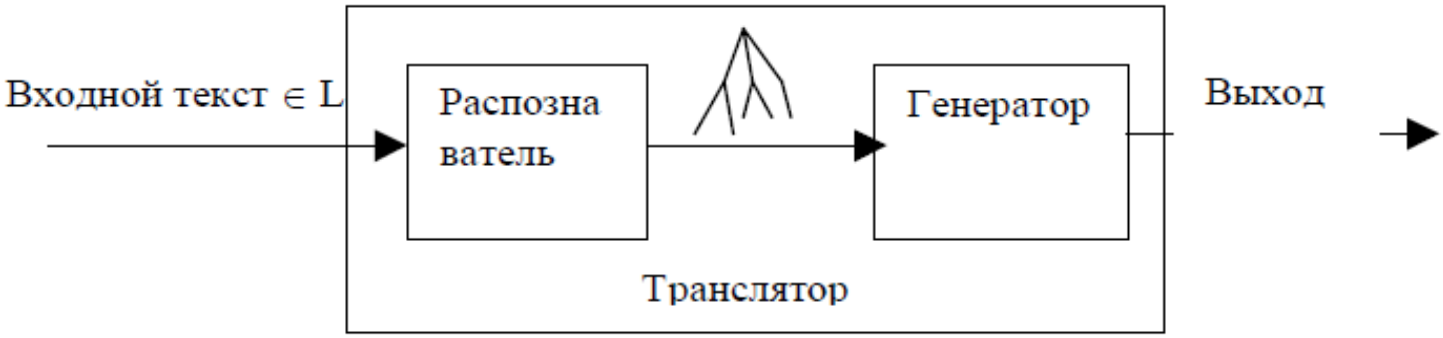
G^0: Prog → begin L end
L→ S | S ; L
S → *i* := E
| if B then L
| if B then L else L
| while B do L
| output (E)
B → E q E
E → (E) | EotsE | EotmE | *i* | *c* | read

Нетерминалы: названия конструкций
Prog - <программа>
L – <список операторов>
S – <оператор>
E – <ар выражение>
B – <условие>

Если имя нетерминала в грамматике G поменять, то это НИКАК не отразится на языке L(G)

Все терминалы – лексемы
begin – слово begin
end – слово end
...
i – имя переменной
q – символ отношения
c - константа
ots – опер типа сложения
otm – опер типа умножен

Грамматики Хомского и проблемы трансляции



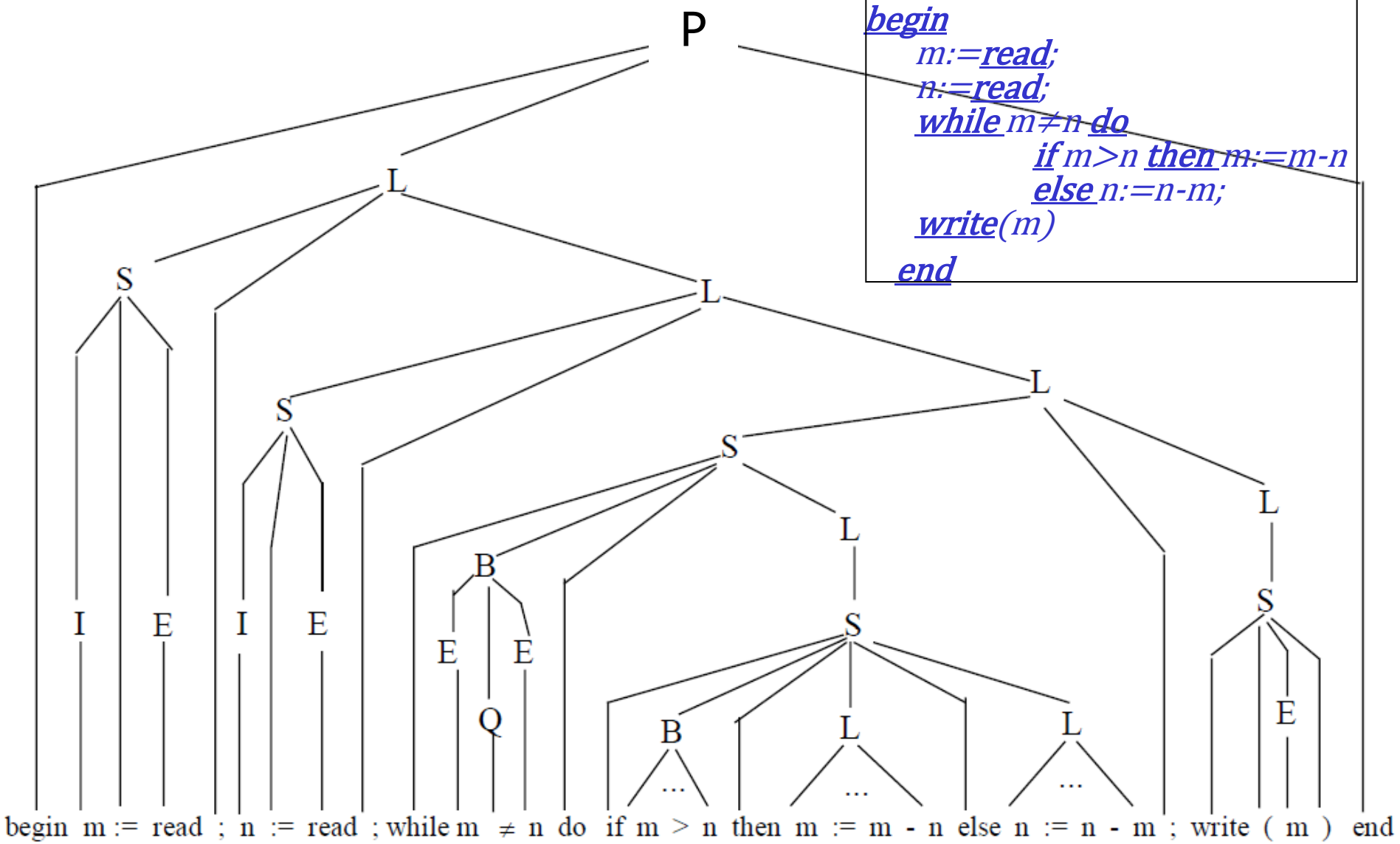
- Основной этап при построении транслятора – распознавание структуры входного текста, т.е. восстановления его вывода из начального символа грамматики – либо дерева вывода, либо последовательности шагов вывода

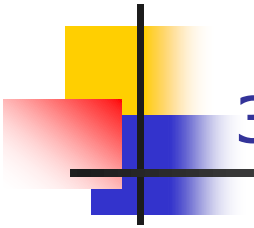
- Структура входной цепочки в КС-грамматике – это дерево вывода этой цепочки в данной грамматике
- Для любой порождающей КС-грамматики Хомского G существуют эффективные (полиномиальные) алгоритмы, позволяющие для произвольной цепочки над входным словарем восстановить дерево вывода этой цепочки в грамматике G

Эти алгоритмы называются алгоритмами синтаксического анализа

Дерево вывода программы на языке Милан

```
begin
    m:=read;
    n:=read;
    while m≠n do
            if m>n then m:=m-n
            else n:=n-m;
    write(m)
end
```



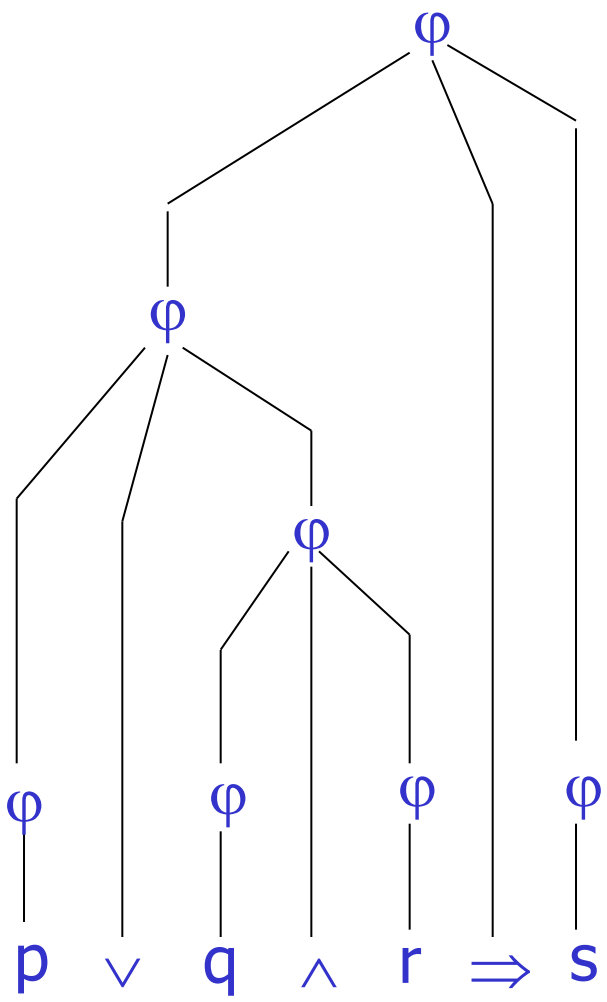


Замечание

- Построенная грамматика языка Милан обладает рядом недостатков, которые будут выявляться и устраняться впоследствии
- В частности, эта грамматика двусмысленная, и приведенная программа нахождения наибольшего общего делителя как раз содержит двусмысленность
- Проблема неоднозначности произвольной КС-грамматики алгоритмически неразрешима

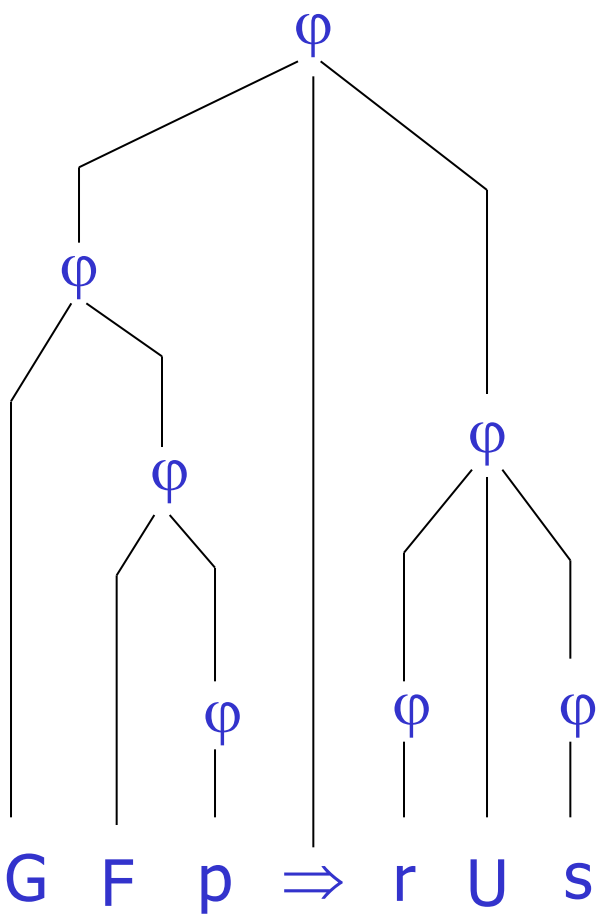
Применение КС-грамматик в математике (ППФ Логики)

- Формулы логики высказываний рекурсивно определяются так:
  - любое атомарное утверждение p, q, ... – формула;
  - если φ и ψ - формулы то $\neg\varphi$, и $\varphi\vee\psi$, $\varphi\wedge\psi$, $\varphi\Rightarrow\psi$ - тоже формулы
- КС-грамматика, задающая все правильно построенные формулы логики (φ - это конструкция ‘формула’):
  - $\varphi \to p \mid q \mid \ldots \mid \neg\varphi \mid \varphi\vee\varphi \mid \varphi\wedge\varphi \mid \varphi \Rightarrow\varphi$
- $\varphi\vee\varphi$ выглядит странно – но здесь φ - это не конкретная формула а нетерминал, имеющий смысл “любая формула”
- такая запись не очень понятна не знающим формальных грамматик, часто пишут:
  - $\varphi, \psi ::= p \mid q \mid \ldots \mid \neg\varphi \mid \varphi\vee\psi \mid \varphi\wedge\psi \mid \varphi\Rightarrow\psi$



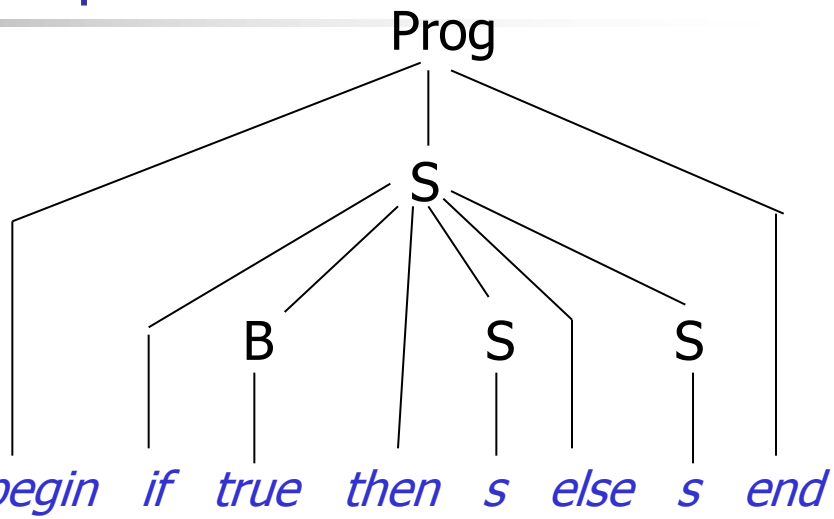
Применение КС-грамматик: ППФ логики LTL

- Формула линейной темпоральной логики (LTL) определяется так:
  - если p - атомарный предикат, то p – формула LTL;
  - если φ и ψ - формулы LTL то $\neg\varphi$, $\varphi\vee\psi$, $X\varphi$ и $\varphi U\psi$ - тоже формулы LTL
- КС-грамматика, задающая все LTL-формулы:
  - $\varphi \to p \mid \neg\varphi \mid \varphi\vee\varphi \mid \varphi \Rightarrow\varphi \mid X\varphi \mid \varphi U\varphi \mid G\varphi \mid F\varphi$
- для не знающих смысла формальных грамматик часто пишут:
  - $\varphi, \psi ::= p \mid \neg\varphi \mid \varphi\vee\psi \mid \varphi \Rightarrow\psi \mid X\varphi \mid \varphi U\psi \mid G\varphi \mid F\varphi$



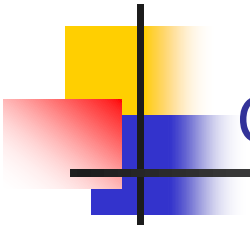
Канонические выводы в КС-грамматике

G:: Prog → begin S end
S → if B then S else S
S → s
B → true
B → false



Одно дерево – несколько выводов

- 1-й вывод (заменяется самый левый нетерминал) – канонический левый вывод:
  Prog ⇒ begin **S** end ⇒ begin if **B** then S else S end ⇒ begin if true then **S** else S end
  ⇒ begin if true then s else **S** end ⇒ begin if true then s else s end
  Левый вывод соответствует порождению дерева вывода "сверху вниз, слева направо".
- 2-й вывод (заменяется самый правый нетерминал) – канонический правый вывод:
  Prog ⇒ begin **S** end ⇒ begin if B then S else **S** end ⇒ begin if B then **S** else s end
  ⇒ begin if **B** then s else s end ⇒ begin if true then s else s end
  Правый вывод соответствует порождению дерева вывода "снизу вверх, справа налево".
- 3-й вывод (заменяется произвольный нетерминал) – не канонический вывод:
  Prog ⇒ begin **S** end ⇒ begin if **B** then S else S end ⇒ begin if true then S else **S** end
  ⇒ begin if true then **S** else s end ⇒ begin if true then s else s end



Синтаксическое дерево и канонические выводы

- Синтаксическому дереву (дереву вывода) цепочки в КС-грамматике соответствует точно один левый и точно один правый вывод этой цепочки
- Левый и правый выводы цепочки в КС-грамматике называются каноническими. Они представляют дерево вывода единственным образом

- Определение. КС-грамматика G называется неоднозначной (двусмысленной), если для некоторой порождаемой этой грамматикой цепочки имеется несколько деревьев вывода

- Для некоторой порождаемой цепочки в КС-грамматике G имеется несколько левых (правых) канонических выводов тогда и только тогда, когда эта цепочка имеет в G несколько деревьев вывода

Двусмысленные грамматики

L={α | в цепочке α количества вхождений a и b равны}, Σ={a, b}.

Если в цепочке первый символ – a, то во всей остальной части цепочки число символов b на единицу больше числа символов a.

Пусть B – нетерминал, из которого порождаются все такие цепочки

G:: S $\rightarrow$ aB | bA | ε
B $\rightarrow$ bS | aBB
A $\rightarrow$ aS | bAA

Левый (канонический) вывод цепочки α = aabbab:

S $\Rightarrow$ aB $\Rightarrow$ aaBB $\Rightarrow$ aabSB $\Rightarrow$ aabB $\Rightarrow$ aabbS $\Rightarrow$ aabbaB $\Rightarrow$ aabbabS $\Rightarrow$ aabbab.

Возможен и другой левый вывод той же самой цепочки:

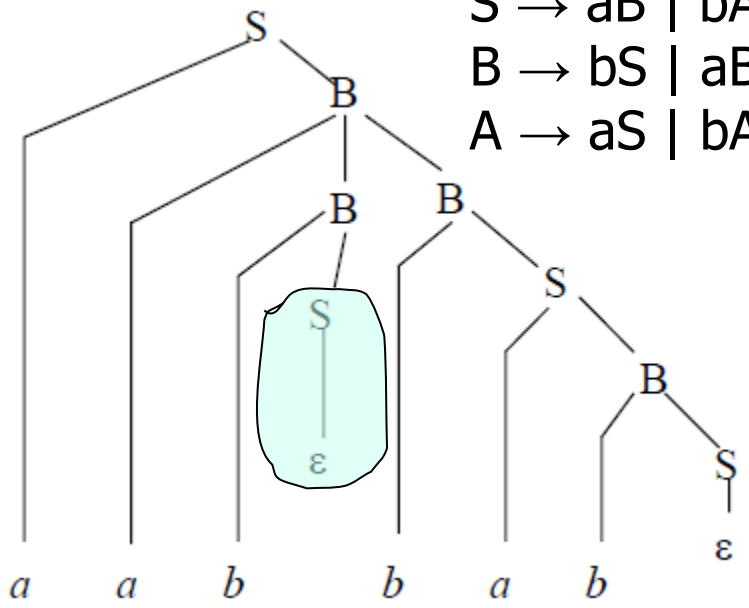
S $\Rightarrow$aB $\Rightarrow$aaBB $\Rightarrow$ aabSB $\Rightarrow$ aabbAB $\Rightarrow$ aabbaSB $\Rightarrow$ aabbaB $\Rightarrow$ aabbabS $\Rightarrow$ aabbab

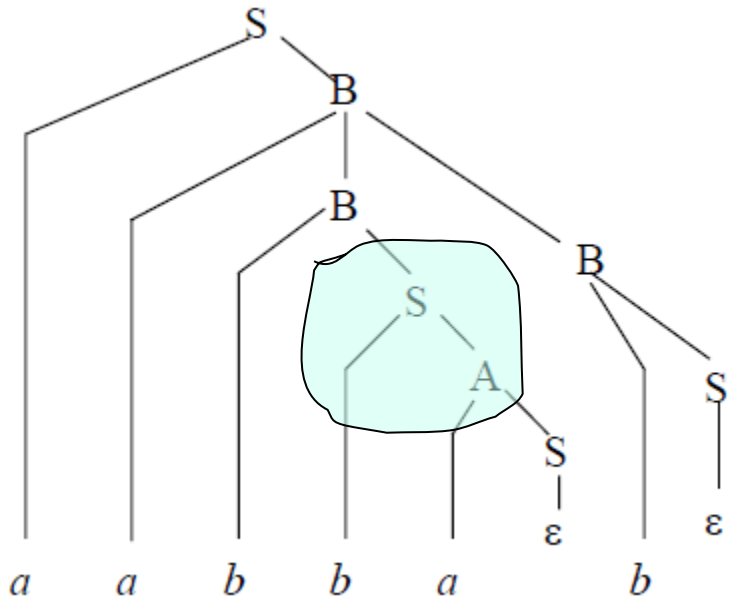
Следовательно, грамматика двусмысленная

Два дерева вывода цепочки aabbab

aabbab

$S \rightarrow aB \mid bA \mid \varepsilon$
$B \rightarrow bS \mid aBB$
$A \rightarrow aS \mid bAA$





Два различных синтаксических дерева, два различных левых вывода

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaB \Rightarrow aabbabS \Rightarrow aabbab$

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow aabbAB \Rightarrow aabbaSB \Rightarrow aabbaB \Rightarrow aabbabS \Rightarrow aabbab$

Пример двусмысленной грамматики

Σ ={ (,) }; L=множество правильных скобочных выражений.

Грамматика порождает язык, описывая структуру правильных цепочек языка. Структуру скобочных выражений можно представить либо как конкатенацию двух стоящих рядом скобочных выражений, либо как вложенное в скобки скобочное выражение. «Крайним частным случаем» является просто отсутствие скобок:

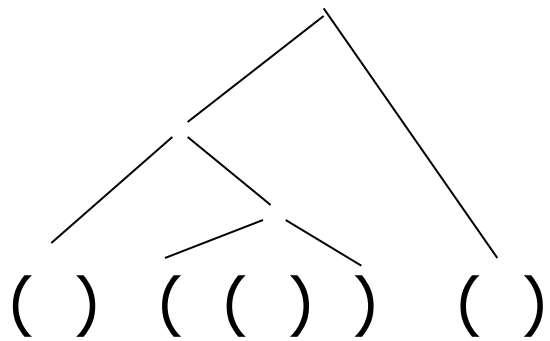
G1:: S→ SS

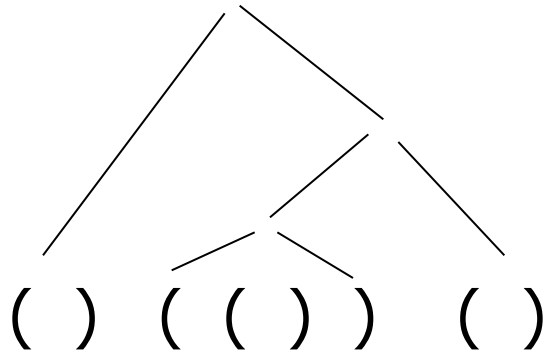
S→ (S)

S→ ε

Многие цепочки языка L имеют различные деревья вывода в грамматике G – это все такие скобочные выражения, которые неоднозначно разбиваются на пару стоящих рядом скобочных выражений.

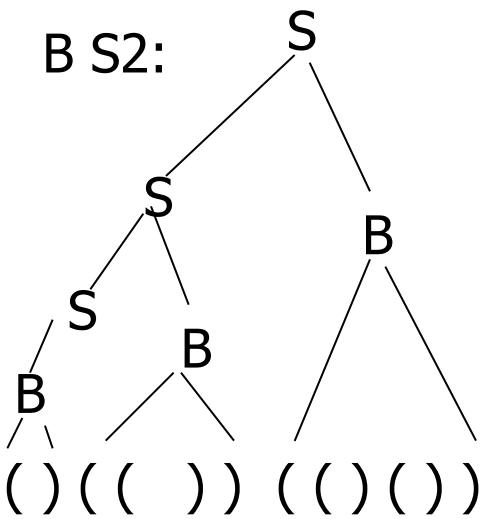
Например, это справедливо для цепочки '( ) (( )) ( )'





Двусмысленные и недвусмысленные грамматики

- По некоторым двусмысленным грамматикам можно построить эквивалентные недвусмысленные грамматики
- Пример: язык правильных скобочных выражений

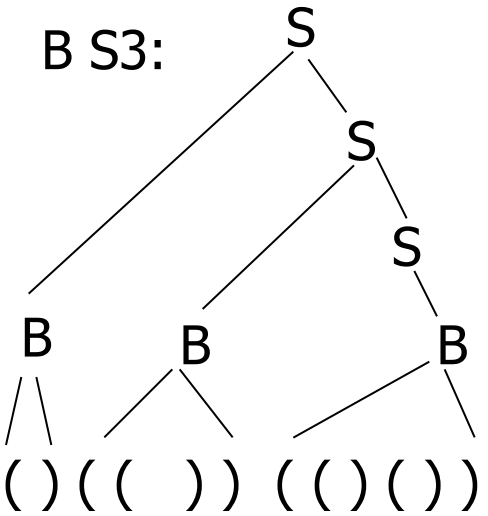


Цепочка: () (( )) (( ) ( ))

G1:: S→ (S) | SS | ε

G2:: S → SB | B
B → (S) | ( )

G3:: S → BS | B
B → (S) | ( )



В - только структура вложенных скобок

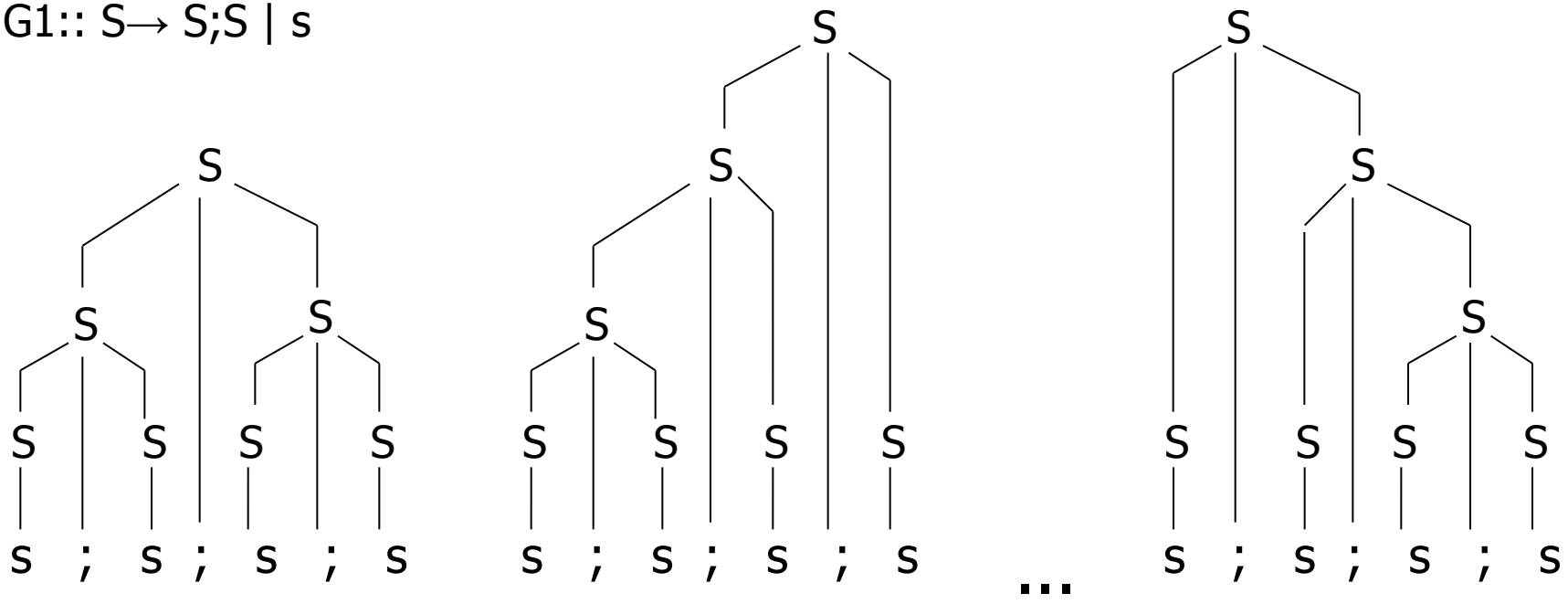
G1 – двусмысленная грамматика,
G2 и G3 – недвусмысленные грамматики

Таким образом часто можно вместо двусмысленной грамматики построить однозначную эквивалентную грамматику

Двусмысленные и недвусмысленные грамматики (2)

- Пример: s; s; s; s - язык последовательностей операторов с “;”

G1:: S→ S;S | s

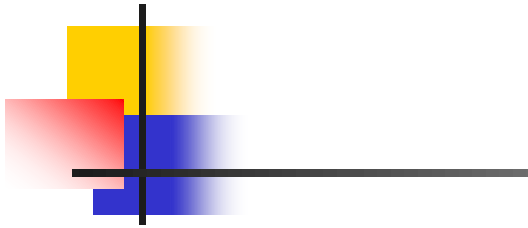


Сколько всего деревьев вывода у цепочки s; s; s; s в этой грамматике?

Проблема проверки того, является ли произвольная грамматика однозначной, алгоритмически неразрешима

Свойства КС-грамматик (грамматик типа 2)

- Многие свойства КС-грамматики являются разрешимыми.
  Можно получить ответы на такие вопросы:
  - порождает ли данная грамматика бесконечный язык?
  - порождается ли данная цепочка данной КС-грамматикой (синтаксический анализ)?
  - является ли язык, порождаемый КС-грамматикой, пустым?
- Для КС-грамматик существуют неразрешимые проблемы
  - для пересечения двух языков, порождаемых двумя КС-грамматиками, неразрешимы проблемы пустоты, автоматности и контекстной свободности
  - проблема двусмысленности КС-грамматик неразрешима: не существует никакой общей процедуры, позволяющей ответить на этот вопрос для любой КС-грамматики
- В большинстве случаев можно написать программу, которая определяет, однозначна ли данная конкретная КС-грамматика.
  Неразрешимость означает, что невозможно написать одну программу ПРОГ, которая давала бы ответ об однозначности-неоднозначности для всех КС-грамматик. Для некоторых грамматик такой ответ не будет получен, как бы долго ни работала программа ПРОГ
- Практически КС-грамматику считаем однозначной, если она принадлежит одному из известных классов однозначных грамматик (которые будем исследовать далее)



Грамматики типа 0 и 1

Грамматики типа 0 и 1

Тип	Вид правил	Название
0	$\alpha \to \beta$	Неограниченные грамматики
1	$\alpha A \beta \to \alpha \gamma \beta$	Контекстно-зависимые грамматики (КЗ-грамматики)
2	$A \to \gamma$	Контекстно-свободные грамматики (КС-грамматики)
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики (А- грамматики)

Грамматики типа 0 и типа 1 не используются в трансляции формальных языков и, в частности, языков программирования

Пример грамматики типа 1

Σ={a,b}; L={ α | в цепочке α количества вхождений a и b равны}.

Пример грамматики: используем грамматику для порождения цепочек $a^n b^n$ и добавляем в нее правило, позволяющее произвольную перестановку символов:

G:: S $\rightarrow$ ASB | ε

A B $\rightarrow$ B A

A $\rightarrow$ a

B $\rightarrow$ b

Вывод цепочки aabbab:

S$\Rightarrow$ASB $\Rightarrow$ AASBB $\Rightarrow$ AAASBBB $\Rightarrow$ AAABBB $\Rightarrow$ AABABB $\Rightarrow$ AABBAB $\Rightarrow$* aabbab

Это грамматика типа 0, но существует эквивалентная ей грамматика типа 1

Задача: Построить грамматику типа 1, порождающую этот язык

Грамматики типа 0

- Грамматика, порождающая язык $L=\{a^n b^n c^n \mid n > 0\}$:
  G:: $S \rightarrow aSBC \mid abC$
  $CB \rightarrow BC$
  $bB \rightarrow bb$
  $cC \rightarrow cc$
- Порождение цепочки *aaabbbccc:*
  $S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaabCBCBC \Rightarrow \ldots$.
  - На первом этапе порождено требуемое число символов a и такие же количества символов B и C, хотя они и не стоят в нужном порядке. Идея этой грамматики в том, что эти нетерминальные символы можно преобразовать в терминальные только тогда, когда они займут требуемое место в выводимой цепочке. За этим следит так называемый "*контекст":* например, bB→ bb: нетерминал B можно преобразовать в терминал b только если ему предшествует уже преобразованный в терминал символ b. Такая подстановка называется "контекстно-зависимой".
- Дальнейшие подстановки:
  aaab**CB**CBC ⇒ aaabBC**CB**C ⇒ aaabB**CB**CC ⇒ aaa**bB**BCCC ⇒ aaab**bB**CCC ⇒ aaabb**bC**CC ⇒ aaabbb**cC**C ⇒ aaabbbc**cC** ⇒ aaabbbccc
- Кроме правила CB→BC все остальные продукции удовлетворяют ограничениям грамматик типа 1

Грамматики типа 1: перестановки нетерминалов

- Продукцию CB→ BC с перестановкой нетерминалов очень просто представить тремя продукциями (контекстно-зависимыми) с эквивалентными возможностями

- Действительно, пусть R – новый нетерминал. Построим три новых правила:

  CB→CR
  CR→BR
  BR→BC

  Очевидно, что применение этих продукций приведет к перестановке C и B, причем это достигнуто с помощью только продукций, удовлетворяющих ограничениям для грамматик типа 1

- Итак, язык $L=\{a^n b^n c^n \mid n > 0\}$ можно породить грамматикой только с контекстно-зависимыми продукциями, и поэтому он является контекстно-зависимым языком

Порождающая грамматика типа 1 языка $\{a^{2\uparrow n} \mid n \geq 0\}$

- $L=\{a^{2\uparrow n} \mid n \geq 0\}$ –цепочки из a длиной 1, 2, 4, 8, 16, 32, …

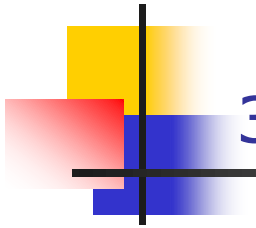
1. $S \to QAQ$
2. $QA \to QAAM$
3. $MA \to AAM$
4. $MQ \to Q$
5. $A \to a$
6. $Q \to \varepsilon$

Идея: каждый символ A заменяется на AA, и маркер M проходит к следующему символу, пока не достигнет правого края цепочки, где он выбрасывается

(правило 3 можно преобразовать в правила типа 1)

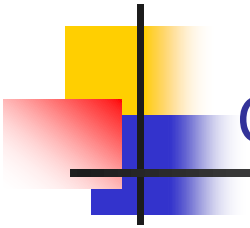
$S \Rightarrow$
$QAQ \Rightarrow QAAMQ \Rightarrow$
$QAAQ \Rightarrow QAAMAQ \Rightarrow QAAAAMQ \Rightarrow$
$QAAAAQ \Rightarrow QAAMAAAQ \Rightarrow QAAAAMAAQ \Rightarrow QAAAAAAMAQ \Rightarrow QAAAAAAAAMQ \Rightarrow$
$QAAAAAAAAQ \Rightarrow \quad ... \quad \Rightarrow$ aaaaaaaa

Никакие цепочки из a, не являющиеся степенью 2, не могут быть порождены:
1. Пока маркер M не будет выброшен, строка не будет терминальной, т.е. M должен дойти до конца
2. Может быть несколько маркеров M, идущих слева. Каждый новый маркер при своем прохождении увеличивает число символов A вдвое



Задача

- Построить грамматику типа 1, порождающую цепочки из символов a длиной 1, 4, 9, 16, 25, 36, …
- Удобно использовать соотношение: $(n^2+1) = n^2+2n+1$



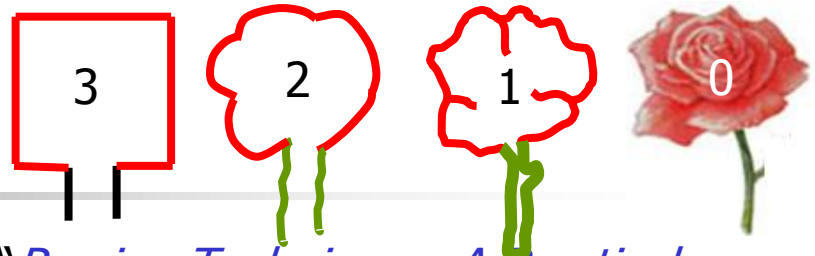
Свойства грамматик типа 1

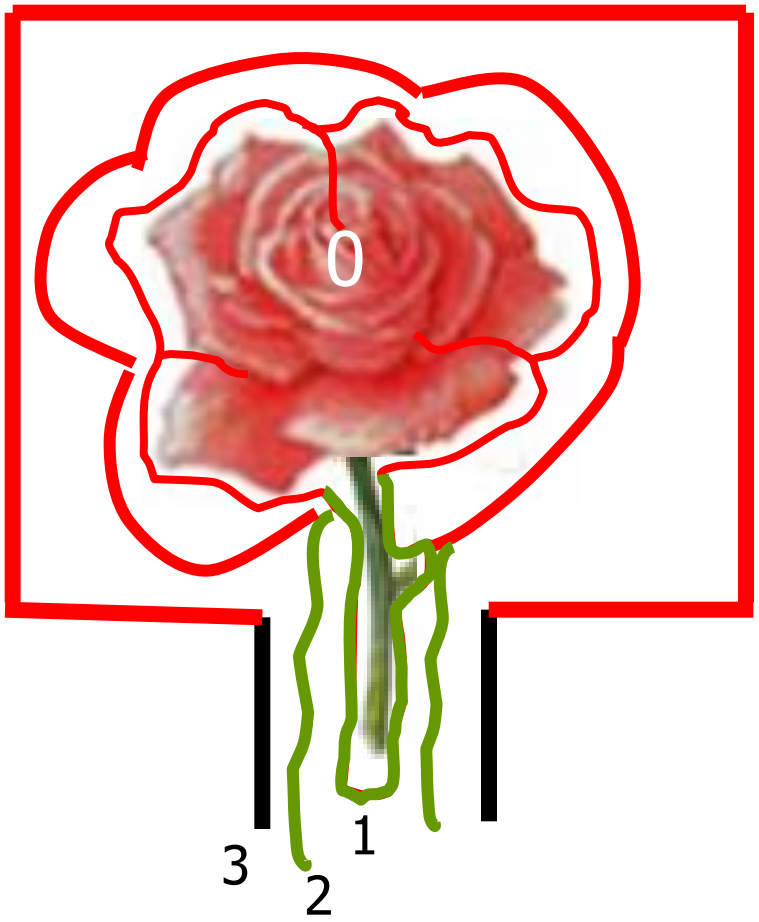
- Все цепочки, последовательно выводимые из начального символа, имеют длину не меньшую, чем предыдущая, поскольку каждое правило должно оставлять длину цепочки неизменной или увеличивать ее
- Контекстно-зависимые грамматики генерируют цепочки, для хранения которых требуется фиксированный объем памяти. Например, такие грамматики способны распознавать цепочки вида a^n b^n c^n, что не может сделать контекстно-свободная грамматика
- Контекстно-зависимые грамматики обычно слишком сложны, чтобы быть практически полезными для моделирования языков программирования
- Некоторые свойства контекстно-зависимых грамматик до сих пор не исследованы

Свойства грамматик типа 0

- Грамматики типа 0 могут использоваться для распознавания любой вычислимой функции, т.е. той, которую можно определить с помощью машины Тьюринга. Например, можно создать грамматику для цепочки a^n b^f(n), представляющей функцию f(n). По заданному n — числу элементов a, эта грамматика генерирует цепочку, содержащую f(n) элементов b
- Большая часть свойств грамматик типа 0 относится к неразрешимым. Это означает, что не существует процесса, с помощью которого можно было бы определить, выполняется ли данное свойство для всех грамматик типа 0, например:
  - пустота (бесконечность) языка, порождаемого произвольной грамматикой типа 0
  - распознаваемость языка, порождаемых грамматикой типа 0
  - пустота пересечения языков, порождаемых двумя грамматиками типа 0
- Отличие от контекстно-зависимых грамматик заключается в том, что у последних о многих свойствах просто ничего не известно — они могут быть истинными, ложными или быть неразрешимыми.

Сравнение типов грамматик





В книге "*Parsing Techniques. A Practical Guide*" (D.Grune, C.Jacobs, 1998) есть четыре рисунка розы, в соответствии с типами грамматик, как бы использованными для рисования:

- Тип 3: Автоматная грамматика - грубый рисунок, еле угадывается силуэт цветка, но какой он - неясно
- Тип 2: КС-грамматика: цветок изображен более точно, некоторые линии прочерчены тоньше, силуэт похож на цветок
- Тип 1: Контекстно Зависимая грамматика – цветок нарисован еще более тонко, уже угадывается роза
- Тип 0: использование формализма неограниченной грамматики дает весьма точное изображение

Это метафора, отражающая соотношение возможностей разных типов грамматик

Выразительная мощность грамматик убывает с увеличением типа

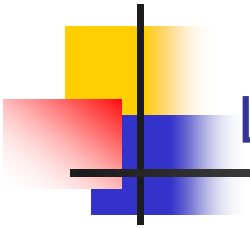
- Самые выразительные - это грамматики типа 0, потом типа 1 и т.д.
- Относительно языка Си эту мысль можно выразить так:
  - под корректными предложениями языка можно понимать программы написанные на языке Си
  - грамматика языка Си должна генерировать все корректные программы на Си и не должна генерировать ни одной некорректной программы
  - КС-грамматика, порождающая все программы на языке Си, генерирует и корректные программы на Си, но также генерирует и некорректные. Например:

```
void f( ) {
  a = 0;       // Ни одного объявления переменной
 }
```

Эта программа синтаксически корректна в том смысле, что она порождается грамматикой языка Си. Но она некорректна в том смысле, что в ней нет объявлений переменной

В КС грамматике языка Си требование использования объявлений до использования переменной выразить НЕЛЬЗЯ. Это контекстное правило

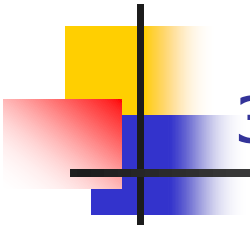
Правило согласования типов – еще одно КОНТЕКСТНОЕ правило, его нельзя выразить контекстно-свободной грамматикой. Проверка этих ограничений в КС-грамм выполняется семантическими действиями



Цитата из книги Рассела и Норвига [1]

- Грамматики, расположенные выше по иерархии Хомского, имеют большую выразительную мощь, но алгоритмы для работы с ними являются менее эффективными.
- Вплоть до середины 80-х годов усилия лингвистов были, в основном, сосредоточены на контекстно-свободных и контекстно-зависимых языках. Но в дальнейшем стали шире применяться регулярные грамматики, что было вызвано необходимостью очень быстро обрабатывать мегабайты и гигабайты текста в оперативном режиме, даже за счет менее полного анализа.
- Как сказал [известный лингвист] Фернандо Перейра: "*Чем старше я становлюсь, тем ниже спускаюсь по иерархии Хомского*"

[1] Стюарт Рассел, Питер Норвиг. Искусственный интеллект. Современный подход. 2006

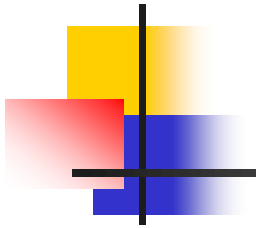


Заключение

- Основной проблемой анализа формальных языков является проблема восстановления вывода произвольной цепочки языка из начального символа порождающей грамматики
- Проблема восстановления вывода произвольной цепочки в произвольной грамматике Хомского алгоритмически неразрешима
- Как и во многих областях формального анализа, часто более простые, частные формализмы описывают важные для практики классы объектов. В порождающих грамматиках Хомского выделены более простые подклассы грамматик: КЗ-грамматики, КС-грамматики и А-грамматики (соответственно грамматики типа 1, типа 2 и типа 3)
- Грамматики называются эквивалентными, если они порождают один и тот же язык. Для каждого языка можно построить несколько грамматик. Но неверно, что для любого языка можно построить очень простую, например, А-грамматику
- Существует вложение языков, аналогичное вложению порождающих грамматик Хомского: существуют языки, которые порождаются грамматикой типа 2, но для них не существует порождающей грамматики типа 3, существуют языки, порождаемые грамматиками типа 1, но не порождаемые никакими грамматиками типа 2, существуют языки, порождаемые грамматиками типа 0, но не порождаемые никакими грамматиками типа 1

Заключение (2)

- Для КС-грамматик существует полиномиальный алгоритм, который восстанавливает дерево вывода цепочки в заданной грамматике
- КС-грамматика однозначна, если для любой выводимой цепочки имеется ровно одно дерево вывода. В общем случае КС-грамматика может быть неоднозначной: для одной цепочки языка можно построить несколько деревьев вывода (т.е. цепочка может иметь несколько структур)
- Выразительная мощность языка падает с увеличением типа порождающей его грамматики (метафора – рисунок розы):
  - грамматики типа 0 могут порождать цепочки с весьма тонкими свойствами (например, объявления переменных и т.п.)
  - грамматики типа 1 могут порождать цепочки со свойствами контекстных зависимостей (например, согласования типов)
  - грамматики типа 2 могут порождать цепочки с учетом вложенностей конструкций, но не могут учесть контекстные зависимости
  - грамматики типа 3 (автоматные) порождают цепочки языков, в которых нельзя выразить даже вложенность конструкций



Спасибо за внимание