2024

КЫК, [] и их друзья

29 августа 2024 г.

Содержание

Часть I

Чистка грамматики

Грамматики могуть быть "плохими": там могут быть лишние символы, лишние правила
Такие грамматики хочется упростить, и убрать из них всяческий мусор

1 Исключение непорождающих переменных

Примем следующие обозначения:

T - алфавит (набор Терминалов языка)
N - набор переменных (Nетерминалов, символов)
P - множество продукций (порождающих правил)
S - стартовый символ
$G =< T, N, P, S >$ - грамматика

$T*$ - замыкание алфавита
L - язык

w - терминал
$L(T*)$ - множество всех возможных строк терминалов
N^G - множество всех возможных строк нетерминалов
$L(T \cup N^G)$ - множество всех возможных строк из терминалов и нетерминалов

Переменная порождает строчку нетерминалов $A \in N^G$, если есть продукции вида:

1. $A \to w$, где $w \in L(T*)$
2. $A \to \alpha$, где $\alpha \in L(T \cup N^G)+$

И на примере это выглядит так:

$T = \{a, b, c\}$
$N = \{A, B, C, S\}$
$P = \{S \to AB,\ S \to C,\ A \to aA,\ A \to a,\ B \to bB,\ C \to c\}$

Действуем итерационно:

1. Ищем правила первого типа: $A \to a, C \to c$

   То есть $C,\ A$ - порождающие

2. Теперь, когда базис собран - ищем правила второго типа, содержащие нетерминалы и порождающие символы: $S \to C,\ A \to aA,$ - то есть S - порождающий

3. Итого, в наши порождающие символы в итоге попали $\{C, A, S\}$

   А вот B ничего в итоге не порождает, а значит возможно он нам и не нужен?

2 Исключение недостижимых символов

Попробуем пройти в другую сторону - от стартового символа

1. S - достижим
2. $A \to \alpha$, если A достижим, то все символы из α достижимы

В общем-то достаточно логичный набор правил
Рассмотрим что получится на примере:

$$S \to AB,\ A \to C,\ C \to c,\ B \to bB,\ D \to aC,\ D \to bc$$
S достижим - значит A, B достижимы
A достижим - значит C достижим
B достижим - значит b достижим
C достижим - значит c достижим

И остались недостижимыми D, a - может они нам тоже не нужны?

3 Полезные и бесполезные символы и продукции

Собственно символ - **полезный** - если он используется в порождении какой-либо строки терминалов из стартового символа грамматики (ну в общем достижимый и порождающий)

А ещё бывают бесполезные продукции, и с ними надо подробнее разобраться

$A \to \varepsilon$ - это **эпсилон-продукция**

$A \to B$ - это **единичная продукция**

И есть одна интересная теоремка - если L - контекстно-свободный язык, то $L - \{\varepsilon\}$ можно предствить в виде контекстно-свободной грамматики без эпсилон-продукций

И есть из этой теоремки забавное следствие: Любой контекстно-свободный язык, содержащий пустую строку, можно представить контекстно-свободной грамматикой вида: $G =< T, N, P \cup \{S \to \varepsilon\}, S >$, где P не содержит эпсилон-продукций

3.1 Избавляемся от эпсилон-продукций

Идём дальше - теперь разберёмся с такой штукой как **зануляемый символ**, оно нам пригодится
Логично - что это символ A такой что

1. $A \to \varepsilon$
2. $A \to \alpha$, где α - строка из зануляемых символов

А теперь... АЛГОРИТМ (хотя как завещала Бабалова - алгоритм был сначала)

1. Находим зануляемые символы
2. (могу сюда вставить честное определение, но....) Во всех продукциях добавляем всевозможные комбинации, где зануляемые символы явно занулены (убраны к чертовой матери)
3. Исключаем все эпсилон-продукции, кроме той, что со стартовым символом слева
4. Исключаем все бесполезные символы

ПРИМЕР: $S \to ABC,\ A \to aA \mid \varepsilon,\ C \to c \mid \varepsilon,\ B \to bB \mid A$

1. Видно что зануляемыми у нас в итоге являются: A, C, B, S
2. Вот тут начинается веселье
   $S \to ABC$ - заменяем на $S \to ABC|BC|AC|AB|C|B|A|\varepsilon$
   $A \to aA$ - заменяем на $A \to aA|a|\varepsilon$
   $C \to c|\varepsilon$ - не не изменилось
   $B \to bB|A$ - заменяем на $B \to bB|A|b|\varepsilon$
3. В итоге остаются только:
   $S \to ABC|BC|AC|AB|C|B|A|\varepsilon$
   $A \to aA|a$
   $C \to c$
   $B \to bB|A|b$

3.2 Избавляемся от единичных продукций

Теперь нам бы ещё надо избавиться от единичных продукций

1. Продукции вида $A \to^* B$ и $B \to \alpha$ заменяем на $A \to \alpha$ - по сути просто сокращаем множество транзитивных переходов до одного
2. (Исключаем все бесполезные символы)

ПРИМЕРЧИК:
$S \to ABE, A \to aA|a, C \to c, B \to A|D, D \to C, E \to B|C|e$

И теперь заменяем все эти единичные переходы...
$S \to ABE, A \to aA|a, C \to c, B \to aA|a|c, D \to c, E \to aA|a|c|e$

Ну и теперь, в принципе можно убрать лишние символы: C, D
$S \to ABE, A \to aA|a, B \to aA|a|c, E \to aA|a|c|e$

4 Алгоритм очистки грамматики!!!

1. Исключаем эпсилон-продукции
2. Исключаем единичные продукции
3. Исключаем все бесполезные символы
4. Ну и если стартовый символ зануляемый - оставялем $S \to \varepsilon$

Часть II

Нормальная форма Хомского (Chomskiy Normal Form, CNF)

Грамматика является грамматикой в нормальной форме Хомского, если включает только продукции вида:

1. $A \to BC$, где $A, B, C \in N$
2. $A \to a$, где $A \in N, a \in T$

И это очень строгие ограничения - и зачем оно нам вообще надо?
Ну в общем-то чтобы применить наш простой парсер КЯК

5 Алгоритм построения CNF (жоский спам нетерминалами)

1. Очистка грамматики - и теперь в грамматике только продукции из 1 терминала или длина правой части правил не меньше 2х
2. Для каждого терминала создаём продукцию $A_a \to a$ и заменяем a на A_a во всех продукциях с длиной ≥ 2 - и теперь у нас есть только продукции из 1 терминала или только из нетерминалов, длины ≥ 2
3. Ну и теперь заменяем все продукции с длиной ≥ 2 на грамматики по такому алгоритму
$A \to B\alpha$, где $|\alpha| \geq 2$ превращаем в $A \to BC$ и $C \to \alpha$

А теперь - пример всего этого дела.

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to id \mid (E)$$

Как тут видно, это ни разу не нормальная форма Хомского. Так что идём по нашему алгоритму.
Для начала - очистим грамматику
Получаем уже вот так (развернули всё что можно развернуть)

$$E \to E + T \mid T * F \mid id \mid (E)$$
$$T \to T * F \mid id \mid (E)$$
$$F \to id \mid (E)$$

А теперь начинается колдовство!
Для наших терминалов $(,), +, *, id$ создаём нетерминалы, и заменяем на них в продукциях, с длиной больше 2

$$E \to EPT|TMF|id|LER$$
$$T \to TMF|id|LER$$
$$F \to id|LER$$
$$L \to ($$
$$R \to)$$
$$P \to +$$
$$M \to *$$
$$I \to id$$

Выглядит это конечно жутковато (но можем сразу выкинуть I, потому что он нигде не нужен). Но вот теперь следуем третьему шагу - заменяем все комбинации из нескольких символов на новые символы так, чтобы получились нормальные продукции.

$$E \to EA|TB|id|LC$$
$$T \to TB|id|LC$$
$$F \to id|LC$$
$$L \to ($$
$$R \to)$$
$$P \to +$$
$$M \to *$$
$$A \to PT$$
$$B \to MF$$
$$C \to ER$$

Вуаля, наша грамматика в нормальной форме Хомского.

Часть III

КЯК парсер

Рассмотрим пример грамматики ниже. Возьмём для разбора строку $baaba$.

$$S \to AB \mid BC$$

$$A \to BA \mid a$$

$$B \to CC \mid b$$

$$C \to AB \mid a$$

Суть КЯК в рассмотрении подстрок от самых маленьких к целой строке. По итогу нам нужно узнать, является ли строка каким-то из нетерминалов (т.е. словом языка, порождённого грамматикой). Если совсем кратко - принадлежит ли строка языку.

Приступим к разбору. Ещё раз напомню, что мы его ведём снизу вверх - от терминалов к нетерминалам и так пока не охватим всю строку.

$$B \to b$$

$$A \to a$$

$$C \to a$$

Идём дальше. Это были подстроки длины 1. $baaba$ также имеет подстроки длины 2, которые формируются из подстрок длины 1 ($\times$ - декартово произведение):

$$ba = b \times a = B \times A = BA = A$$

$$ba = b \times a = B \times C = BC = S$$

Пока у нас есть $A \to ba, S \to ba$. Которые получаются, как мы выяснили, вот так:

$$A \to BA \to bA \to ba$$

$$S \to BC \to bC \to ba$$

Тупо последовательно заменяем нетерминалы, пока не доберёмся до терминалов. К сожалению, дальше мы упремся и запутаемся, если будем так продолжать. Это была всего лишь первая 2-подстрока, а я уже устала.
Видимо, кто-то из авторов тоже устал, и придумал лестницу КЯК или как она там. Выглядит она следующим образом (цифры - длины подстрок, НТ - нетерминалы):

5	НТ				
4	НТ	НТ			
3	НТ	НТ	НТ		
2	НТ	НТ	НТ	НТ	
1	НТ	НТ	НТ	НТ	НТ
	b	a	a	b	a

Эта таблица показывает, из каких нетерминалов какую строку мы можем построить. Пока что для нас она выглядит так:

5	НТ				
4	НТ	НТ			
3	НТ	НТ	НТ		
2	НТ	НТ	НТ	НТ	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

И, учитывая что мы уже успели проанализировать первое ba:

5	НТ				
4	НТ	НТ			
3	НТ	НТ	НТ		
2	S,A	НТ	НТ	НТ	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

Теперь давайте заполнять эту таблицу. Для следующих подстрок мы получим:

$$aa = a \times a = A, C \times A, C = AA, CA, AC, CC = B$$

$$ab = a \times b = A, C \times B = AB, CB = S, C$$

$$bb = b \times b = B \times B = \varnothing$$

5	НТ				
4	НТ	НТ			
3	НТ	НТ	НТ		
2	S,A	B	S,C	S,A	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

Строки длины 3.

$$baa = b \times aa = B \times B = BB = \varnothing$$

$$baa = ba \times a = S, A \times A, C = SA, AA, SC, AC = \varnothing$$

$$aab = a \times ab = A, C \times S, C = AS, CS, AC, CC = B$$

$$aab = aa \times b = B \times B = \varnothing$$

$$aba = a \times ba = A, C \times S, A = AS, CS, AA, CA = \varnothing$$

$$aba = ab \times a = S, C \times A, C = SA, CA, SC, CC = B$$

5	НТ				
4	НТ	НТ			
3	∅	B	B		
2	S,A	B	S,C	S,A	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

Строки длины 4.

$$baab = b \times aab = B \times B = BB = \varnothing$$

$$baab = ba \times ab = S, A \times S, C = SS, AS, SC, AC = \varnothing$$

$$baab = baa \times b = \varnothing \times B = \varnothing$$

$$aaba = a \times aba = A, C \times B = AB, CB = S, C$$

$$aaba = aa \times ba = B \times S, A = BS, BA = A$$

$$aaba = aab \times a = B \times A, C = BA, BC = A, S$$

5	НТ				
4	∅	S,A,C			
3	∅	B	B		
2	S,A	B	S,C	S,A	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

Строки длины 5.

$$baaba = b \times aaba = B \times S, A, C = BS, BA, BC = A, S$$

$$baaba = ba \times aba = S, A \times B = SA, AB = S, C$$

$$baaba = baa \times ba = \varnothing \times S, A = \varnothing$$

$$baaba = baab \times a = \varnothing \times A, C = \varnothing$$

5	S,A,C				
4	∅	S,A,C			
3	∅	B	B		
2	S,A	B	S,C	S,A	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

Таким образом, строка может быть получена путем последовательного раскрытия (развертки) нетерминалов S, A, C.

УПРАЖНЕНИЕ. Убедиться, что все указанные нетерминалы преобразуются в искомую строку.

Часть IV

LL1 парсер

Всё начинается с таблицы парсинга. Для составления таблицы необходимо вначале посчитать множества символов FIRST и FOLLOW для каждого нетерминала NT.
Заметим, что первое правило грамматики для стартового нетерминала, если не указано иное. После стартового нетерминала может идти конец строки, для остальных в общем случае это неверно. Рассмотрим следующую грамматику:

$$A \to CB$$

$$B \to +CB \mid \varepsilon$$

$$C \to ED$$

$$D \to *ED \mid \varepsilon$$

$$E \to id \mid (A)$$

FIRST(NT) - множество символов на первой позиции строки NT (aka begin()):

$$FIRST(A) = FIRST(C)$$

$$FIRST(B) = \ '+' \cup \varepsilon$$

$$FIRST(C) = FIRST(E)$$

$$FIRST(D) = \ '*' \cup \varepsilon$$

$$FIRST(E) = \ 'id' \cup \ '('$$

FOLLOW(NT) - множество символов на первой после последней позиции строки NT (aka end()) (т.е.
1) если видим $P \to ...NT\ Q$, то добавляем $FIRST(Q)$ (если Q непусто) или $FOLLOW(P)$ (если Q пусто)
2) если Q может быть ε, то добавляем $FOLLOW(Q)$
3) ε не может быть в FOLLOW, если появляется, то его надо убрать)

$$FOLLOW(A) = \ ')' \cup \ '\$'$$

$$FOLLOW(B) = FOLLOW(A)$$

$$FOLLOW(C) = FIRST(B) \cup FOLLOW(A) \cup FOLLOW(B) \ \setminus \varepsilon$$

$$FOLLOW(D) = FOLLOW(C) \ \setminus \varepsilon$$

$$FOLLOW(E) = FIRST(D) \cup FOLLOW(C) \cup FOLLOW(D) \ \setminus \varepsilon$$

Получм:

NT	FIRST	FOLLOW
A	id, (	), $
B	+, ε	), $
C	id, (	+,), $
D	*, ε	+,), $
E	id, (	*, +,), $

Теперь, таблица парсинга. Она содержит производящие правила на пересечении терминалов и нетерминалов. Заполняется она так для каждого нетерминала NT:
1. Если $\varepsilon \in FIRST(NT)$, то пишем правило, которым получился ε, во всех строках из FOLLOW(NT).
2. Для всех прочих символов (не $) из FIRST(NT) в соответствующих столбцах пишется правило, которым этот символ был получен.

NT/T	id	(	+	)	*	$
A	$A \to CB$	$A \to CB$				
B			$B \to +CB$	$B \to \varepsilon$		$B \to \varepsilon$
C	$C \to ED$	$C \to ED$				
D			$D \to \varepsilon$	$D \to \varepsilon$	$D \to *ED$	$D \to \varepsilon$
E	$E \to id$	$E \to (A)$				

Удивительно, но алгоритм предельно прост. Мы всего лишь инициализируем стек стартовым нетерминалом, а в конец рассматриваемой строки добавляем конечный символ $.

Далее происходит следующее. Мы достаём символ из строки и нечто со стека. Если нечто - терминал и совпало с символом, то *pop* и двигаем текущий символ. Если не совпало - ошибка. (При любой ошибке выходим из цикла.)
Если же нечто - нетерминал, то мы ищем в таблице запись на пересечении нечто и текущего символа строки. Если запись не найдена, то ошибка. Иначе *pop*. Правая часть записи добавляется в стек, так что первый её элемент наверху.

Так повторяется, пока текущий символ не станет концом строки или не будет получена ошибка. Если ошибок не было, то строка принадлежит языку.

Рассмотрим пример. Грамматика

$$S^{'} \to S\$$$

$$S \to xYzS \mid a$$

$$Y \to xYz \mid y$$

и строки $xxyzza$ и $xxyzzz$.
Таблица 1:

NT	FIRST	FOLLOW
$S^{'}$	x,a	$
S	x,a	$
Y	x,y	z,$

Таблица 2:

NT/T	a	x	y	z	$
$S^{'}$	$S^{'} \to S\$$	$S^{'} \to S\$$			
S	$S \to a$	$S \to xYzS$			
Y		$Y \to xYz$	$Y \to y$		

Начинаем разбор строки $xxyzza$:

Стек: $S^{'}$
Строка: $xxyzza\$$
Символ: x
Нечто - $S^{'}$
Нечто - нетерминал, и потому ищем пересечение $S^{'}$ и x. Находим $S^{'} \to S\$$.
Стек: $\$S$

Строка: $xxyzza\$$
Символ: x
Нечто - S
Нечто - нетерминал, и потому ищем пересечение S и x. Находим $S \to xYzS$.
Стек: $\$SzYx$

Строка: $xxyzza\$$
Символ: x
Нечто - x

Нечто - терминал, и он совпал с текущим символом.
Стек: $\$SzY$
Строка: $xyzza\$$

Символ: x
Нечто - Y
Нечто - нетерминал, и потому ищем пересечение Y и x. Находим $Y \to xYz$.
Стек: $\$SzzYx$

Строка: $xyzza\$$
Символ: x
Нечто - x
Нечто - терминал, и он совпал с текущим символом.
Стек: $\$SzzY$
Строка: $yzza\$$

Символ: y
Нечто - Y
Нечто - нетерминал, и потому ищем пересечение Y и y. Находим $Y \to y$.
Стек: $\$Szzy$

Строка: $yzza\$$
Символ: y
Нечто - y
Нечто - терминал, и он совпал с текущим символом.
Стек: $\$Szz$
Строка: $zza\$$

Символ: z
Нечто - z
Нечто - терминал, и он совпал с текущим символом.
Стек: $\$Sz$
Строка: $za\$$

Символ: z
Нечто - z
Нечто - терминал, и он совпал с текущим символом.
Стек: $\$S$
Строка: $a\$$

Символ: a
Нечто - S
Нечто - нетерминал, и потому ищем пересечение S и a. Находим $S \to a$.
Стек: $\$a$

Строка: $a\$$
Символ: a
Нечто - a
Нечто - терминал, и он совпал с текущим символом.
Стек: $
Строка: $

Символ: $
Нечто - $
Нечто - $, и он совпал с текущим символом.
Стек:
Строка:

Стек опустел, значит цикл закончился. Ошибки не было, посему строка $xxyzza$ принадлежит языку.

Теперь строка $xxyzzz$. Здесь всё то же самое до момента:

Стек: $\$Sz$
Строка: $zz\$$

Символ: z
Нечто - z
Нечто - терминал, и он совпал с текущим символом.
Стек: $\$S$
Строка: $z\$$

Символ: z
Нечто - S
Нечто - нетерминал, и потому ищем пересечение S и z. Не нашли. ОШИБКА. Выходим.
Таким образом, строка $xxyzzz$ не принадлежит языку.

Часть V

LR(0): то, ради чего мы сюда пришли

Мы преодолели долгий путь, но тут на нас напали конечные автоматы. Присаживайтесь поудобнее, сейчас начнётся реальная мясорубка.
LR-челики строятся на двух основных операциях: сдвиг и свертка:

Сдвиг:
Стек: aBC, Строка: def
Стек: aBCd, Строка: ef

Свертка (предполагая правило A->BC):
Стек: aBC
Стек: aA

Вся сложность парсинга состоит в том, что произвольную строку не спарсить, тыкая наугад либо сдвиг либо свертку. Последовательностей действий может быть разной, и какая-то будет успешной, а какая-то нет (это как игра в карты, непонятно когда какую карту выложить, а когда придержать). Вообще, ТА - то ещё казино. Особенно когда грамматика некорректная. Ну да ладно.
Короче, какой-то больной на голову человек придумал анализировать сдвиги через ввод дополнительного символа. В разных источниках его обозначают по-разному, мы будем использовать _. Производящее правило с _ в правой части правила называется LR(0)-ситуацией.
Рассмотрим грамматику

$$S \to TF$$

$$F \to +T$$

$$T \to a \mid (F)$$

Поехали строить LR(0) состояния. Изначально вспомогательный символ всегда находится в начале правой части производящего правила.

$$S \to \_TF$$

Далее рекурсивно повторяем это для нетерминала перед _, в данном случае T.

$$T \to \_a,\ T \to \_(F)$$

Итак, 1. $S \to \_TF,\ T \to \_a,\ T \to \_(F)$.

Теперь двигаем _ во всех LR(0)-ситуациях. Сдвиг относительно каждого отдельного терминала или нетерминала порождает новое состояние.

$$2.\ S \to T\_F$$

$$3.\ T \to a\_$$

$$4.\ T \to (\_F)$$

2е и 4е состояния порождает ситуацию _NT, так что опять плодят в себе другие состояния:

$$F \to \_+T$$

В итоге

$$2.\ S \to T\_F,\ F \to \_+T$$

$$3.\ T \to a\_$$

$$4.\ T \to (\_F),\ F \to \_+T$$

Далее из 2

$$5.\ S \to TF\_$$

$$6.\ F \to +\_T,\ T \to \_a,\ T \to \_(F)$$

Из 4

$$7.\ T \to (F\_)$$

Из 6

$$8.\ F \to +T\_$$

Из 7

$$9.\ T \to (F)\_$$

Соберём состояния воедино:

$$1.\ T \to \_a,\ T \to \_(F),\ S \to \_TF$$

$$2.\ F \to \_+T,\ S \to T\_F$$

$$3.\ T \to a\_$$

$$4.\ T \to (\_F),\ F \to \_+T$$

$$5.\ S \to TF\_$$

$$6.\ F \to +\_T,\ T \to \_a,\ T \to \_(F)$$

$$7.\ T \to (F\_)$$

$$8.\ F \to +T\_$$

$$9.\ T \to (F)\_$$

Сост/Элм	a	+	(	)	конец	F	T
1	сдвиг 3		сдвиг 4				сдвиг 2
2		сдвиг 6				сдвиг 5	
3 свертка $T \to a$							
4						сдвиг 7	
5					свертка $S \to TF$, успех		
6							сдвиг 8
7						сдвиг 9	
8 свертка $F \to +T$							
9 свертка $T \to (F)$							

Таким образом, таблица парсинга LR(0) построена.