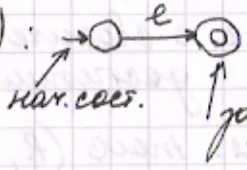
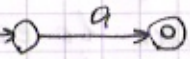
1. НКА по регулярным выражениям.

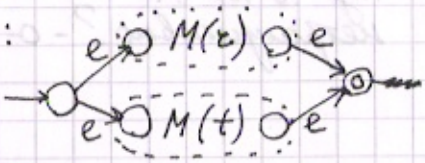
1. Переход по ε (пустой символ):



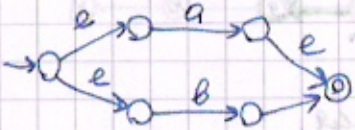
2. Переход по a:



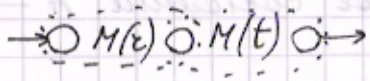
3. $r \mid t$:



Пример: $a \mid b$

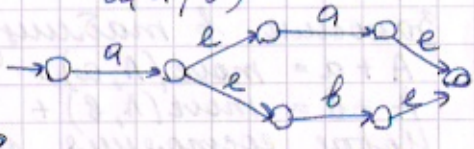


4. rt:

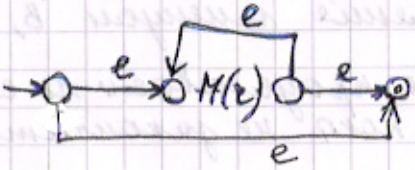


Пример: ab

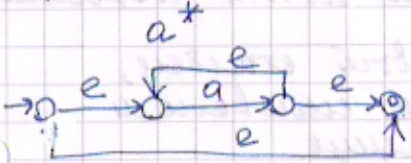
$a(a \mid b)$



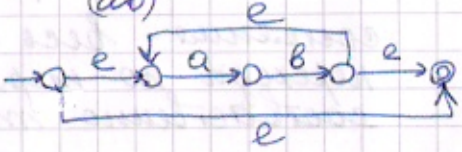
5. r^*:



Пример: a^*



$(ab)^*$



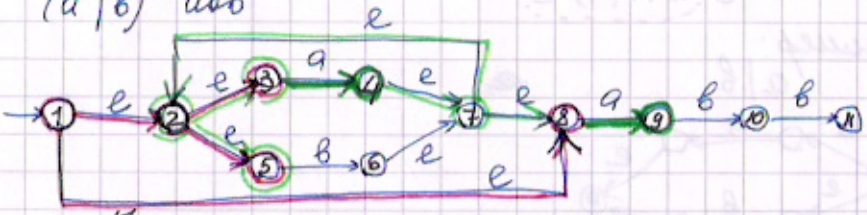
Грамотно комбинируя, получаем нужное

2. Построение ДКА по НКА

1. e-closure(R) - мн-во состояний, достижимых из R по ε. Транзитивно.
2. move(R, a) - мн-во сост., достижимых из R по a.
   нетранзитивно. ? - 0 —a→ 0 —a→ 0

Пример:

(a|b)*abb



1. Пронумеруем состояния.
2. Начальное новое состояние A — e-closure(1).

   A = {1, 2, 3, 5, 8}. (→)

3. Заносим в таблицу новые состояния:
   A + a = move(A, a) + e-closure(move(A, a))
   A + b = move(A, b) + e-closure(move(A, b)).
   Новые состояния именуем B, C ... и т.д.

   То же самое проделываем и с ними. До тех пор, пока не закончатся состояния.

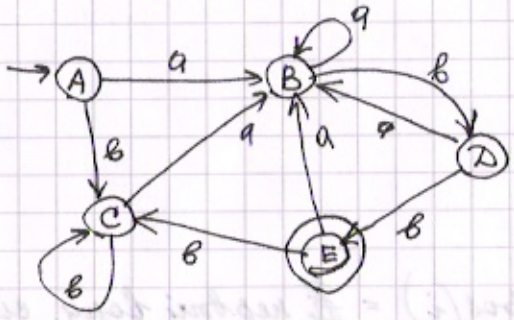
4. По таблице строим новый автомат - состояния - весь первый столбец, переходы по терминальным символам - соответственно таблице.

move(A, a) = {4, 9} (→)
e-closure(move(A, a)) = {7, 8, 2, 3, 5} (→)
⇒ B = {2, 3, 4, 5, 7, 8, 9}

	a	b
→ A {1, 2, 3, 5, 8}	{4, 9, 7, 8, 2, 3, 5} B move ε-closure	{6, 7, 3, 2, 5, 5} C move ε-closure
B	B	{6, 10, 7, 8, 2, 3, 5} D move ε-closure
C	B	C
D	B	{6, 11, 7, 8, 2, 3, 5} E move ε-closure
* E {2, 3, 5, 6, 7, 8, 11}	B	C

5. Начальное состояние - A.
Конечные состояния - все состояния, включающие в себя конечное состояние нашего автомата.

В примере конечное сост. НКА - 11 ⇒ новое конечное состояние E



6. В таблице возможны нули (если move(A, a) = ∅) ⇒ нет перехода из A по a в новом автомате.

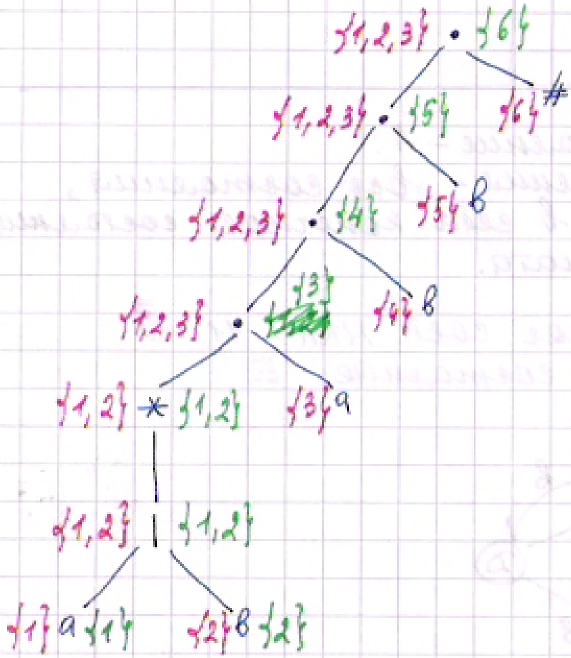
3. Построение ДКА по РВ.

1. ~~[illegible]~~ Дописываем в конец РВ #.

2. Нумеруем позиции (и # тоже).

Пример: $((a|b)^*abb)\#$ — позиции: a = 1, b = 2, a = 3, b = 4, b = 5, # = 6

3. Строим дерево.



4. Определяем firstpos(i) = ~~[illegible]~~ первые возм. символы поддерева i

= {
- • i, если i – лист
- • firstpos(лев.) ∪ firstpos(прав.), если i = |
- • если nullable(лев.), то firstpos(лев) ∪ firstpos(прав.), если i = •
  иначе firstpos(лев.)
- • firstpos(лист), если i = *

а вообще здравый смысл и внимательность тоже помогают, ибо алгоритм ещё учить надо

nullable (i) - обращается ли i в ε хоть когда-нибудь

- false, если i = терм. символ
- true, если i = *
- null (лев) or null (прав), если i = |
- null (лев) and null (прав), если i = •

опять же здравый смысл помогает.
Алгоритм из учебника переписан - стр. 19

lastpos (i) - последний терм. символ поддерева i.

- i, если i - лист
- lastpos (лев) ∪ lastpos (прав), если i = |
- lastpos (лист), если i = *
- если nullable (прав), то lastpos (лев) ∪ lastpos (прав.), если i = •
  иначе lastpos (прав.)

5. Строим табличку - для каждого i - followpos (i) - позиции, которые могут стоять за i

опять же здравый смысл... а вот алгоритм

1) Для операции a • b:
для каждой позиции i ∈ lastpos (a)
добавим в таблицу firstpos (b)

2) Для операции a*:
для каждой позиции i ∈ lastpos (a)
добавим в таблицу firstpos (a).

i	followpos(i)
1	1, 2, 3
2	1, 2, 3
3	4
4	5
5	6
6	∅

6. Строим таблицу переходов.
Начальное состояние A = firstpos(корень дерева)

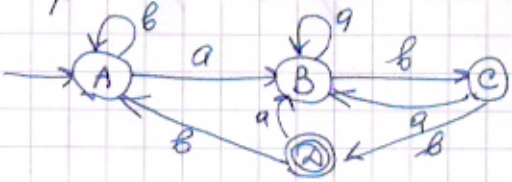
Конечное состояния – состояния, содержащие номер # (в нашем примере – 6).

	a {1,3}	b {2,4,5}
→ A {1,2,3}	{1,2,3,4} B	A
B {1,2,3,4}	B	C {1,2,3,5}
C {1,2,3,5}	B	D {1,2,3,6}
* D {1,2,3,6}	B	A

Заполняем таблицу: A+a = ∪ followpos(i), где i = A ∩ a

в нашем примере:
{1,2,3} ∩ {1,3} = {1,3} → {1,2,3} ∪ {4} = {1,2,3,4}

7. Строим автомат по таблице.



4. Минимизация ДКА.

1. Избавиться от переходов в никуда:
(то есть отсутствующих)
$\delta(q, a) = \emptyset$ – пусто!
Добавим новое состояние q' и будем переходить в него.

2. Строим множества:

Π_0: $\{F\}$ $\{Q \setminus F\}$ ← все остальные.
↑ все конечные сост.

Π_1: если A и $B \in$ одному мн-ву и
$A \xrightarrow{a}$ } в одной группе
$B \xrightarrow{a}$ }
$A \xrightarrow{b}$ } в одной группе
$B \xrightarrow{b}$ }
$\Rightarrow$

A и B остаются в одном мн-ве.

Повторять до стабилизации.

Пример:

	a	b
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

нет пустых мест в таблице – не нужно вводить новое состояние.

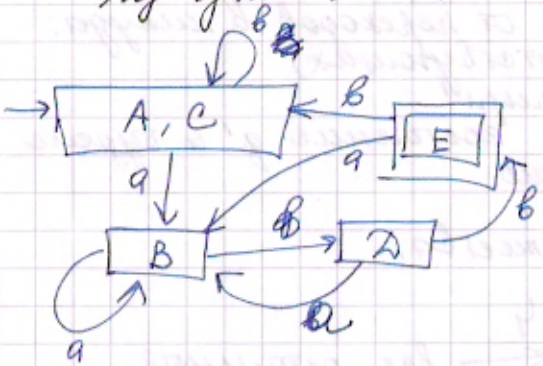
Π_0: $\{E\}$ $\{A, B, C, D\}$ ← есть переход $D \xrightarrow{b} E$, E из другой группы – отрезаем D.

Π_1: $\{E\}$ $\{D\}$ $\{A, B, C\}$ ← есть переход $B \to D$ (остальное в эту же группу) → отрезаем B

Π_2: $\{E\}$ $\{D\}$ $\{B\}$ $\{A, C\}$

Π_3: ——" ——" ——" $= \Pi_2$

3. Строим автомат, объединяя состояния по одного мн-ва



4. + нужно удалить недостижимые состояния и те, из которых нет пути в конечное.

5. Построение КА по правилам грамматике

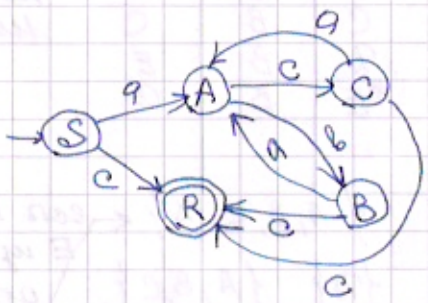
S - нач. сост.
F - конечные сост. = { {S, R}, если S → e
{R} иначе }

A → aB → (A) —a→ (B)

A → a → (A) —a→ (R)

Пример:

S → aA/c
A → bB/cC
B → aA/c
C → aA/c



построение грамматики по ДКА - аналогично.

6. Приведение КС-грамматик

1. Бесполезные символы (нетерм.) – из них нельзя вывести цепочку.

Алгоритм удаления:

$N_0 = \emptyset$

$N_i = \{A, A \to \alpha, \alpha \in \{N_{i-1}\} \cup \{T\}^*\} \cup \{N_{i-1}\}$

– на каждом шаге добавляем к множеству нетерминалы, из которых выводятся термин. символы или нетерминалы из нашего множества.

Пример: в конце удаляем все правила, содерж. символы, не вошедшие в результат. множество

$S \to AB \mid cA$
$A \to a$
$B \to BC \mid AB$
$C \to aB \mid b$
$D \to aC \mid bB.$

$N_0 = \emptyset$
$N_1 = \{A, C\}$ $(A \to a, C \to b)$
$N_2 = \{A, C, S, D\}$ $(S \to cA, D \to aC)$
$N_3 = \{A, C, S, D\}$ – нечего больше добавить

⇓

$S \to cA$
$A \to a$
$C \to b$
$D \to aC$

2. Недостижимые символы (удалять только после бесполезных!!)

$V_0 = S$

$V_i = \{X \mid A \to xXy, A \in \{V_{i-1}\}, x, y \in T^*\}$

– на каждом шаге добавляем нетерминалы, выводимые из нетерминалов текущего множества.

$S \to cA$
$A \to a$
$C \to b$
$D \to aC$

$V_0 = \{S\}$

$V_1 = \{S, A\}$ $S \to cA$

$V_2 = \{S, A\}$ - некого добавить

$\Downarrow$

$S \to cA$
$A \to a$

3. Удаление ε-правил
(чтобы осталось только $S \to \varepsilon$)

Пример:

$S \to ABB \mid BAA$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$

$\Rightarrow$

с учетом $B \to \varepsilon$ и $A \to \varepsilon$.

$ABB \to$	$BAA \to$
ABB	BAA
AB	BA
BB	AA
A	B
B	A
ε	ε

$\Downarrow$

$S \to ABB \mid BAA \mid AB \mid BA \mid BB \mid AA \mid A \mid B \mid \varepsilon$
$A \to a$
$B \to b$

---

1. Удаление левой рекурсии

$A \to A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_k$

$\Downarrow$

$A \to \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_k A'$

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$

Удаление произвольного уровня рекурсии - мне не совсем понятно, но есть пример для 2х:

$S \to Aa \mid b$
$A \to Ac \mid Sd$

$\Rightarrow$ 1) $S \to Aa \mid b$
$A \to Ac \mid Aad \mid bd$

2) удаление лев. рекурсии для A

$S \to Aa \mid b$
$A \to bdA'$
$A' \to cA' \mid adA' \mid \varepsilon$.

2. Удаление левой факторизации

$A \to \alpha\beta_1 \mid \dots \mid \alpha\beta_n \mid \dots \mid \gamma$

$A \to \alpha A' \mid \gamma \mid \dots$
$A' \to \beta_1 \mid \dots \mid \beta_n$.

7. Построение LL(1) - анализатора

1. Грамматика должна быть нелеворекурсивна и однозначна (если не так - удаляем левую рекурсию и факторизуем).

FIRST(α) – (α - строка) - символы, с которых может начинаться α (терминалы)

FOLLOW(α) - терм. символы, которые могут следовать за α в цепочках языка

Для грамматики должно выполн.:

$A \to \alpha \mid \beta$ – правило из грамматики

1) $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$

2) $\varepsilon \in FIRST(\alpha) \Rightarrow FIRST(\beta) \cap FOLLOW(A) = \emptyset$

2. Строим таблицы FIRST и FOLLOW для всех нетерминалов грамматики.

1) FIRST: $a \in T \Rightarrow FIRST(a) = \{a\}$

$X \to Y_1 Y_2 Y_3 \Rightarrow FIRST(Y_1) \in FIRST(X)$.

2) FOLLOW: $\# \in FOLLOW(S)$ S - начальный символ!

$A \to \alpha B \beta$, $\beta \in (N \cup T)^*$

$FIRST(\beta) \setminus \{\varepsilon\}$ добавл. в $FOLLOW(B)$

$A \to \alpha B$ или $A \to \alpha B \beta$, где $\beta \overset{*}{\Rightarrow} \varepsilon$

$\Rightarrow FOLLOW(A)$ добавл. в $FOLLOW(B)$.

Пример.

$S \to aS'$
$S' \to ABC \mid BC \mid \varepsilon$
$C \to BS' \mid S'$
$A \to aA'$
$A' \to a \mid b$
$B \to c$.

FIRST

S	S'	C	A	A'	B
a	a, b, ε	c, a, b, ε	a	a, b	c

$S \to aS' \Rightarrow FIRST(S) = a$ и только a

$S' \to ABC$, $A \to aA'$, $FIRST(A) = a$, $\Rightarrow$
$\Rightarrow a \in FIRST(S')$
$S' \to BC \Rightarrow b \in FIRST(S')$
$S' \to \varepsilon \Rightarrow \varepsilon \in FIRST(S')$

$FOLLOW(S) = \#$!

FOLLOW

S	S'	C	A	A'	B
#	#	#	b	b	b, #

$S \to aS' \Rightarrow FOLLOW(S)$ добавим в $FOLLOW(S')$

$S \to BS'$, $B \to c \Rightarrow$
$\Rightarrow c \in FIRST(C)$
$C \to S' \Rightarrow$
$FIRST(S') \in FIRST(C)$

$C \to BS' \Rightarrow$ добавим $FIRST(S')$ в $FOLLOW(B)$ + $S' \overset{*}{\Rightarrow} \varepsilon \Rightarrow FOLLOW(C)$ в $FOLLOW(B)$

$A \to aA'$, добавим $FOLLOW(A)$ в $FOLLOW(A')$

если только правило $S \to ABC$ (с A) $\Rightarrow$
$\Rightarrow FOLLOW(A) = b$.

3. Строим таблицу анализатора

Правило $A \to \alpha$. ячейка в таблице

1) $a \in FIRST(\alpha) \Rightarrow M[A, a] =$ номер правила
2) $\left.\begin{matrix} \varepsilon \in FIRST(\alpha) \\ b \in FOLLOW(A) \end{matrix}\right\} \Rightarrow M[A, b] =$ номер правила.

Пример: терминалы

нетерм.

	a	b	c	#
S	1			
S'	2	3		4
C	6	6	5	6
A	7			
A'	8	9		
B			10	

номера правил

1	$S \to aS'$
2,3,4	$S' \to ABC / BC / \varepsilon$
5,6	$C \to BS' / S'$
7	$A \to aA'$
8,9	$A' \to a / b$
10	$B \to c$

1: $S \to aS'$. $FIRST(aS') = a$
2: $S' \to ABC$. $FIRST(ABC) = a$
3: $S' \to BC$. $FIRST(BC) = b$
4: $S' \to \varepsilon$. $FIRST(\varepsilon) = \varepsilon$
$FOLLOW(S') = \#$

4. Анализируем (разбираем) строку.

Начало:

магазин	
$S\#$	$a_1 \dots a_n \#$

нач. симв. — строка.

Пусть A - первый символ в магазине,
a - первый символ строки.

1) $A = a = \# \to$ Accept (ура=) строка разобрана)
2) $A = a \neq \# \to$ убираем A и a.
3) $A \in T$ (терминал), $A \neq a$ — ошибка! строка недопустима
4) $A \in N$: правило $M[A, a]$ примен. к магазину слева

Пример:

$S\#$	$aabb\#$	$[S,a]=1: S \to aS'$
$aS'\#$	$aabb\#$	$a=a$ – убираем
$S'\#$	$abb\#$	$[S',a]=2: S' \to AbC$
$AbC\#$	$abb\#$	$[A,a]=7: A \to aA'$
$aA'bC\#$	$abb\#$	$a=a$ – убираем
$A'bC\#$	$bb\#$	$[A',b]=9: A' \to b$
$bbC\#$	$bb\#$	$b=b$, $b=b$, убираем 2 раза
$C\#$	$\#$	$[C,\#]=6: C \to S'$
$S'\#$	$\#$	$[S',\#]=4: S' \to e$
$\#$	$\#$	Accept

Применяя к S все правила сверху вниз, можно получить лев. цепочку

8. Построение LR(1) анализатора по КС-грамматике.

1. Пополнение грамматики.
добавим правило $S' \to S$, где S – наш начальный символ.

2. LR-1 ситуации:

$[A \to \alpha . B\beta, a]$, $A \to \alpha B\beta$ – правило грамм., a – терминал

начальная ситуация: $[S \to . S, \#]$.

3. Применение замыкания к множ-ву: (I)
a – терминал сразу после.

$\forall [A \to \alpha . B\beta, a] \in I$
$\forall B \to \gamma$
$\forall b \in FIRST(\beta a)$ $\Big\}$ $\to$

в I добавляем правила вида:
$[B \to . \gamma, b]$

4. Переход к новому множеству:
в I: $A \to \alpha . B\beta, b$ $\Rightarrow$
в I': $A \to \alpha B . \beta, b$.

5. Строим множества.

Пример:

пополним грамматику!

$A' \to A\$$

$A \to BS$

$B \to Aa \mid e$

$S \to Sb \mid b$

I_0: $A' \to .A\$, \#$ ← начальная ситуация.

замыкание:

есть $A \to BS$, $FIRST(\#) = \#$ → $A \to .BS, \#$

можно сжать в

есть $B \to Aa$, $FIRST(S\#) = b$ → $B \to .Aa, b$

есть $B \to e$, $FIRST(S\#) = b$ → $B \to ., b$

$A \to .BS, a\#$ но не стоит если пытаемся

есть $A \to BS$, $FIRST(ab) = a$ → $A \to .BS, a$

больше ничего нового.

A

(взяли все правила с буквой A после . и сдвинули ее за .)

I_1: $A' \to A., \#$; $B \to A.a, b$

замыкание — нечего добавить, нет терминалов сразу за.

a → I_3: $B \to Aa., b$

B

I_2:

$A \to B.S, \#$

$A \to B.S, a$

замыкание:

$S \to .Sb, \#$

$S \to .Sb, a$

$S \to .Sb, b$

$S \to .b, a\ b\ \#$

b → I_4: $S \to b., \#\ a\ b$

S → I_5: $S \to S.b, \#\ a\ b$; $A \to BS., \#a$

b → I_6: $S \to Sb., ab\#$

больше нечего двигать за .

здесь 3 ситуации! мы их склеили, чтобы меньше писать

В процессе могут получаться одинаковые множества, у них должен быть один номер, не нужно разделять их на несколько

6. Строим таблицу

N - нетерминал

1) $I_i \to I_j$

добавим в таблицу $Goto(I_i, N) = I_j$

2) $I_i \xrightarrow{a \in T} I_j$

$Action(I_i, a) =$ ~~Shift~~ Shift I_j

3) в I_i есть ситуация $A \to \alpha . \beta$

$Action(I_i, \beta) =$ Reduce $A \to \alpha$ → номер правила

4) Reduce $S' \to S$ = Accept

5) все остальное - Error.

Пример:

	a	b	#	A	B	S
0		R3		1	2	
1	S3		Acc			
2		S4			5	
3		R2				
4	R5	R5	R5			
5	R1	S6	R1			
6	R4	R4	R4			

Action | Goto

номера сост-в I

терминалы | нетерм.

0	$A' \to A$
1	$A \to BS$
2, 3	$B \to Aa \mid e$
4, 5	$S \to Sb \mid b$

I_0:
- $A \to . A, \#$
- $A \to . BS, \# a$
- $B \to . Aa, b$
- $B \to ., b$

3: $B \to e$

A → I_1:
- $A' \to A ., \#$
- $B \to A . a, b$

a → I_3

B → I_2

7. Разбор цепочки.

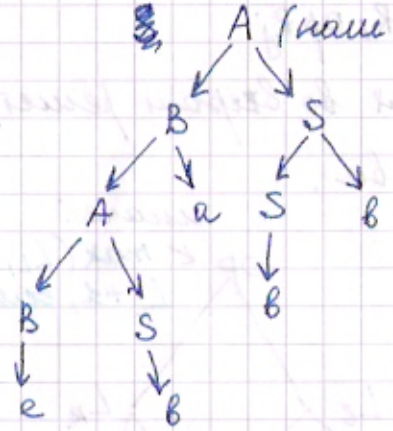
сразу на примере.

начальное состояние - 0 | $a_1 \dots a_n \#$

маш. | строка

	пустой симв.		
0ε	babb #	[0, B] = R3.	Reduce - выполняем правило в обр. стор.
0B2		[0,B] = 2	всегда посмотрим в магазине - цифра!
		3: B → e	
0B2	babb #	[2, b] = S4.	Shift - переносим символ в магазин, ставим цифру, кот. после S.
0B2b4	abb #	[4, a] = R5	
		5: S → b - ц в ромашке	
0B2S5	abb #	[5, a] = R1	сделать S; [2,S]=5
		1: A → BS, [0,A] = 1	
0A1	abb #	[1, a] = S3	
0A1a3	bb #	[3, b] = R2	2: B → Aa, [0,B] = 2
0B2	bb #	[2, b] = S4	
0B2b4	b #	[4, b] = R5	5: S → b [2,S] = 5
0B2S5	b #	[5, b] = S6	
0B2S5b6	#	[6, #] = R4	4: S → Sb [2,S] = 5
0B2S5	#	[5, #] = R1	1: A → BS [0,A] = 1
0A1	#	[1, #] = Acc	

применяя правила R... снизу вверх к S, можем получить нек. цепочку



8. Определение, является ли грамматика LR(0) по таблице её LR(1) анализатора.

1) в LR(0) Shift и Reduce не могут встречаться в одной строке.

S1	R3	S2

– не LR(0)!

2) в LR(0) для Reduce в одной строке должно быть одно и то же правило!

R3	R4	R5

– не LR(0)!

S1 R1	S2 R1	S3 R1

– можно

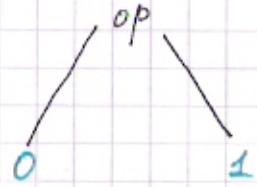
9. Трансляция арифметических выражений

op R_i, R_j $R_j := R_i$ op R_j

(рез-т сохраняется во втором регистре)

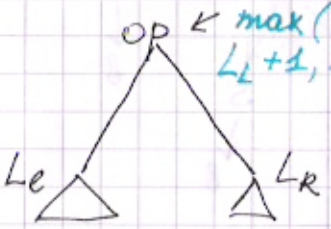
① Разметка дерева.

~~Если~~ листья:



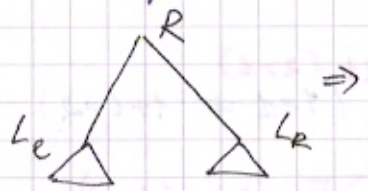
иначе:

← $\max(L_L, L_R)$ или $L_L + 1$, если $L_L = L_R$

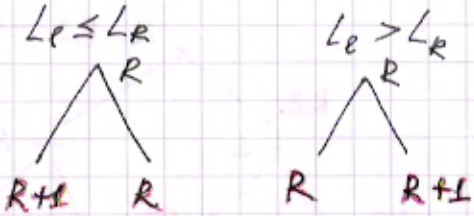


②. Распределение регистров.

корень – R1

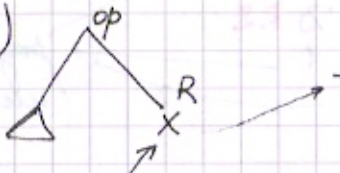


$\Rightarrow$



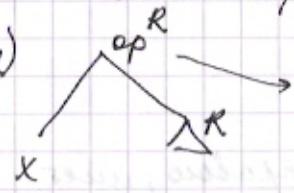
= при = приоритет правому

3. Генерация кода

1) 

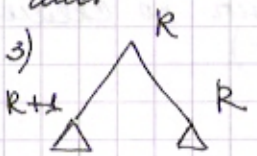
→ move X, R

имеет с перем.

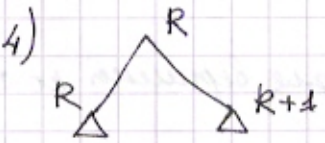
2) 

→ код для вершины op X, R.

имеет

3) 

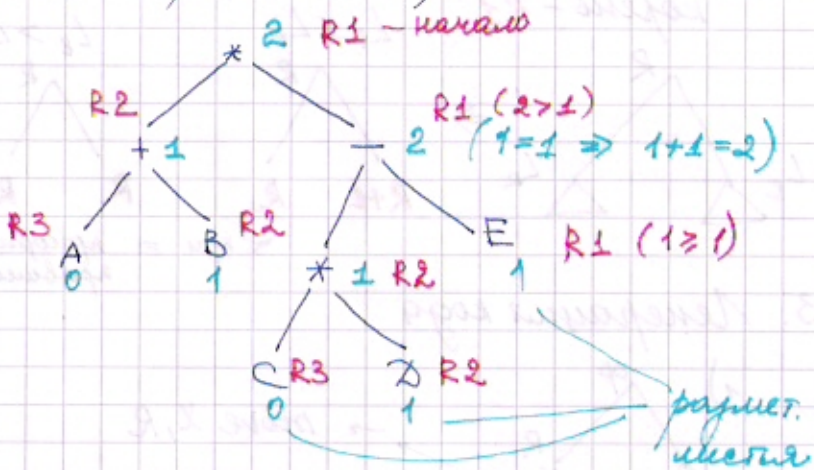
→ $L_L \leq L_R$
код правого
код левого
op R+1, R

4) 

$L_L > L_R$
код левого
код правого
op R, R+1
move R+1, R.

Пример:

~~A*B~~ (A+B)*(C*D-E)



Код:

- начинаем с корня.
- 2>1 ⇒ код правого.
  - 1≥1 – код правого
    - move E, R1
    - код левого
      - 1≥0 – код правого, мест
        - move D, R2
        - код для вершины по схеме 2) (слева – мест)
        - mul C, R2
    - sub R2, R1
  - код левого
    - 1≥0, мест справа
    - move B, R2
    - слева мест ⇒ код для вершины по сх. 2)
    - add A, R2
- mul R2, R1

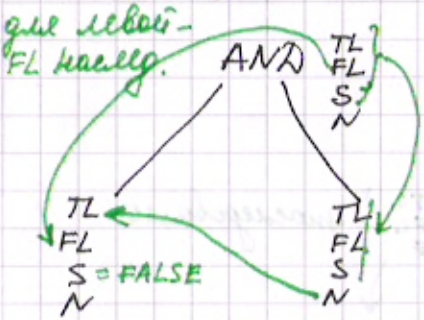
10. Трансляция логических выражений.

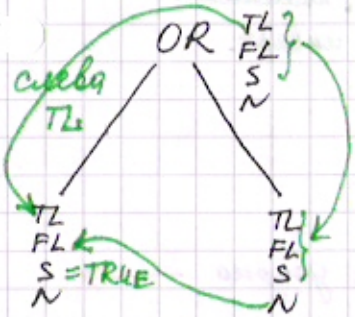
1. У каждой вершины метки:

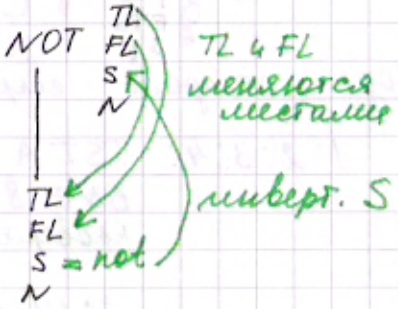
TL (truelabel)
FL (falselabel)
S
N - номер вершины. (нумеруем как угодно, главное - нет одинаковых).

У корня: TL = TRUELAB
FL = FALSELAB
S = FALSE
N = номер

2. Расставляем метки вершинам:







3. Генерация кода.

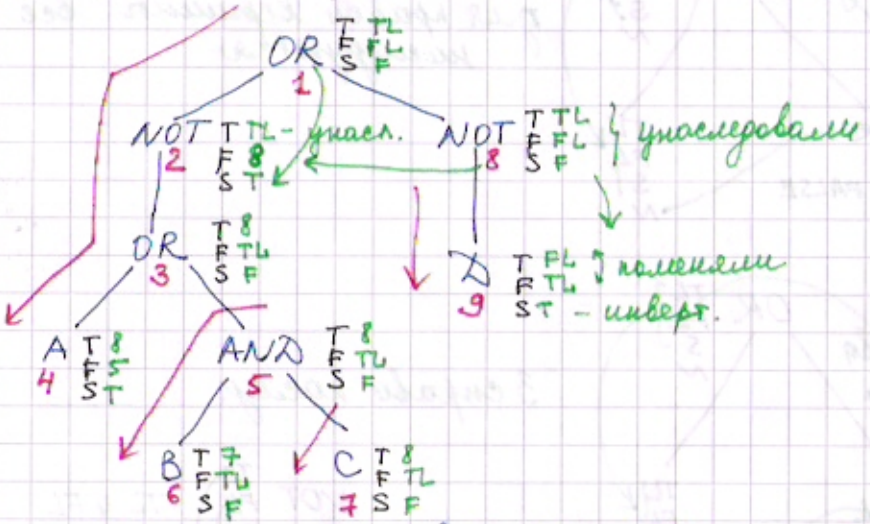
1) ставим при обходе метки-номера пройденных вершин

2) если вершина - лист, то
TST A - лог. перем. листа
BNE TL, если S=TRUE — переход, если 1
BEQ FL, если S=FALSE — переход, если 0

3) в конце
TRUELAB:
FALSELAB:

Пример:

not (A or B and C) or not D.



Код: обходим сверху вниз как удобно →

1:2:3:4: TST A
BNE 8 - S=TRUE, TL=8.
проверяем: если A=1, то проверить not D
если A=1, то A or B and C=1 =>
not (A or B and C)=0.

5:6: TST B
BEQ TRUELAB S=FALSE, FL=TRUELAB.
проверяем: если B=0 (A уже = 0, если сюда дошли),
то (A or B and C) = 0 ⇒
not (A or B and C) = 1 ⇒ все выражение - 1.

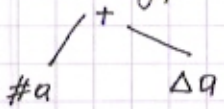
7: TST C
BEQ TRUELAB – аналогично

8:9: TST D
BNE FALSELAB – S=TRUE, TL=FALSELAB
проверяем: дошли сюда, значит
левая часть = 0. ~~Если~~ если D=1, то
not D=0 и все выр-е 0.

TRUELAB:
FALSELAB:

11. Генерация оптимального кода методом сопоставления образцов.

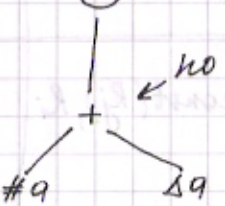
1. a – адрес (a = ...)



– адрес + смещение

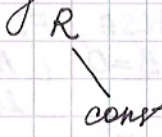
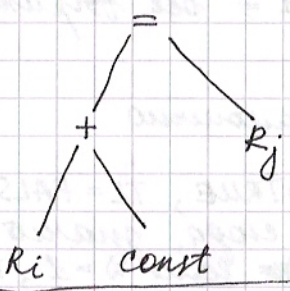
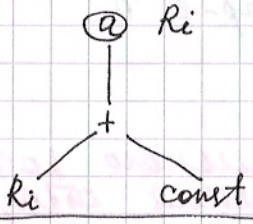
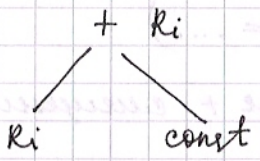
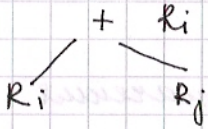
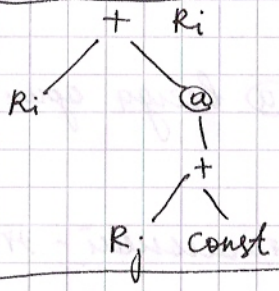
a – значение

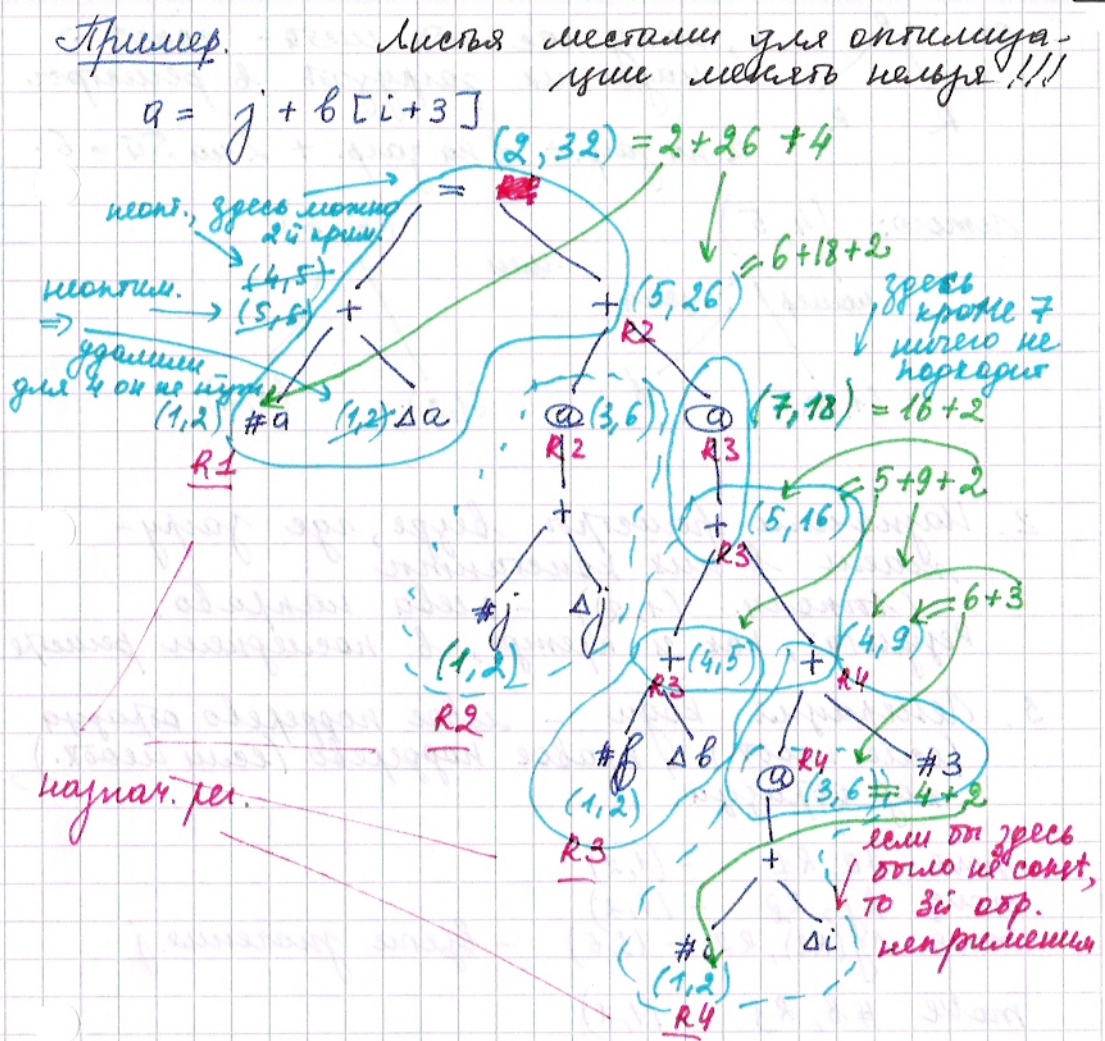
@ – взятие значения по адресу



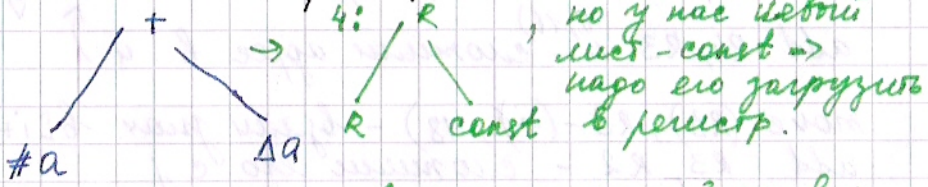
у @ всегда один потомок!

2. Образцы (на контрольной – эти, на эк. могут быть и другие)

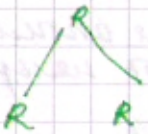
N	вид	код	цена
1		move #const, R загрузка константы в регистр	2
2.		move Rj, const(Ri)	4
3.		move const(Ri), Ri	4
4.		ADD #const, Ri	3
5.		ADD Rj, Ri	2
6.		ADD const(Rj), Ri	4
7.		move (Ri), Ri	2.



1. Спуск вверх слева направо (или справа налево? в общем как-то) будем смотреть, какие образцы можно применить. например:

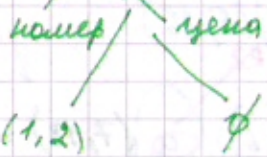


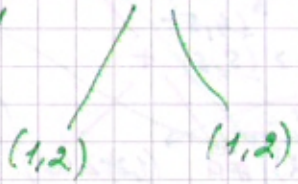
2 на загрузку + 3 на вып.
4-го образца = 5.

5: 

, но у нас оба листа - сопst =>
надо их загрузить в регистры

2 на загр. + 2 на загр. + 2 на 5й = 6.

итого: (4, 5)


или (5, 6)


2. Назначим регистры везде, где загружаем в них константы (выполн. (1,2)) - слева направо.
Результат, как и прежде, в последнем регистре.

3. Генерация кода - левое поддерево образуя (если необх.) ; правое поддерево (если необх.), код образуя.

```
move #a, R1     - (1,2)
move #j, R2     - (1,2)
move Δj(R2), R2 - (3,6)  - взяли значение j

move #b, R3     - (1,2)
add  Δb, R3     - (4,5)
move #i, R4     - (1,2)
move Δi(R4), R4 - (3,6) - взяли знач. i
add  3, R4      - (4,9) - прибавили к нему 3
add  R4, R3     - (5,16) - сложили адрес в ч
move (R3), R3   - (7,18) - взяли знач. b[i+3]
add  R3, R2     - сложили его с j
move R2, Δa(R1) - загрузили полученное в a.
```