1. Преобразование недетерминированного автомата в детерминированный.

Метод состоит в построении взаимно-однозначного соответствия состояниям ДКА подмножества состояний НКА.

Алгоритм преобразования НКА в ДКА. Будем называть состояния НКА вершинами, состояния ДКА состояниями.

0) «Чистка» НКА (не обязательное действие, но сокращающее трудоёмкость алгоритма детерминизации). Из НКА удаляются: – петли без меток; – вершины, недостижимые из начальных вершин; – вершины, из которых недостижимы заключительные вершины.

1) Таблица НКА. В таблице должны быть следующие столбцы: – имя вершины, – индекс вершины, – индикатор (флаг) финальной вершины, остальные столбцы соответствуют всем символам входного алфавита плюс пустой символ. Первая строка таблицы соответствует единственной входной вершине. Если входных вершин было более одной, то это будет новая вершина, которая соединяется дугами без меток (пустой меткой) со всеми старыми входными вершинами. Остальные строки соответствуют всем вершинам НКА.

2) Нумерация дуг (переходов). Каждый помеченный переход получает уникальный номер, начиная с 2 и далее 3,4,...

3) Индексация вершин. Входная вершина получает индекс 1. Вершинам НКА присваивается индекс, состоящий из номеров всех входящих в эту вершину дуг. Индекс вершины, из которой выходит пустая дуга, приписывается к индексу вершины, в которую эта дуга входит.

4) Автоматная таблица. Заготавливается таблица (пустая), в которой должны быть следующие столбцы: – шифр состояния, – флаг финальных вершин, – остальные столбцы соответствуют всем символам входного алфавита.

5) Заполнение строки автоматной таблицы. Первой строке, соответствующей начальному состоянию автомата, присваивается шифр 1. Начиная с этой строки, используя таблицу источника, в столбцы, именованные входными символами, записываются номера дуг, имеющих метку, совпадающую с именем столбца и выходящих из всех вершин, в индексах которых содержится какой-либо номер из шифра этого состояния (строки). Если таких дуг нет, то записывается номер 0. В строке с шифром 0 во всех столбцах содержатся нули. (Состоянию с шифром 0 соответствует “тупиковое” состояние.)

6) Порождение новых строк. После заполнения очередной строки в столбцах, именованных входными символами, записаны шифры состояний. Если в строке появились новые шифры, не встречавшиеся ранее, то они порождают новые строки.

7) Финальными будут те состояния, в шифре которых есть номер, совпадающий с каким-либо номером из индексов финальных вершин НКА.

Пример: (см. Ахо с.208, рис.3.34)

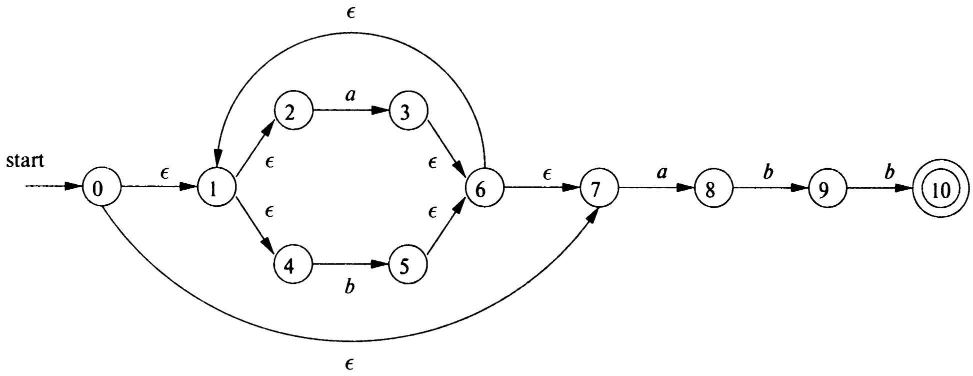

Рис. 1.

Вершина НКА	a	b	ε	индекс
(0)			(1) (7)	1
(1)			(2) (4)	123
(2)	(3)**2**			123
(3)			(6)	2
(4)		(5)**3**		123
(5)			(6)	3
(6)			(1) (7)	23
(7)	(8)**4**			123
(8)		(9)**5**		4
(9)		(10)**6**		5
(10)				**6**

Шифр состояния	Fl	a	b
1 (A)		24	3
24 (B)		24	35
3 (C)		24	3
35 (D)		24	36
3**6 (E)**	1	24	3

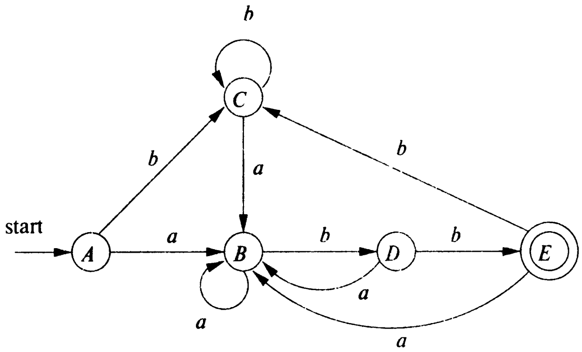

Рис. 2.

2. Минимизация детерминированного автомата.

1) Явно эквивалентные состояния заменяются одним состоянием.

2) Вычисление S_0. Состояниям с одинаковыми флагами присваивается один и тот же суффикс. Автоматная таблица переписывается с записью суффиксов в строке состояний.

3) Вычисление S_i. Состояниям с одинаковой комбинацией в строке выхода и суффиксов присваивается один и тот же может быть новый суффикс.

Если количество различных суффиксов, т.е. классов эквивалентности не увеличилось или равно числу состояний, то вычисление классов эквивалентности закончено, иначе пункт (3).

4) За не более чем n-1 итерацию классы эквивалентности вычислены. Каждому классу эквивалентности в минимальном автомате сопоставляется одно состояние.

Примеры

Q	**Fl**	**a**	**b**	**S_0**
~~A~~	~~0~~	~~B~~	~~C~~	
B	0	B	D	2
C	0	B	C	2
D	0	B	E	2
E	1	B	C	1

Fl	**a**	**b**	**S_1**
0	B-2	D-2	21
0	B-2	C-2	21
0	B-2	E-1	22
1			1

Fl	**a**	**b**	**S_2**
0	B-21	D-22	211
0	B-21	C-21	212
0			22
1			1

Комментарий: Состояния A и C явно эквивалентные

3. Синтез недетерминированного LR-автомата по заданной КС-грамматике.

Недетерминированный автомат (LRAN)

Состояниями LRAN сопоставляются *пункты* продукций. Каждый из пунктов это позиция в продукции, отсекающая ту часть продукции, которая могла быть сопоставлена уже просмотренной части входной строки. Пункт обозначим метасимволом «•».

Переходы в LRAN

1) из состояния $A\to\alpha\bullet\theta\gamma$ в состояние $A\to\alpha\theta\bullet\gamma$, переход инициируется символом $\theta\in T\cup N$, $\alpha,\gamma\in(T\cup N)^*$;

2) переходы по пустому символу ε из состояния $A\to\alpha\bullet B\gamma$, в состояние $B\to\bullet\beta$, где B нетерминал, $\alpha,\beta,\gamma\in(T\cup N)^*$.

Последний из возможных пунктов в продукции ($A\to\omega\bullet$) означает, что часть строки может быть сопоставлена телу продукции ω и эта часть строки должна быть *свёрнута* в нетерминал A. Такие пункты соответствуют *квазисостояниям*, так как они не являются источниками переходов. Состояние в LRAN идентифицируется парой натуральных чисел (X,Y), где X – позиция пункта в продукции (начиная с 0), Y – номер продукции.

Пример

(0) $S\to A\perp$

(1) $A\to E$

(2) $A\to +E$

(3) $A\to A+E$

(4) $A\to A\times E$

(5) $E\to id$

(6) $E\to (A)$

Состояния и переходы в LRAN

(0,0) S→•A⊥	A→	S→A•⊥	(1,0)
	ε→	A→•E	(0,1)
	ε→	A→•+E	(0,2)
	ε→	A→•A+E	(0,3)
	ε→	A→•A×E	(0,4)
(0,1) A→•E	E→	A→E•	($\boxed{1,1}$)
	ε→	E→•id	(0,5)
	ε→	E→• (A)	(0,6)
(0,2) A→•+E	+→	A→+•E	(1,2)
(0,3) A→•A+E	A→	A→A•+E	(1,3)
	ε→	A→•E	(0,1)
	ε→	A→•+E	(0,2)
	переход в (0,3) – «в себя» может быть пропущен		
	ε→	A→•A×E	(0,4)
(0,4) A→•A×E	A→	A→A•×E	(1,4)
	ε→	A→•E	(0,1)
	ε→	A→•+E	(0,2)
	ε→	A→•A+E	(0,3)
	переход в (0,4) – «в себя» может быть пропущен		
(0,5) E→•id	id→	E→id•	($\boxed{1,5}$)
(0,6) E→•(A)	(→	E→(•A)	(1,6)
(1,0) S→A•⊥	⊥→	S→A⊥•	$\boxed{(2,0)}$
(1,2) A→+•E	E→	A→+E•	($\boxed{2,2}$)
	ε→	E→•id	(0,5)
	ε→	E→•(A)	(0,6)
(1,3) A→A•+E	+→	A→A+•E	(2,3)
(1,4) A→A•×E	×→	A→A×•E	(2,4)
(1,6) E→(•A)	A→	E→(A•)	(2,6)
	ε→	A→•E	(0,1)
	ε→	A→•+E	(0,2)
	ε→	A→•A+E	(0,3)
	ε→	A→•A×E	(0,4)
(2,3) A→A+•E	E→	A→A+E•	($\boxed{3,3}$)
	ε→	E→•id	(0,5)

	ε→	E→• (A)	(0,6)
(2,4) A→A×•E	E→	A→A×E•	($\boxed{3,4}$)
	ε→	E→•id	(0,5)
	ε→	E→•(A)	(0,6)
(2,6) E→(A•)	E→	E→(A)•	($\boxed{3,6}$)

Таблица недетерминированного автомата

(xy)	**id**	**+**	**×**	**(**	**)**	**⊥**	**A**	**E**	**ε**	**индекс**
00							*10***2**		*01–04*	**1**
01								*11***3**	*05,06*	**1,8**
02		*12***4**								**1,8**
03							*13***5**		*01,02,04*	**1,8**
04							*14***6**		*01,02,03*	**1,8**
05	*15***7**									**1,4,8,11,12**
06				*16***8**						**1,4,8,11,12**
10						*20***9**				**2**
11										**3**
12								*22***10**	*05,06*	**4**
13		*23***11**								**5**
14			*24***12**							**6**
15										**7**
16							*26***13**		*01–04*	**8**
20										**9**
22										**10**
23								*33***14**	*05,06*	**11**
24								*34***15**	*05,06*	**12**
26					*36***16**					**13**
33										**14**
34										**15**
36										**16**

4. Детерминизация LR-автомата.

Детерминированный автомат (LRAD)

Состояниям LRAD соответствуют подмножества пунктов продукций и соответственно подмножества состояний LRAN. Строго говоря, LRAD не соответствует точно определению детерминированного автомата, но неоднозначности в переходах автомата нет.

Далее следуя алгоритму 1. Преобразование недетерминированного автомата в детерминированный, из предыдущей таблицы получим таблицу детерминированного автомата. Определим замену шифра номером S.

S	шифр	сост-ние	id	+	×	(	)	⊥	A	E
1	*1*	0∀	*7*/**2**	*4*/**3**		*8*/**4**			*2,5,6*/**5**	*3*/**6**
2	*7*	15								
3	*4*	12,05,06	*7*/**2**			*8*/**4**				*10*/**7**
4	*8*	16,0X	*7*/**2**	*4*/**3**		*8*/**4**			*5,6,13*/**8**	*3*/**6**
5	*2,5,6*	10,13,14		*11*/**9**	*12*/**10**			*9*/**11**		
6	*3*	11								
7	*10*	22								
8	*5,6,13*	13,14,26		*11*/**9**	*12*/**10**		*16*/**12**			
9	*11*	23,05,06	*7*/**2**			*8*/**4**				*14*/**13**
10	*12*	24,05,06	*7*/**2**			*8*/**4**				*15*/**14**
11	*9*	20								
12	*16*	36								
13	*14*	33								
14	*15*	34								

Причёсанный вариант

S	состояние	id	+	×	(	)	⊥	A	E
1	0∀	2	3	0	4	0	0	5	6
②	15	(5) ⟨E→id⟩							
3	12,05,06	2	0	0	4	0	0		7
4	16,0X	2	3	0	4	0	0	8	6
5	10,13,14	0	9	10	0	0	11		
⑥	**11**	(1) ⟨A→E⟩							
⑦	**22**	(2) ⟨A→+E⟩							
8	13,14,26	0	9	10	0	12	0		
9	23,05,06	2	0	0	4	0	0		13
10	24,05,06	2	0	0	4	0	0		14
⑪	**20**	(0) ⟨S→A⊥⟩, финиш							
⑫	**36**	(6) ⟨E→(A)⟩							
⑬	**33**	(3) ⟨A→A+E⟩							
⑭	**34**	(4) ⟨A→A×E⟩							
0		Обработка ошибки							

0∀ – все 0-пункты, 0X – все 0-пункты, кроме (00).

5. Использование LR-автомата в синтаксическом разборе.

В процессе синтаксического анализа вместе с LRAD используются два стека:
SS – стек состояний,
SC – стек символов.
Пусть:
ξ – символ на вершине стека символов SC,
s_i – номер состояния на вершине стека состояний SS,
x – терминальный символ анализируемой строки.

Действия:

$s_1 \underline{\ \xi\ } s_2$, SC←*x* − переход из состояния s_1 в состояние s_2, запись состояния s_2 в стек состояний (SS), запись очередного терминального символа *x* анализируемой строки в стек символов (SC).

$s_1 \underline{\ \xi\ }$ ①, ⟨A→ω⟩ − переход из состояния s_1 в квазисостояние ①, которое в стек не записывается, но вызывает свёртку продукции A→ω к нетерминалу A. Где ω часть (суффикс) строки, расположенной на вершине стека символов (SC), заменяется нетерминалом A.

Количество номеров состояний в стеке состояний (SS) всегда равно количеству символов в стеке символов (SC). Лишние номера состояний удаляются согласно дисциплине стека.

Пример. Разбор строки: id×(id+id)

№	действия	SS	SC	строка
1	__1, SC←id	.1	.id	×(id+id) ⊥
2	1 id ②, ⟨E→id⟩	.1	.E	×(id+id) ⊥
3	1 E ⑥, ⟨A→E⟩	.1	.A	×(id+id) ⊥
4	1 A 5, SC←×	.1,5	.A×	(id+id) ⊥
5	5 × 10, SC←(	.1,5,10	.A×(	id+id) ⊥
6	10 (4, SC←id	.1,5,10,4	.A×(id	+id) ⊥
7	4 id ②, ⟨E→id⟩	.1,5,10,4	.A×(E	+id) ⊥
8	4 E ⑥, ⟨A→E⟩	.1,5,10,4	.A×(A	+id) ⊥
9	4 A 8, SC←+	.1,5,10,4,8	.A×(A+	id) ⊥
10	8 + 9, SC←id	.1,5,10,4,8,9	.A×(A+id	) ⊥
11	9 id ②, ⟨E→id⟩	.1,5,10,4,8,9	.A×(A+E	) ⊥
12	9 E ⑬, ⟨A→A+E⟩	.1,5,10,4	.A×(A	) ⊥
13	4 A 8, SC←)	.1,5,10,4,8	.A×(A)	⊥
14	8) ⑫, ⟨E→(A)⟩	.1,5,10	.A×E	⊥
15	10 E,⑭ ⟨A→A×E⟩	.1	.A	⊥
16	1 A 5, SC←⊥	.1,5	.A⊥	
17	5 ⊥ ⑪, ⟨S→A⊥⟩	.1	.S	

Резюме:

После того как определён LRAD, который будет участвовать в обработке синтаксического объекта, выполняется алгоритм:

Вход: список продукций, таблица LRAD, строка токенов (в дальнейшем называемых символами) синтаксического объекта.

Начало: запись в стек состояний SS номера 1, запись в стек символов SC начального символа из входной строки.

Итерация:

- Прочесть (без выталкивания) с вершин стеков SS − номер состояния *s*, SC – символ *x*.
- Если в стеке символов SC – один стартовый нетерминал, то завершить обработку синтаксического объекта. Иначе
- Обратиться к таблице <u>LRAD</u> с тем, чтобы узнать номер и тип следующего состояния $s_i=\delta(s,x)$.

– Если s_i состояние ошибки, то завершить обработку объекта.

– Если s_i квазисостояние, то выполнить сворачивание продукции: из стека символов SC убрать (POP) k символов, заменив (PUSH) их нетерминальным символом продукции, из стека состояний SS убрать (POP) k−1 номеров состояний.

– Если s_i состояние автомата «обычное», то записать (PUSH) его номер s_i в стек состояний SS, записать (PUSH) в стек символов SC очередной терминальный символ входной строки.

LR синтаксический анализ с LRAD автоматом, у которого только состояния и квазисостояния называют LR(0) анализом.

Ни один из методов разбора не распознает все возможные КС-грамматики.

Неоднозначные грамматики

Определение: КС-грамматика называется *неоднозначной*, если существует слово, которое имеет два или более правосторонних (левосторонних) вывода. В противном случае КС-грамматика называется *однозначной*.

Пример: Грамматика (слияние строк):

A→A+A|**d** – неоднозначная. Существует более одного варианта правого вывода цепочки **d+d+d**.

6.2.8. Проблема однозначности: *не существует алгоритма*, позволяющего по произвольной КС-грамматике G над алфавитом T (|T|≥2) узнать, является ли грамматика G однозначной.

6.2.9. Применяя LR-анализ к неоднозначным КС-грамматикам, получаем LRAD, у которого в некоторых состояниях возможны оба действия: или чтение (и возможно запись) терминального символа входной строки, или сворачивание продукции, назовём их *полусостояниями*. Потребуется дополнительный анализ грамматики, после которого возможны следующие решения:

– полусостояние определяется только как состояние;

– полусостояние определяется только как квазисостояние;

– в полусостоянии возможны оба действия, в таком случае применяется LR(1) анализ, при котором в полусостояниях читается очередной терминал входной строки и в зависимости от его значения принимается один из альтернативных вариантов, либо полусостояние интерпретируется как состояние (запись терминала в стек символов и запись состояния в стек состояний), либо – интерпретируется как квазисостояние (сворачивание продукции).

Условный оператор

Оператор ::=**if** Предикат **then** Оператор **else** Оператор
| **if** Предикат **then** Оператор
| ОператорДругой

(В Си вместо **then** будет **if** (Предикат) Оператор …)

Пусть A ⇌ Оператор
B ⇌ Предикат
d ⇌ ОператорДругой

Грамматика: A→ **if** B **then** A **else** A | **if** B **then** A | **d**

Пусть: **i** ⇌ **if** B **then**
e ⇌ **else**

(0) S→A⊥
(1) A→ **i** A **e** A
(2) A→ **i** A
(3) A→ **d**

if B **then** **if** B **then** **d** **else** **d**

i i d e d

Правый вывод (неоднозначный)

A ⇒ **i** A ⇒ **i i** A **e** A ⇒ **i i** A **e d** ⇒ **i i d e d**
A ⇒ **i** A **e** A ⇒ **i** A **e d** ⇒ **i i** A **e d** ⇒ **i i d e d**

(00) S→•A⊥	A→	S→A•⊥	(10)
	ε→		(01)(02)(03)
(01) A → •**i**A**e**A	**i**→	A → **i**•A**e**A	(11)
(02) A → •**i**A	**i**→	A → **i**•A	(12)
(03) A → •**d**	**d**→	A → **d** •	($\boxed{13}$)
(10) S→A•⊥	⊥→	S→A⊥•	($\boxed{20}$)
(11) A → **i**•A**e**A	A→	A → **i**A•**e**A	(21)
	ε→		(01)(02)(03)

(12) A → **i**•A	A→	A → **i**A•	($\boxed{22}$)
	ε→		(01)(02)(03)
(21) A → **i**A•**e**A	**e**→	A → **i**A**e**•A	(31)
(31) A → **i**A**e**•A	A→	A → **i**A**e**A•	($\boxed{41}$)
	ε→		(01)(02)(03)

(xy)	i	e	d	⊥	A	ε	index
00					*10***2**	*01,02,03*	**1**
01	*11***3**						**1,3,4,9**
02	*12***4**						**1,3,4,9**
03			*13***5**				**1,3,4,9**
10				*20***6**			**2**
11					*21***7**	*01,02,03*	**3**
12					*22***8**	*01,02,03*	**4**
13							**5**
20							**6**
21		*31***9**					**7**
22							**8**
31					*41***10**	*01,02,03*	**9**
41							**10**

S	шифр	состояние	i	e	d	⊥	A	
1	*1*	0∀	*3,4*/**2**	0	*5*/**3**	0	*2*/**4**	
2	*3,4*	01-03,11,12	*3,4*/**2**	0	*5*/**3**	0	*7,8*/**5**	
3	*5*	13	(3) ⟨A→ **d**⟩					
4	*2*	10	0	0	0	*6*/**6**		
5	*7,8*	21,22	0	*9*/**7**	0	0		(2) ⟨A→ **i** A⟩
6	*6*	20	(0) ⟨S→A⊥⟩					
7	*9*	0X,31	*3,4*/**2**	0	*5*/**3**	0	*10*/**8**	
8	*10*	41	(1) ⟨A→ **i** A **e** A⟩					

S	i	e	d	⊥	A	
1	2	0	3	0	4	
2	2	0	3	0	5	
③	(3) ⟨A→ **d**⟩					
4	0	0	0	6		
(5)	0	7	0	0		(2) ⟨A→ **i** A⟩
⑥	(0) ⟨S→A⊥⟩ финиш					
7	2	0	3	0	8	
⑧	(1) ⟨A→ **i** A **e** A⟩					
0	Обработка ошибки					

Разбор строки: **i i d e d** ⊥

№	действия	SS	SC	строка
1	__1, SC←i	.1	.i	i d e d ⊥
2	1 **i** 2, SC←i	.12	.i i	d e d ⊥
3	2 **i** 2, SC←d	.122	.i i d	e d ⊥
4	2 **d** ③, ⟨A→ **d**⟩	.122	.i i A	e d ⊥
5	2 A **(5)**, (e), SC←e	.1225	.i i A e	d ⊥

6	5 **e** 7, SC←d	.12257	.i i A e d	⊥
7	7 **d** 3, ⟨A→ **d**⟩	.12257	.i i A e A	⊥
8	7 A 8, ⟨A→ **i** A **e** A⟩	.12	.i A	⊥
9	2 A **(5)**, (⊥) ⟨A→ **i** A⟩	.1	.A	⊥
10	1 A 4, SC←⊥	.14	.A ⊥	
11	4 ⊥ 6, ⟨S→A⊥⟩	.1	.S	

В полусостоянии **(5)** анализируется терминал входной строки (см №5, №9).

6. Создать обратную польскую запись для арифметической формулы.
Вычислить значение, используя обратную польскую запись.

7. Создать обратную польскую запись для логической формулы с отношениями.
Вычислить значение, используя обратную польскую запись.

Алгоритм вычисления выражений, записанных в PN

1) Выражение просматривается один раз слева направо и для каждого элемента выполняются шаги (2) или (3);

2) Если очередной элемент PN — операнд, то его значение заносится в стек;

3) Если очередной элемент PN — операция, то на вершине стека сейчас находятся ее операнды (это следует из определения PN и предшествующих действий алгоритма); они извлекаются из стека, над ними выполняется операция, результат снова заносится в стек;

4) Когда выражение, записанное в PN, прочитано, в стеке останется один элемент — это значение всего выражения, т. е. результат вычисления.

Элемент	Вычисление	Стек
a		a
b		ab
c		abc
+	bc+ $\Rightarrow \rho_1$	aρ_1
×	aρ_1× $\Rightarrow \rho_2$	ρ_2
d		ρ_2d
e		$\rho_2$de
–	de– $\Rightarrow \rho_3$	$\rho_2\rho_3$
f		$\rho_2\rho_3$f
/	ρ_3f/ $\Rightarrow \rho_4$	$\rho_2\rho_4$
–	$\rho_2\rho_4$– $\Rightarrow \rho_5$	ρ_5

Замечание

1. Для интерпретации, кроме PN выражения, необходима дополнительная информация об операндах (например, о типах), хранящаяся в таблицах.

2. Может оказаться так, что знак бинарной операции по написанию совпадает со знаком унарной; например, знак «–» в большинстве языков программирования означает и бинарную операцию вычитания, и унарную операцию изменения знака. На этапе лексического анализа это различие должно быть установлено.

Устранить неоднозначность можно, по крайней мере, двумя способами:

- заменить унарную операцию бинарной, т. е. считать, что -*a* означает 0 - *a*;
- либо ввести специальный токен для унарной операции

Алгоритм Дейкстры перевода выражений в PN

Трансляторы языка программирования переводят выражения из скобочной записи в обратную польскую за один просмотр строки формулы, слева направо. В алгоритме перевода используются следующие правила:

Будем считать, что PN выражения будет формироваться в массиве, содержащем токены, будет использоваться вспомогательный стек, содержащий элементы PN (операции, имена функций) и круглые скобки.

Выражение просматривается один раз слева направо

1. Прочитать очередной элемент входной строки
2. Элемент входной строки:
   - 2.1 *конец строки*, перенести оставшиеся в стеке элементы в выходную строку, если стек пуст, то закончить перевод;
   - 2.2 *операнд* (переменная, число), перенести в выходную строку, вернуться к пункту 1;
   - 2.3 *операция*, если приоритет операции на вершине стека не меньше приоритета прочитанной операции, то перенести операцию из стека в выходную строку и повторить этот же пункт, иначе поместить прочитанную операцию в стек и вернуться к пункту 1;
   - 2.4 *символ функции*, записывается в стек, вернуться к пункту 1;
   - 2.5 *разделитель аргументов функции* (например, запятая), – до тех пор, пока верхним элементом стека не станет открывающаяся скобка, выталкиваем элементы из стека в выходную строку, вернуться к пункту 1;
   - 2.6 *открывающая скобка* записывается в стек, вернуться к пункту 1;
   - 2.7 *закрывающая скобка*, если элемент на вершине стека не является открывающей скобкой, то перенести этот элемент в выходную строку и повторить этот же пункт, если на вершине стека открывающая скобка, то удалить её из стека и вернуться к пункту 1.

Входная строка: a × ((b + c) + (d – e)) / f

Элемент	Стек	Выходная строка
a		a

×	×	a
(	×(	a
(	×((	
b	×((	ab
+	×((+	ab
c	×((+	abc
)	×(	abc+
+	×(+	abc+
(	×(+(	abc+
d	×(+(	abc+d
−	×(+(−	abc+d
e	×(+(−	abc+de
)	×(+	abc+de−
)	×	abc+de−+
/	/	abc+de−+×
f	/	abc+de−+×f
.		abc+de−+×f/

Перегрузка операторов требует определение типов операндов и записи в PN соответствующего варианта операции. Для определения вызова метода, который может быть полиморфным потребуется более сложный семантический анализ, связанный с контекстом.

Указатели функций

y−G(x,y+1,z)

PN: yxy1+zĞ − , где Ğ – токен имени функции, в описании которой есть количество операндов и указан номер в PN, с которого начинается PN оператора функции.

Конечно, обработка каждого из фактических параметров функции зависит от описания формальных параметров.

PN вызова функции представляет собой последовательность ее аргументов в PN, за которыми следует имя функции. Для функций с переменным числом параметров перед именем функции в PN вставляется дополнительный аргумент — количество фактических параметров в данном вызове функции. При интерпретации сначала из стека извлекается этот дополнительный аргумент, а затем — соответствующее ему число фактических параметров.

8. Создать обратную польскую запись для операторов присваивания.

$a := E$ будет записан в PN: $a\ E^{pn} :=$,

где «:=» — это двухместная операция, a и E^{pn} — ее операнды

В языках Си и других присваивание является оператором и операцией, причём операцией право-ассоциативной.

Пример. a=b=c=d+1; PN: abcd1+ = = =
Каждое следующее присваивание не выталкивает из стека предыдущее
В отличие от правила 2.3 в алгоритме Дейкстра.

строка	**стек**	**PN**
a		*a*
=	=	*a*
b	=	*ab*
=	==	*ab*
c	==	*abc*
=	===	*abc*
d	===	*abcd*
+	===+	*abcd*
1	===+	*abcd1*
.		*abcd1*+===

a+=b*=c/=d+1; PN: aabbccd1+/=*=+=

9. Создать обратную польскую запись для условного оператора.

10. Создать обратные польские записи для операторов цикла (while, do-while, for)

Безусловный переход (goto L)

t, k – позиции текущей строки Row и строки PN, k_L‼ − одноместная операция, безусловный переход в позицию k_L PN строки.

t	0	…	t_L	…	t_g	
Row	…	…	L:	…	goto L	
k	0	…	k_L	…	k_g	
PN	…	…	…	…	k_L	‼

$x:=k_L$ $=x$

t	0	…	t_g	…	…	t_L
	…	…	goto L	…	…	L:
k	0	…	k_g	…	…	k_L
	…	…	(p_L)	‼	…	…

$(p_L):=k_L$

Условный оператор (*if B then S*1 *else S*2)

k_L!F − одноместная операция, переход по лжи в позицию k_L PN строки.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
if	a	>	0	then	a	≔	–	a	else	a	≔	b	+	1	⊥		
k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	0	>	(p_3)	!F	a	0	a	–	≔	(p_{10})	‼	a	b	1	+	≔	

$(p_3)=k+12$ $(p_{10})=k+17$

11. Преобразовать n-мерный массив в одномерный. Реализовать обращение к нему как n-мерному, используя обратную польскую запись.

Переменные с индексами

Будем считать, что индексы начинаются с нуля.

Описание: *тип array* $m[n_1][n_2][n_3][n_4]$

Обращение: $m[i_1][i_2][i_3][i_4]$

Грамматика для массива: нетерминал A генерирует имя массива с последовательностью индексных выражений A→ A[E]|**id**[E].

Массив хранится в виде одномерного вектора M[N], где $N= n_1 \times n_2 \times n_3 \times n_4$.

Каждый элемент $m[i_1][i_2][i_3][i_4]$ хранится в векторе M[N] со смещением s

По строкам

$s=w\times[i_1\times n_2\times n_3\times n_4+i_2\times n_3\times n_4+i_3\times n_4+i_4]$

$s=w\times[((i_1\times n_2+i_2)\times n_3+i_3)\times n_4+i_4]$

PN: $mwi_1n_2\times i_2+n_3\times i_3+n_4\times i_4+\times Ă$

По столбцам

$s=w\times[i_1+i_2\times n_1+i_3\times n_1\times n_2+i_4\times n_1\times n_2\times n_3]$

$s=w\times[i_1+n_1\times(i_2+ n_2\times(i_3+n_3\times i_4))]$

PN: $mwi_1\ i_2n_2i_3n_3i_4\times+\times+\times+\times Ă$

где m – токен имени массива,

w – размер каждого элемента массива,

Ă – токен операции с двумя операндами (m,s) вычисления смещения адреса

m[3]=a+b[i+1][j]×c+d

PN: $mw3\times Ăabwi1+n_1j\times+Ăc\times+d+=$

Двумерный массив в JAVA не обязан быть прямоугольным, но в реализации (после трансляции) это будут различные одномерные массивы хранения. В других языках допустимы описания с произвольными целочисленными границами $[i_1:k_1]$, что не очень усложнило бы вычисления индекса хранения.