Методы нисходящего синтаксического анализа

Применяемый класс грамматик

- Контекстно-свободные грамматики
- Левая часть правила – нетерминальный символ, правая часть – произвольная строка

  $a \to v$, где a из N, v из A^*
- Порождаемые языки – контекстно-свободные языки
- Механизм распознавания – автоматы с магазинной памятью

Нисходящий рекурсивный алгоритм

- Пусть имеется входная лента, содержащая строку символов в алфавите $\mathcal{Z}$.
- Строка заканчивается специальным символом >.
- $\mathcal{N}$ **={L, E, F, T, P, Q}**
- $\mathcal{Z}$ = **{ x, +, *, (,), > }**
- Правила вывода:
  1. **L -> E Q**
  2. **E -> T F**
  3. **E -> T**
  4. **F -> + E**
  5. **F -> * T**
  6. **T -> x**
  7. **T -> (E P**
  8. **P ->)**
  9. **Q -> >**
- Исходный символ – L
- Построить алгоритм нисходящего разбора данной строки

Основные множества, управляющие синтаксическим разбором

- для каждого нетерминального символа построим множества терминальных символов, которые могут за ними следовать
  - **fin(E) = {), > }**
  - **fin (F) = {), > }**
  - **fin(T) = {), > }**
- Для каждого правила вывода множество символов, с которых может начинаться выводимая из него строка

1. **beg (L -> E >) = {x, (}**
2. **beg (E -> T F) = { x, (} dir(E -> TF) = first(F)= { +, * }**
3. **beg (E -> T) = {x , (}   dir (E -> T) = fin(T) = {), > }**
4. **beg (F -> + E) = { + }**
5. **beg (F -> * T) = { * }**
6. **beg (T -> x) = { x }**
7. **beg (T -> (E P} = { (}**
8. **beg (P ->)) = {) }**

Последовательность грамматического разбора LL(1)

```
L
E Q
T F Q
a F Q
a + E Q
a + T F Q
a + b F Q
a + b * T Q
a + b * ( E P Q
a + b * ( T F P Q
a + b * ( c F P Q
a + b * ( c + E P Q
a + b * ( c + T F P Q
a + b * ( c + d F P Q
a + b * ( c + d * T P Q
a + b * ( c + d * e P Q
a + b * ( c + d * e ) Q
a + b * ( c + d * e ) >
```

```
(1) L -> E >
(2) E -> T F   dir = +
(6) T -> x
(4) F -> + E
(2) E -> T F   dir = *
(6) T -> x
(5) F -> * T
(7) T -> ( E P
(2) E -> T F    dir = +
(6) T -> x
(4) F -> + E
(2) E -> T F    dir = *
(6) T -> x
(5) F -> * T
(6) T -> x
(8) P -> )
(9) Q-> >
```

```
a | + b*(c+d*e)>
a | + b*(c+d*e)>
+ | b *(c+d*e)>
b | * (c+d*e)>
b | * (c+d*e)>
* | ( c+d*e)>
( | c +d*e)>
c | + d*e)>
c | + d*e)>
+ | d *e)>
d | * e)>
d | * e)>
* | e )>
e | ) >
) | >
> |
```

1. **beg (L -> E >) = {x, (}**
2. **beg (E -> T F) = { x, (}** **dir (E -> T F) = { +, * }**
3. **beg (E -> T) = {x , (}** **dir (E -> T) = {), > }**
4. **beg (F -> + E) = { + }**
5. **beg (F -> * T) = { * }**
6. **beg (T -> x) = { x }**
7. **beg (T -> (E P} = { (}**
8. **beg (P ->)) = {) }**

Ограничения на КС-грамматику, чтобы она была LL(1)

- Грамматика не должна содержать правил вида: A->Bu, B->Cv, C->Aw, то есть лево рекурсивных цепочек вывода
- Для любых двух правил с одинаковой левой частью должно выполняться одно из двух следующих правил:

1. beg(A->v) & beg(A->w) == null то есть правла должны различаться по первому выводимому терминальному символу
2. Если это не так, dir(A->v) & dir(A->w) == null, то есть правила должны различаться по второму терминальному символу, выводимому из строк v и w

- Пример

Дополнительное ограничение

- В грамматику должны входить только правила следующего вида:
  - N -> u где u из $\mathcal{N}^*$
  - N -> tu где t из $\mathcal{T}$ и u из $\mathcal{N}^*$

Например:

Из правила

T -> (E)

следует сделать 2 правила:

T -> (ER

R ->)

где R – новый нетерминальный символ

Порядок работы

- В стек помещаем аксиому
- На входную ленту – строку, подлежащую распознаванию
- Выбираем самый левый нетерминал N в стеке и самый левый символ x на входной ленте и по этой паре принимаем решение:
  - если существует правило, N->u однозначно определяемое парой N, x, то:
    - если u = xv, то заменяем в стеке N на xv и удаляем символ x из вх ленты
    - если u начинается с нетерминала или u – пустая строка, то только заменяем N на u
  - Если пара N, x не однозначно определяют правило замены, то привлекаем След символ
  - Если не находится правила по паре N, x то ошибка
- Критерий правильности распознавания:
  - входная лента пуста
  - в стеке – копия входной строки