## Робота №05

*LL(1) граматики*

LL(1)-граматики – широкий клас КВ-граматик, котрі допускають ефективний синтаксичний аналіз. Тобто існує алгоритм, котрий для довільної LL(1)-граматики G будує синтаксичний аналізатор, що допускає ефективний синтаксичний аналіз мови L(G).

Мета роботи – реалізувати LL(1)-аналіз у випадку простого представлення КВ-граматик.

Більш детальну інформацію про LL(1)-граматики можна знайти в підручнику Ахо Д, Лам М, Сети Р, Ульман Дж. Компиляторы: принципы, технологии и инструментарий. (4.4 Нисходящий синтаксический анализ)

Для довільної КВ граматики G = (VN, VT, P, S) з об’єднаним алфавітом V = VN   VT можна визначити наступну функцію *first*.

* Якщо α V\*, то first(α) = {a  VT | α \*=lm> ax, xVT \*}  {$ | α \*=> ε }
* α \*=lm> β – існує лівосторонній вивід слова β зі слова α.
* Очевидно, що first(a) ={a}, a  VT, first(ε) = {$}.

Для довільної КВ граматики G = (VN, VT, P, S) first(α) визначає множину термінальних символів, кожний з яких може бути першим символом у слові, що виводиться зі слова α.

Граматика G - *LL(1) граматика*, якщо для довільних двох лівосторонніх виводів виду

* S \*=lm> wAα =lm> wβα \*=lm> wx , A -> β  P, xVT \*
* S \*=lm> wAα =lm> wγα \*=lm> wy , A -> γ  P, yVT \*

в яких first(x) = first(y) маємо β = γ.

КВ граматика G – *проста LL(1) граматика*, якщо в ній немає анулюючих правил виводу (A -> ε) і всі ВСІ альтернативи для кожного нетерміналу починаються з різних терміналів.

Наприклад КВ граматика G = ({S,B}, {a,b}, P, S) з множиною правил виводів P = {S -> aBS, S -> b, B -> a, B -> bSB} є проста LL(1) граматика.

Для довільної КВ граматики G = (VN, VT, P, S) з об’єднаним алфавітом V = VN   VT конструктивнобудується наступна таблиця nxt:

* Якщо A  VN, то nxt(A) = {a  VT | S \*=> γAα, a first(α)}  {$ | S \*=> γA}
* Для кожного нетерміналу A  VN, nxt(A) - прогнозуюча множина «наступних» терміналів, тобто терміналів, котрі можуть з»явитися ЗА нетерміналом A в одному з виводів граматики.
* Сукупність всіх таких множин, зв»язаних з відповідними нетерміналами, утворює прогнозуючу таблицю наступних терміналів nxt.

Просто довести, що КВ граматика (VN, VT, P, S) - LL(1) граматика тоді і тільки тоді, коли

first(βα)  first(γα) = {}

для всіх α, β, γ таких, що існують A -> β  P, A -> γ  P, β ≠ γ і S \*=lm> wAα.

Використовуючи функції first, nxt і попереднє означення LL(1) граматики отримуємо наступний критерій:

G – *LL(1) граматика тоді і тільки тоді*, коли для кожного нетерміналу A  VN  і всіх правил виводу A -> α1, …, A -> αn  P виконуються дві умови

* 1. first(αi)  first(αj) = {}, 1 ≤ i ≠ j ≤ n.
  2. якщо існує αi  \*=> ε, то first(αj)  nxt(A) = {}, 1 ≤ i ≠ j ≤ n.

Аналогічно таблиці nxt за КВ-граматикою можна побудувати таблицю fst:

* Для довільного нетерміналу A  VN : fst(A) = first(A)
* Для кожного нетерміналу A  VN, fst (A) - прогнозуюча множина «початкових» терміналів, тобто терміналів, котрі можуть з»явитися ПЕРШИМИ у виводі з нетерміналу A.
* Сукупність всіх таких множин утворює прогнозуючу таблицю-функцію початкових терміналів fst.

З математичної точки зору таблиці fst і nxt – це функції, область визначення яких є скінченна множина нетерміналів VN .

* *fst:* VN -> 
* *nxt*: VN -> .

Алгоритм *побудови таблиці fst*.

Будується послідовність наближень цієї таблиці fst0, fst1, …, fstm

* Для довільного A VN fsti(A)  fsti+1(A).
* Існує n, починаючи з якого ці наближення співпадають fstn = fstn+1 = ….

Покладаємо fst = fstn.

* Побудова початкової таблиці fst0
  + Для довільного A VN  такого, що існує A -> ε   P, fst0(A) = {$}.
  + Для довільного A VN  такого, що НЕ існує A -> ε   P, fst0(A) = {}
* Побудова fstn за fstn-1
  + Покладаємо fstn = fstn-1

Для кожного правила виводу A -> Y1 …Yk і кожного 1 ≤ i ≤ k

* + fstn(A) = fstn(A) (fstn(Y1) – {$})
  + Якщо для 1 ≤ j < i, $  fstn(Yj), то fstn(A) = fstn(A) (fstn(Yj) – {$})
  + Якщо для 1 ≤ j ≤ k, $  fstn(Yj), то fstn(A) = fstn(A) {$}

Для граматики G = ({S, T, A}, {+, -, (, ), d}, P3, S), де P3 = {S ->TA, A -> +TA, A -> -TA, A -> ε, T -> d, T -> (S)} побудова функції fst складається з кроків:

* fst0(S)={}, fst0(T)={}, fst0(A)={$}.
* fst1(S)={}, fst1(T)={(,d}, fst1(A)={+,-,$}.
* fst2(S)={(,d}, для інших значень x  VN, fst2(x) = fst1(x).
* Для всіх значень x  VN, fst3(x) = fst2(x).

Алгоритм *обчислення функції first*: V\* ->  базується на використанні таблиці fst.

* Якщо α=ε, то first(α) = {$}
* Якщо α=aβ, a  VT, то first(α) = {a}
* Якщо α=A, A  VN, то first(α) = fst(A)
* Якщо α=Aβ, A  VN  i $  fst(A), то first(α) = fst(A)
* Якщо α=Aβ, A  VN  i $  fst(A), то first(α) = (fst(A) – {$})  fist(β)

Алгоритм *побудови таблиці nxt*.

Як і випадку таблиці fst будується послідовність наближень nxt0, nst1, …, nxtn.

* Побудова початкової таблиці nxt0
  + nxt0(S) = {$}.
  + Якщо A VN і A ≠ S, nxt0(A) = {}.
* Побудова nxtn за nxtn-1
  + Покладаємо nxtn = nxtn-1

Для кожного правила виводу A -> α і кожного нетерміналу Y, що входить в α, тобто α = µYβ для деяких µ,β  V\*.

* + Якщо $  first(β), то nxtn(Y) = nxtn(Y)  first(β)
  + Якщо $  first(β), то nxtn(Y) = nxtn(Y) ( first(β) – {$})  nxtn(A)

Для граматика G = ({S, T, A}, {+, -, (, ), d}, P3, S), де P3 = {S ->TA, A -> +TA, A -> -TA, A -> ε, T -> d, T -> (S)} побудова таблиці nxt складається з кроків:

* nxt0(S) = {$}, nxt0(T) = {}, nxt0(A) = {}.
* nxt1(S) = {$,)}, nxt1(T) = {+,-}, nxt1(A) = {$} .
* nxt2(S) = {$,)}, nxt2(T) = {+,-,$,)}, nxt2(A) = {$,)}.
* Для всіх значень x  VN, nxt3(x) = nxt2(x)

Синтаксичний аналіздля LL(1) граматики G = (VN, VT, P, S) виконує *LL(1) аналізатор*  M = (VT, V{$}, W, σ, S, $), де

* VT - вхідний алфавіт,
* V{$} – алфавіт магазина,
* $  V – «дно» магазину,
* W = {0,1,2, …, n-1} – вихідний алфавіт – список номерів правил виводу граматики.
* S - початковий символ магазину (початковий нетермінал граматики).
* σ - управляюча таблиця, σ: VN (VT  {$}) W  {***E***}, котра для пари (A,a), A  VN, a  (VT  {$}), вказує дію аналізатора. Де дії
  + i  W - *Заміна (change)* верхнього символу магазину,використовуючи правило виводу з номером i.
  + ***E*** – *Помилка (error),*  аналізатор зупиняється.

Конфігурація LL(1) аналізатора – це (y, u, w) , де

* y – нерозпізнана частина слова x (Вхід),
* u – магазин,
* w – список номерів правил виводу (Вивід).
  + (x$, S$, ε) – початкова конфігурація
  + ($, $, π) – заключна конфігурація

На множині конфігурацій задається відношення наступна конфігурація (╞>), яке визначається управляючою таблицею σ. Якщо поточна конфігурація (ax, Bα, w) і

* B  VN  і σ(B,a) = i, і i-те правило виводу є B -> β, то
  + (ax, Bα, w) ╞> (ax, βα, w i)
* B  VT i B = a, то
  + (ax, Bα, w) ╞> (x, α, w)
* B = $ i a = $, то аналізатор попав в заключну конфігурацію ($, $, w)
  + початкове слово x розпізнане (x  L(G)) і w задає його лівосторонній вивід.
* В усіх інших випадках аналізатор зупиняється, знайшовши помилку
  + початкове слово x  L(G)

Відношення ╞\*> на конфігураціях – це транзитивне і рефлексивне замикання відношення ╞>.

Якщо (x$, S$, ε) ╞\*> ($, $, π), то M(x) = π і π - вихід аналізатора для вхідного слова x.

Якщо аналізатор, почавши працювати з початкового стану (x$, S$, ε), не попадає в заключний стан, то вважається, що M(x) - не визначено.

Для LL(1) граматики G існує M – аналізатор такий що L(G) = {x | M(x) - визначено}.

Нехай G = (VN, VT, P, S), де VN= {S,A}, VT = {a,b} і множина правил виводу P = {S -> aAS, S -> b , A -> a, A -> bSA}. Для перевірки чи являються граматика - LL(1) граматикою, шукаємо значення функції first(α) для всіх правил виводу.

A: first(a) = {a}, first(bSA) = {b} -- правила виводу A -> a, A -> bSA

S: first(aAS) = {a}, first(b) = {b} -- правила виводу S -> aAS, S -> b

Граматика G – являється LL(1) граматикою.

Для використання в аналізаторі всі правила виводу перенумеровуються:

0: S -> aAS, 1: S -> b, 2: A -> a, 3: A -> bSA

|  |  |  |  |
| --- | --- | --- | --- |
|  | a | b | $ |
| S | 0 | 1 | ***E*** |
| A | 2 | 3 | ***E*** |

Управляюча таблиця аналізатора має вид.

Розглянемо *як працює аналізатор*

* Вхідне слово abbab
  + Послідовність конфігурацій аналізатора:

(abbab$, S$, ε) ╞> (abbab$, aAS$, 0) ╞> (bbab$, AS$, 0) ╞> (bbab$, bSAS$, 03) ╞> (bab$, SAS$, 03) ╞> (bab$, bAS$, 031) ╞> (ab$, AS$, 031) ╞> (ab$, aS$, 0312) ╞> (b$, S$, 0312) ╞> (b$, b$, 03121) ╞> ($, $, 03121)

* + abbab  L(G)
  + Лівосторонній вивід слова abbab

S =lm> aAS =lm> abSAS =lm> abbAS =lm> abbaS =lm> abbab

* Вхідне слово aaa
  + Послідовність конфігурацій аналізатора:
  + (aaa$, S$, ε) ╞> (aaa$, aAS$, 0) ╞> (aa$, AS$, 0) ╞> (aa$, aS$, 02) ╞> (a$, S$, 02) ╞> (a$, aAS$, 020) ╞> ($, AS$, 020) ….. аналізатор зупиняється ..
  + aaa  L(G)

Алгоритм *побудови управляючої таблиці*. Нехай G = (VN, VT, P, S) – LL(1) граматика правила виводу якої перенумеровані послідовно 0, 1, … n-1, де n = |P|. Тоді у LL(1) аналізатора M = (VT, V{$}, W, σ, S, $) для цієї граматики вихідний алфавіт W = {0,1,2, …, n-1} і управляюча таблиця σ визначається так:

* Для правила виводу A -> α з номером i
  + Якщо a VT  і a  first(α), то σ(A,a) = i,
  + Якщо ε  first(α), то для довільного b  nxt(A), то σ(A,b) = i.
* Для всіх інших значень B  VN , b  VT  {$}, σ(B,b) = ***E***.

Побудова управляючої таблиці для різних граматик.

* Нехай G1 = ({S}, {+, -, (, ), d}, P1, S), де P1 = {S -> S+S, S -> S-S, S -> d, S -> (S)}. Граматика задає всі правильні арифметичні вирази, які використовують операції + і -, круглі дужки ( і ) та операнд d. Граматика неоднозначна, тому не LL(1) граматика.
* Граматика G2 = ({S, T}, {+, -, (, ), d}, P2, S), де P2 = {S -> S+T, S -> S-T, S -> T, T -> d, T -> (S)}, еквівалентна граматиці G1 і являється однозначною. Граматика задає мови виразів, в якій операції + і – являються лівоасоціативними. Але граматика має ліворекурсивні правила S -> S+T та S -> S-T, тому не LL(1) граматика.

Граматика G3 = ({S, T, A}, {+, -, (, ), d}, P3, S), де P3 = {S ->TA, A -> +TA, A -> -TA, A -> ε, T -> d, T -> (S)}, еквівалентна граматиці G2 і являється LL(1) граматика. Для цієї граматики

* first(S) = {d, (}, first(T) = {d,(}, first(A) = {+, -, $}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | + | - | ( | ) | d | $ |  |
| S | ***E*** | ***E*** | 0 | ***E*** | 0 | ***E*** |  |
| T | ***E*** | ***E*** | 5 | ***E*** | 4 | ***E*** |  |
| A | 1 | 2 | ***E*** | 3 | ***E*** | 3 |  |

* nxt(A) = {), $}

Якщо правилам виводу мають номера 0: S ->TA, 1: A -> +TA, 2: A -> -TA, 3: A -> ε, 4: T -> d, 5: T -> (S), то управляюча таблиця, що побудована згідно наведеного алгоритму, має вид.

В цій роботі для представлення термінальних і нетермінальних символів КВ-граматики використовуються символи (дані типа Char):

* Нетермінали – великі літери c (isUpper c == True).
* Термінали - довільні інші символи, крім символу '$', котрий використовується для спеціальних цілей.

***type***Grammar = [Production]

***type*** Production = (Char,String)

Граматика - список продукцій. Кожна продукція -це пара (n,rl), у якої n - нетермінал і rl - довільне слово, що не містить символу '$'.

* Нетермінал першої пари – це початковий нетермінал граматики.
* Всі продукції різні.

В LL(1)-аналізі використовуються множини термінальних символів і символу '$'. Кожна така множина задається впорядкованим списком символів без дублікатів.

***type*** Predict = [(Char,String)]

Елементи Predict - прогнозуючі таблиці початкових і наступних терміналів.

Прогнозуюча таблиця – це список пар (n,ts): n - нетермінал, ts – рядок, що задає множину (термінали і '$').

* ВСІ нетермінали n в таблиці різні і таблиця впорядкована по зростанню

***type*** Control = [((Char,Char),Int)]

Управляюча таблиця (елементи типу Control) для граматики gr - це список пар ((n,t),i):

* перша компонента - пара (n,t): n - нетермінал, t - термінал або '$'
* друга компонента - i : номер продукції граматики gr.

В допоміжному файлі, котрий включає визначення типів і даних для тестування, надати визначення наступних функцій. В разі потреби визначити необхідні додаткові функції.

1. Функція *addOne st c* - додає до множини-рядка *st* символ *c*. Функція *addAll st wd* - додає до множини-рядка *st* символи з рядка *wd*, повертаючи множину-рядок (*wd* - не обов"язково множина-рядок). Функція *addWithout st wd* – схожа на попередню але символ '$' з рядка *wd* НЕ додається (ігнорується). Функція *inter st1 st2* - реалізує перетин двох множин-рядків *st1* i *st2*, повертаючи множину-рядок.
   * addOne “(+-“ ‘d’ = “(+-d”
   * addOne “(+-“ ‘+’ = “(+-”
   * addAll “(+-“ “d+d” = “(+-d”
   * addWithout “(+-“ “d$” = “(+-d”
   * inter “acd” “cde” = “cd”
   * inter “$)” “\*+-“ = “”
2. Функція *tkPredict pt n* вибирає з прогнозуючої таблицi *pt* множину, що зв"язана з нетерміналом *n*. Якщо нетермінал *n* відсутній, то повертається "" - порожня множина. Функція *upPredict pt n st* змінює в таблиці *pt* множину, зв"язану з нетерміналом *n*, на рядок-множину *st*. Якщо нетермінал *n* відсутній, то в таблицю *pt* додається пара *(n,st)*.
   * tkPredict [('E',"(d"), ('F',"(d"), ('T',"(d")] ‘F’ = “(d”
   * tkPredict [('E',"(d"), ('T',"(d")] ‘F’ = “”
   * upPredict [('E',"(d"), ('T',"(d")] ‘T’ “$d” = [('E',"(d"), ('T',"$d")]
   * upPredict [('E',"(d"), ('T',"(d")] ‘F’ “$d” = [('E',"(d"), ('F',"$d"), ('T',"(d")]
3. Функція *parse gr ctl word*, що моделює роботу LL(1)-аналізатора для граматики *gr* з управляючою таблицею *ctl* на слові *word*, повертаючи Just il, де *il*- список номерів продукцій, що задають лівосторонній вивід слова *word*, або *Nothing* - якщо слово *word* НЕ належить мові граматики *gr*. Можна додатково визначити функцію *step gr ctl (input, staсk, result)*, що описує один крок роботи LL(1)-аналізатора з таблицею *ctl* на конфігурації *(input, staсk, result).* 
   * parse gr0 ctl0 "abbab” = Just [0,3,1,2,1]
   * parse gr0 ctl0 "aaa” = Nothing
4. Функція *first pFst st*, котра визначає множину початкових термінальних символів, котрі можна вивести з слова *st* (*pFst* - прогнозуюча таблиця початкових терміналів)
   * first [(‘S’,”(d”), (‘V’,”$+-“)] “VVS” = “(+-d”
   * first [(‘S’,”(d”), (‘V’,”$+-“)] “VV” = “$+-“
   * first [(‘S’,”(d”), (‘V’,”$+-“)] “+V” = “+”
5. Функція *buildingControl gr pFst pNxt*, що будує управляючу таблицю для LL(1)-граматики *gr* з прогнозуючими таблицями *pFst* початкових і *pNxt* наступних терміналів.
   * buildingControl gr0 pFst0 pNxt0 = [(('A','a'),2),(('A','b'),3),(('S','a'),0),(('S','b'),1)]
6. Функція *testingLL1 gr pFst pNxt*, що перевіряє, чи є граматика *gr* з прогнозуючими таблицями *pFst* – початкових і *pNxt* наступних терміналів - LL(1)-граматикою. Для реалізації можна додатково визначити функції:
   * *fromGrammar g*r - групує всі продукції граматики *gr* в список пари (n,rls), де n - нетермінал граматики і rls - ВСІ праві частини продукцій, що виводяться з n,
   * *testFst rls* - перевіряє 1 властивість для одного з нетерміналів граматики - з різних правих частин його продукцій *rls* виводяться різні початкові термінали.
   * *testFollow fs rls* – перевіряє 2 властивість для одного з нетерміналів граматики, коли з одної з продукцій нетерміналу *rls* виводиться порожнє слово ( *fs* - наступні символи для цього нетерміналу).
   * testingLL1 gr1 pFst1 pNxt1 = True
   * testingLL1 gr4 pFst4 pNxt4 = False
7. Функція *buildFst gr*, котра будує для граматики *gr* прогнозуючу таблицю початкових терміналів. Для реалізації можна додатково визначити функції:
   * *evalFst gr pFst* - будує послідовність наближень прогнозуючої таблиці *pFs*
   * *extandFst pFst (n,rul)* - розширює прогнозуючу таблицю *pFst,* обробляючи продукцію *(n,rul).*
   * buildFst gr0 = [('A',"ab"),('S',"ab")]
   * buildFst gr1 = [('S',"(d"),('T',"(d"),('V',"$+-")]
8. Функція *buildNxt gr pFst*, що будує для граматики *gr* прогнозуючу таблицю наступних терміналів, використовуючи ВЖЕ побудовану таблицю початкових терміналів *pFst.* Для реалізації можна додатково визначити функції:
   * *nontermTails gr* - будує ВСІ нетермінальні "хвости" продукцій граматики *gr.*
   * *evalNxt tails pFst pNxt* - будує послідовність наближень прогнозуючої таблиці наступних терміналів *pNx*, використовуючи: *tails* - ВСІ нетерміналні "хвости" продукцій граматики і *pFst*  - прогнозуючу таблицю початкових терміналів.
   * *extandNxtOne pFst n pNxt (m:st)* - розширює прогнозуючу таблицю *pNxt* обробляючи "хвіст" продукції граматики *(n,m:st).*
   * buildNxt gr0 pFst0 = [('A',"ab"),('S',"$ab")]
   * buildNxt gr1 pFst1 = [('S',"$)"),('T',"$)+-"),('V',"$)")]

*addOne*  :: String -> Char -> String

*addAll*  :: String -> String -> String

*addWithout* :: String -> String -> String

*inter* :: String -> String -> String

*tkPredict* :: Predict -> Char -> String

*upPredic*t :: Predict -> Char -> String -> Predict

*parse* :: Grammar -> Control -> String -> Maybe [Int]

*first*  :: Predict -> String -> String

*buildingControl* :: Grammar -> Predict -> Predict -> Control

*testingLL1* :: Grammar -> Predict -> Predict -> Bool

*buildFst*  :: Grammar -> Predict

*buildNxt* :: Grammar -> Predict -> Predict

Зауваження:

Назва файлу Family05.hs (Family – прізвище студента). Файл включає модуль Family05 і створюється на основі файла-заготовки HWP05.hs