## Робота №04

*Синтаксичний аналіз*

Алфавіт – скінченна множина символів: букви, цифри, спеціальні символи. Як правило підмножина елементів типу Char.

Мова L в алфавіті V – це довільна підмножина слів в алфавіті V, як правило, нескінченна. Скінченна множина правил, яка описує як побудувати всі слова мови – синтаксис мови.

Задача синтаксичного аналізу – для заданих мови L і слова w визначити чи містить мова L слово w. Програми, що розв’язують цю задачу – аналізатори.

Перш ніж розв’язувати задачу аналізу, часто потрібно спочатку описати саму мову – задати синтаксис мови. Основний засіб для задання синтаксису мови – КВ-граматики.

*Контекстно вільна граматика* (КВ граматика) G = (VN, VT, P, S), має

* Алфавіти нетерміналів VN  і терміналів VT, що неперетинаються VN   VT  = {}.
* Початковий нетермінал, S VN
* Об’єднаний алфавіт позначається як V = VN   VT
* Скінченну множину правил A -> α  P, де A VN , α V\*.

Нехай α, β  V\*, говорять, що β *безпосередньо виводиться* з α в граматиці G (α => β), якщо існують A -> γ  P, α1  V\*, α2  V\* такі, що α = α1 Aα2 , β = α1 γα2.

Нехай α, β  V\*, говорять, що β *виводиться* з α в граматиці G (α \*=> β), якщо

* α = β або
* α0 = α, α1, α2, …, αn = β, такі, що α0, α1, α2, …, αn  V\*, αi => αi+1 для 0 ≤ i < n.

Послідовність α0 = α => α1 => α2 => … => αn = β - вивід слова βзі словаα.

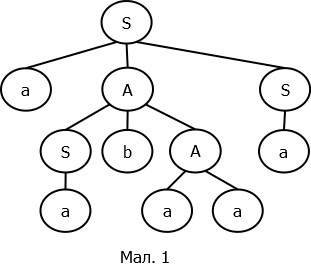
Множина L(G) = {w | w  VT \*, S \*=> w} – *мова породжувана граматикою* G.

Вивід α0 = α => α1 => α2 => … => αn в граматиці G називається *лівостороннім*, якщо для 1 ≤ i < n, αi. = xi Ai βi, xi  VT \*, Ai VN, βi  V\*, Ai -> γi  P, αi+1. = xi γi  βi. Лівостороння безпосередня виводимість позначається =lm>, а лівостороння виводимість - \*=lm> (lm – скорочення від “left most”).

У граматики G = (VN, VT, P, S) VN= {S,A}, VT = {a,b}, P = {S -> aAS, S -> a, A -> SbA, A -> ba } лівосторонній вивів слова aabba має вид:

* S =lm> a**A**S => a**S**bAS =lm> aab**A**S =lm> aabba**S** =lm> aabbaa

Якщо G = (VN, VT, P, S) КВ-граматика, у якої S \*=> w, wVT \* то існує лівосторонній вивід S \*=lm> w

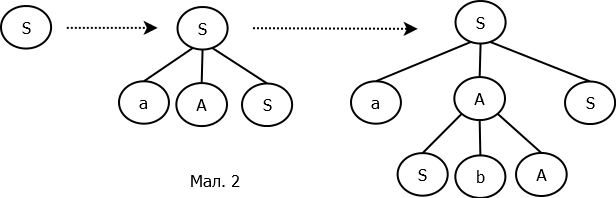
Нехай G = (VN, VT, P, S) – КВ-граматика. *Дерево виводу* (*дерево розбору, синтаксичне дерево*) для G – це помічене дерево, що задовольняє умовам:

* Кожен вузол має мітку - символ з V або ε.
* Мітка кореня S.
* Якщо вузол має хоч одного сина, то його мітка – нетермінал.
* Якщо вузли n1, n2, …, nk – сини вузла n, перераховані зліва направо, з мітками A1, A2, …, Ak, а мітка вузла n – A, то A -> A1A2…Ak  P.
* ε – може бути тільки міткою листа

На Мал. 1 наведено приклад дерева виводу для попередньої граматики G. Це дерево можна побудувати за довільним виводом слова aabbaa. За деревом можна побудувати лише один лівосторонній (правосторонній) вивід слова aabba. Крона дерева – це результат виводу.

*Задача синтаксичного аналізу* – для заданої G = (VN, VT, P, S) КВ граматики і слова w  VT \* визначити чи w  L(G). Якщо належить, то побудувати вивід слова w в граматиці G або синтаксичне дерево, крона якого є слово w.

* Як правило, в задачі синтаксичного аналізу граматика – зафіксована, а слово – змінюється.

Часто дерево виводу будують зверху-вниз, проходячи лівосторонній вивід слова w.

Метод рекурсивного спуску (recursive descent) реалізує синтаксичний аналіз зверху вниз, будуючи для вхідного слова w його лівосторонній вивід в G *.* Створюється набір рекурсивних аналізаторів (функцій, процедур) по одному аналізатору для кожного нетермінала A. Цей аналізатор розпізнає одну синтаксичну конструкцію – ВСІ слова що виводяться з нетерміналу A.

* Головна проблема : на кожному кроці побудови S \*=> xAγ слова w=xav необхідно вибрати одну з можливих альтернати A->α1, …, A->αk.
* Найчастіше цей вибір приймається на основі першого символу a залишку - слова av.

Не кожна граматика дозволяє ефективно розв»язувати цей вибір. В попередній граматиці G що наведена, при побудові виводу довільного слова для нетермінала S є дві альтернативи S -> aAS, S -> a для ефективного вибору з яких не досить першого символу залишку.

Розглядається мова, алфавіт якої містить дужки ‘(‘, ‘)’, ‘[‘, ‘]’, ‘{‘, ‘}’ та символ проміжку ‘ ‘. В мову входять всі слова, що мають баланс відкриваючих і закриваючих круглих, квадратних і хвилястих дужок. Кожна будь яка пара дужок розділяє рядок на три частини α, β, γ: α’(‘β’)’γ; α’[’β’]’γ; α’{‘β’}’γ, котрі також мають баланс дужок.

Граматика з нетерміналами B і C задає цю мову G0 = (VN, VT, P0, S) VN= {B,C}, VT = {[,],{,},(,), ‘ ‘}, P0 = {B -> (B)B, B -> [B]B, B -> {B}B, B->C, C->’ ‘C, C-> ε}

Нехай є мова M, що породжується граматикою G, M=L(G) . Кожне слово w M має

* синтаксис (структуру), котру задає вивід слова S \*=> w,
* семантику (значення v). Семантику v (w) можна обрахувати за виводом слова S \*=> w.

Для того, щоб обчислити семантику слова w паралельно з синтаксичним аналізом слова w виконують додаткові обчислення – *синтаксично-орієнтована трансляція*.

* З кожним правилом A -> A1…An-1An  P зв»язується правило, що дозволяє вирахувати значення v(A) конструкції, що описується нетерміналом A, за значеннями її компонент v(A1),… v(An-1), v(An).

Розглядаються арифметичні вирази, які можна скласти з десяткових цифр, круглих дужок і бінарних операцій ‘+’, ‘-‘ та ‘\*’. В залежності від семантики обчислення операцій (пріоритет і асоціативність) одне слово (синтаксис) може мати різну семантику. Наприклад значення виразу 6+8\*5-4 може бути:

* 66 = (((6+8)\*5)-4) – всі операції мають однаковий пріоритет і ліво-асоціативні.
* 14 = (6+(8\*(5-4))) - всі операції мають однаковий пріоритет і право-асоціативні.
* 42 = ((6+(8\*5))-4) - операції ‘+’, ‘-‘ мають однаковий пріоритет, котрий менший за пріоритет операції ‘\*’, і всі операції ліво-асоціативні.

Щоб успішно реалізувати обчислення таких арифметичних виразів в процесі синтаксичного аналізу, для кожного випадку потрібна своя контекстно-вільна граматика, всі граматики мають однаковий термінальний алфавіт VT  = {(,),0,1,2,3,4,5, 6,7,8,9} :

* G1 = ({E,F,A}, VT, P1, E) P1 = {E -> FA, A -> +FA, A -> -FA, A -> \*FA, A -> ε, F -> (E), F -> 0,..., F -> 9 }
* G2 = ({E,F,A}, VT, P2, E) P2 = {E -> FA, A -> +E, A -> -E, A -> \*E, A -> ε, F -> (E), F -> 0,..., F -> 9 }
* G3 = ({E,T,A,F,B}, VT, P3, E) P3 = { E -> TA, A -> +TA, A -> -TA, A -> ε , T -> FB, B -> \*FB, B -> ε, F -> (E), F -> 0,..., F -> 9 }

Більш детальну інформацію про синтаксичний аналіз і метод рекурсивного спуска можна знайти в підручнику Ахо Д, Лам М, Сети Р, Ульман Дж. Компиляторы: принципы, технологии и инструментарий. (4.2 Контекстно-свободные грамматики, 4.3 Разработка грамматики, 4.4.1 Синтаксический анализ методом рекурсивного спуска )

На основі допоміжного файлу створити файл, в якому надати визначення наступних функцій-аналізаторів. В разі потреби визначити необхідні додаткові функції.

1. Функцію *analyseG*, яка реалізує синтаксичний аналіз мови, котра породжується наступною КВ-граматика G = (VN, VT, P, S) VN = {S, A}, VT = {a, b}, P = {S -> aSbAa, S -> b, A -> baAS, A -> a}.

Можна скористатися функцією match, що визначена в допоміжному файлі, і визначити допоміжні функції s і a, котрі розпізнають слова, що виводяться з нетерміналів S і A відповідно.

* analyseG “abbbaaba” = True
* analyseG “ba” = False

1. Написать функція *balance str,* котра в рядку *str* розпізнає мову (задану граматикою G0), слова якої мають баланс дужок. Наприклад:
   * balance “{[[] () ] } {}” = True
   * balance “{[[}]] ( ) ] } {}” = False
2. В задачах 3-6 розглядаються арифметичні вирази, які можна скласти з десяткових цифр, круглих дужок і бінарних операцій ‘+’, ‘-‘ та ‘\*’. Функція-аналізатор *analyseExpr* розпізнає рядок, що містить синтаксично вірний арифметичний вираз. Для побудови аналізатора *parseExpr*, можна використати довільну з наведених трьох граматик G1, G2, G3.
   * analyseExpr “6+8\*5-4” = True
   * analyseExpr “6+8\*5-“ = False
3. Функції *evalLeft* розпізнає синтаксично вірний арифметичний вираз, повертаючи його ціле значення v – Just v або Nothing в іншому випадку. Значення виразу обчислюється в припущенні, що всі операції лівоасоціативні і мають однаковий пріоритет.
   * evalLeft “6+8\*5-4” = Just 66
   * evalLeft “6+8\*5-” = Nothing
4. Функції *evalRigth* схожа на попередню *evalLeft,* але значення виразу обчислюється в припущенні, що всі операції правоасоціативні і мають однаковий пріоритет.
   * evalRigth “6+8\*5-4” = Just 14
   * evalRigth “6+(8\*” = Nothing
5. Функція *evalPrior* схожа на попередню *evalLeft,* але значення виразу обчислюється в припущенні, що всі операції лівоасоціативні, але операції ‘+’, ‘-‘ мають однаковий пріоритет, котрий менший за пріоритет операції ‘\*’.
   * evalPrior “6+8\*5-4” = Just 42
   * evalPrior “6+” = Nothing

Для останніх трьох задач використайте три різні еквівалентні КВ-граматики G1, G2, G3.

При розв»язку задач 3-6 можна створити допоміжні функції, котрі розпізнають слова, що виводяться з відповідних нетерміналів.

*analyseG*  :: String -> Bool

*balance* :: String -> Bool

*analyseExpr* :: String -> Bool

*evalLeft* :: String -> Maybe Int

*evalRigth* :: String -> Maybe Int

*evalPrior* :: String -> Maybe Int

Зауваження:

Назва файлу Family04.hs (Family – прізвище студента). Файл включає модуль Family04 і створюється на основі файла-заготовки HWP04.hs