JIP

Analiza składni, gramatyki

Książka o różnych językach i paradygmatach

Polecam jako obowiązkową lekturę do przeczytania dla wszystkich prawdziwych programistów!

Podsumowanie wykładu 2

- Analiza leksykalna
- Automaty skończone AS
- Wyrażenia regularne RE
- Języki AS i RE – równoważność
- Reguły Thompsona
- Narzędzie flex

- Fortran

Plan wykładu 3

- Analiza syntaktyczna
- Gramatyki (G)
- Podstawowe pojęcia
- Niejednoznaczność G
- Drzewa parsowania/składni

Gramatyki formalne

- Gramatyka
  - narzędzie do opisu i analizy języka
  - zbiór reguł, według których buduje się poprawne zdania języka
- Przykład: gramatyki języków naturalnych; polsski,...

zdanie -> <podmiot><orzeczenie><dopełnienie>

podmiot -> <osoba> | rzecz | zwierzę

orzeczenie -> idzie | stoi | leci | mówi | jedzie

dopełnienie -> do kina | szybko | na zajęcia | ładnie

osoba -> Jan | ...; rzecz -> samolot | auto ...; zwierzę -> koza | kot | ...;

Wg tych reguł można skonstruować (prawie) poprawne zdania języka polskiego, np.

Jan idzie do kina.

Samolot leci ładnie.

Auto mówi szybko.

itd.

Gramatyki formalne

- Jedne z tych zdań są sensowne, a inne nie są
- W procesie analizy syntaktycznej interesuje nas tylko składnia (zgodność z gramatyką), a nie znaczenie zdań
- Zdania te można otrzymać w procesie zwanym wyprowadzeniem lewo- lub prawostronnym
- Przykład wyprowadzenia lewostronnego:

zdanie -> <podmiot><orzeczenie><dopełnienie>

-> Jan <orzeczenie><dopełnienie>

-> Jan idzie <dopełnienie>

-> Jan idzie do kina

Gramatyki formalne

- Znaczeniem zdań języka zajmuje się semantyka
- Analiza składni nosi nazwę analizy syntaktycznej

- Gramatyka G =
  (Zbiór symboli nieterminalnych = N,
  Zbiór terminali = T,
  Produkcje = P,
  Symbol początkowy = S)

Gramatyki formalne

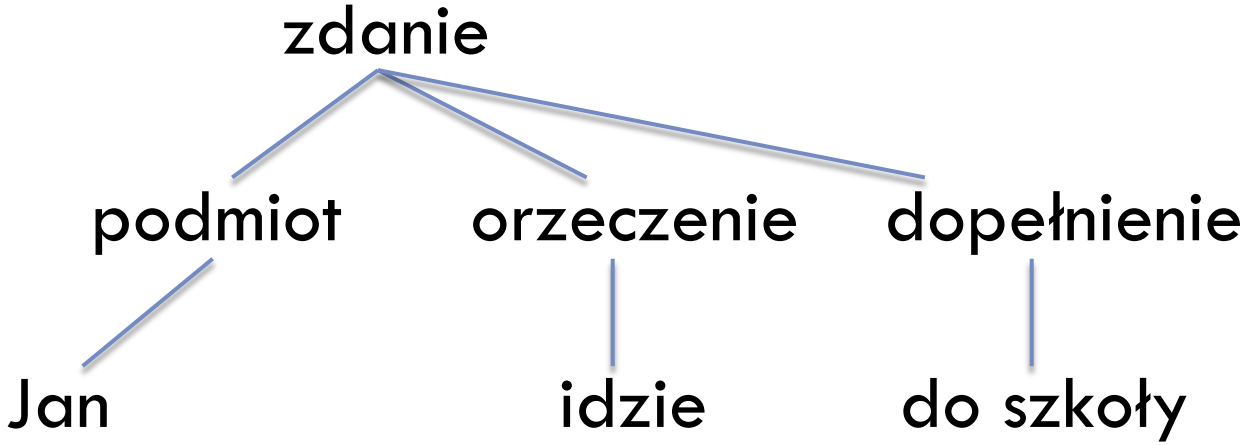
- Gramatyka G = (V, T, P, S)
- Produkcje – reguły zastępowania symboli, wyprowadzania zdań, np. X -> $Y_1 Y_2 Y_3$ ... Y_n
- głowa produkcji – lewa strona reguły
- symbol produkcji - ->
- terminal, symbol końcowy, słowo
- symbol nieterminalny, symbol
- symbol startowy, początkowy
- Zdanie – ciąg symboli terminalnych otrzymany w procesie wyprowadzenia lewo- | prawostronnego
- Zapis BNF (Backus-Naur Form)

Gramatyki formalne

- Wyprowadzaniu zdań towarzyszy proces wybierania reguł i ich kolejność – historia wyprowadzeń
- Inną metodą wyprowadzania zdań sa drzewa wyprowadzeń, drzewa syntaktyczne

Gramatyki formalne

- Przykład



Przykład

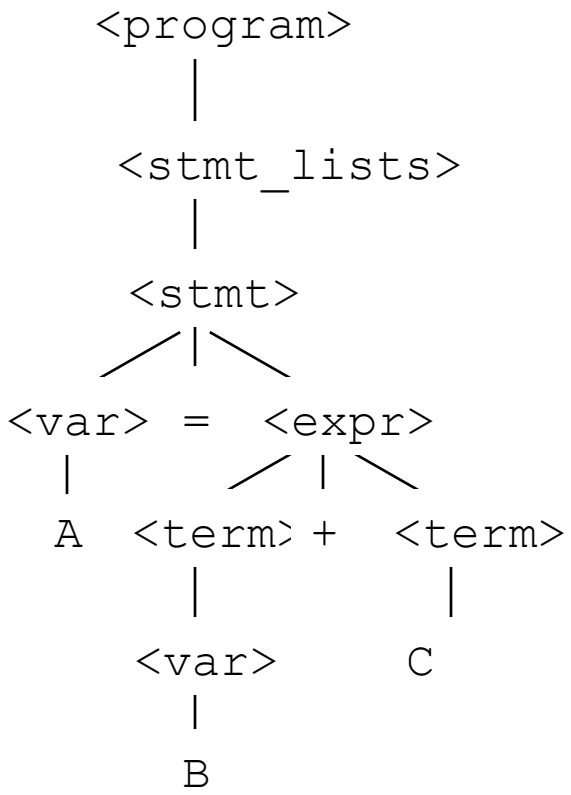

```
<program> => begin <stmt_list> end
<stmt_list> => <stmt> | <stmt> : <stmt_list>
<stmt> => <var> = <expr>
<var => A|B|C
<expr> => <var> + <var>
                | <var> - <var>
                | <var>
```

Przykład


```
A = B + C
<stmt_lists> => <stmt>
             => <var> = <expr>
             => A = <expr>
             => A = <term> + <term>
             => A = <var> + <term>
             => A = B + <term>
             => A = B + C
```

Przykład

- Drzewo wyprowadzenia



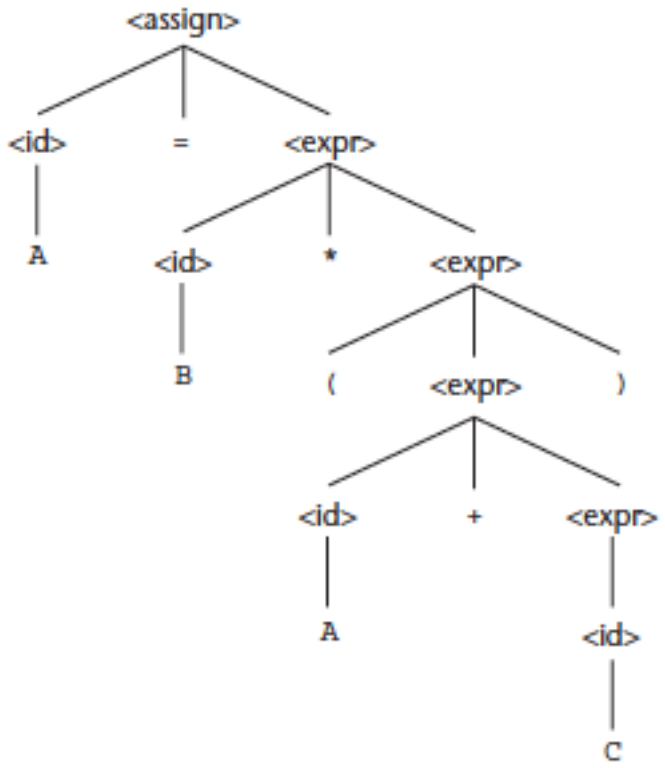
Przykład (Sebesta)

Prosta gramatyka instrukcji przypisania

<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
| <id> * <expr>
| (<expr>)
| <id>

Wyprowadzenie A = B*(A+C)

<assign> => <id> = <expr>
=> A = <expr>
=> A = <id > * <expr>
=> A = B * <expr>
=> A = B * (<expr>)
=> A = B * (<id> + <expr>)
=> A = B * (A + <expr>)
=> A = B * (A + <id>)
=> A = B * (A + C)



Więcej definicji

- Rozpatrzmy gramatykę z produkcjami
  S -> A| B
  A -> Ax | y
  B -> z
- symbole: {S, A, B, x, y, z}
- symbole nieterminalne: N = {S, A, B}
- symbole terminalne: T = {x, y, z}
- V i T są rozłączne

Więcej definicji

- Umowa
  - Duże litery – symbole nieterminalne
  - małe litery łac. – terminale
  - małe litery gr. – dowolne, tzn terminale $\vee$ nieterminale
- Dla zdefiniowania języka potrzebujemy zbioru produkcji postaci α -> β
- W gramatykach bezkontekstowych α jest pojedyńczym symbolem nieterminalnym, a β jest dowolnym łańcuchem terminali i symboli nieterminalnych.

Więcej definicji

- Forma zdaniowa = ciąg symboli wyprowadzonych z symbolu startowego S, zawierający ew. symbole nieterminalne

- Oznaczenia
  - α -> β wyprowadzenie β z α
  - α ->* β wyprowadzenie β z α przy zastosowaniu zera lub więcej produkcji, reguł
  - α ->$^+$ β wyprowadzenie β z α przy zastosowaniu przynajmniej jednej produkcji

Równoważność gramatyk

- Językiem L(G) zdefiniowanym przez gramatykę G nazywamy zbiór zdań wyprowadzonych z G

- Dwie gramatyki G i G' są równoważne jeśli języki L(G) i L(G') sa takie same.

Hierarchia gramatyk wg Chomsky'ego

- Noam Chomsky
  - 4 typy gramatyk
- typ 0 – gramatyki ogólne: produkcje α -> β , α , β są z V i α jest niepusty
- typ 1 – gramatyki kontekstowe: produkcje są postaci α X γ -> α β γ ; α , β , γ są z V, α , γ stanowią kontekst symbolu X
- typ 2 – gramatyki bezkontekstowe: X -> α , α jest z V, X jest pojedynczą kategorią syntaktyczną; Każde X można zastąpić przez α niezależnie od kontekstu
- typ 3 – gramatyki regularne: X -> a, X-> aY, X -> ε , gdzie a jest symbolem terminalnym

Hierarchia gramatyk wg Chomsky'ego

- Najbardziej restrykcyjną jest gramatyka typu 3, a najogólniejszą jest gramatyka typu 0
- Gramatyka typu 3 jest gramatyką typu 2, a gramatyka typu 2 jest typu 1, a ta jest typu 0.
- Gramatyka typu 3, bezkontekstowa, okazuje się najdogodniejszą w projektowaniu języków programowania

Przykłady

- Gramatyka palindromów, terminale: 0, 1

P -> ε | 0 | 1 | 0P0 | 1P1

0
0101010
111
00110101100

Przykłady

- Gramatyka prostych wyrażeń arytmetycznych

  E -> E op E | (E) | int
  op -> + | - | * | /

  2+3*4, 7+(8+5)*4/3, (6*7+2)/2+8+5

- Wyprowadzenia, zadanie.

Problemy

- Parsery = analizatory składni mogą mieć problemy w przypadku gdy gramatyka jest niejednoznaczna lub gdy występuje tzw. rekurencja lewostronna, lub faktoryzacja

  Krótko o powyższym.

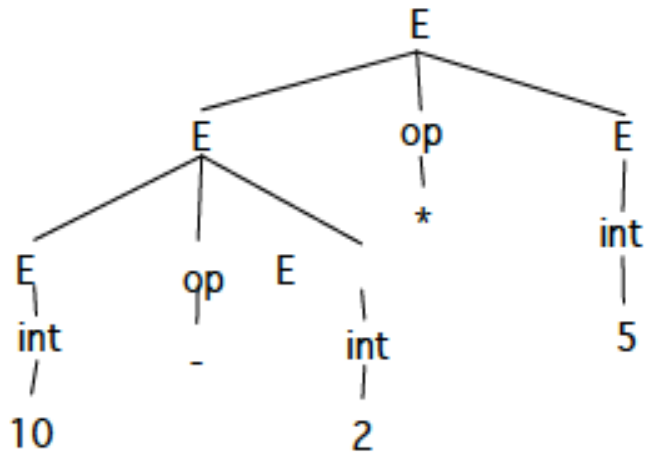
Niejednoznaczność G

- Niejednoznaczność objawia się w przypadku gdy istnieje więcej niż jedno drzewo wyprowadzeń
- Przykład. Gramatyka prostych wyrażeń arytmetycznych z przykładu powyżej.
  E -> E op E | (E) | int
  op -> + | - | * | /

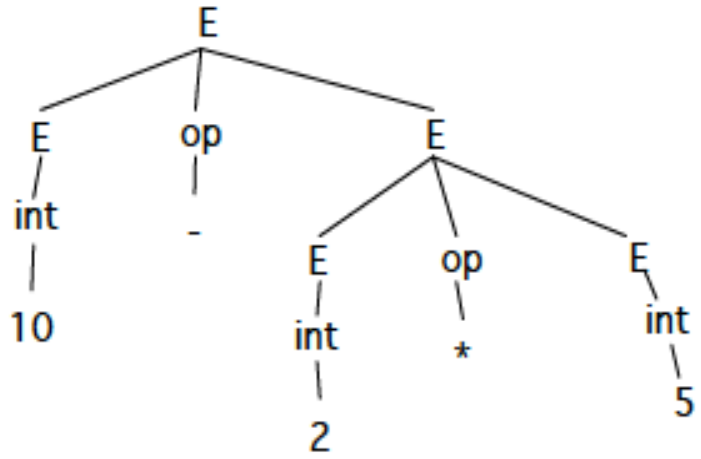
- Zajmijmy się wyprowadzeniem wyrażenia 10-2*5
  - E -> E op E -> 10 op E -> 10 + E -> 10-E op E ->10-2 op E -> 10-2*E -> 10-2*5
  - E -> E op E -> E op E op E -> 10 op E op E -> 10 - 2 op E -> 10-2*E -> 10-2*5

Niejednoznaczność G

- Lepiej widać to na drzewach wyprowadzeń. Mamy dwa drzewa – niejednoznaczna G



plon ~ (10 – 2)*5



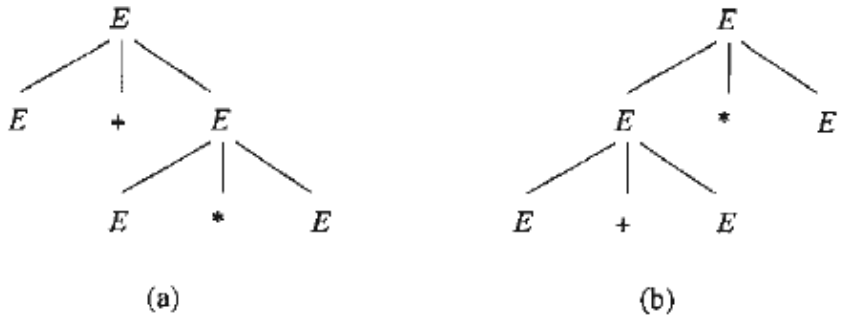
plon ~ 10 – (2*5)

Niejednoznaczność G

- Lewe drzewo: najpierw op zamieniany jest na *, a następny na –
- Prawe drzewo: odwrotnie
- Sugeruje to, że plon lewego drzewa można złożyć do wyrażenia (10-2)*5, a prawego do 10-(2*5)
- Oba drzewa są legalne dla podanej gramatyki
- Prawidłowe powinno być drzewo prawe gdyż prawidłowo interpretuje kolejność działań (*, -)

Przykład, ten sam

- Produkcje E->E+E | E*E pozwalają generować wyrażenia w dowolnej kolejności. Np. forma zdaniowa E+E*E może powstać w wyprowadzeniach:
  - 1. E -> E+E -> E+E*E
  - 2. E -> E*E -> E+E*E
- Drzewa



Niejednoznaczność

- Nie ma ogólnej metody, która pozwala przewidzieć niejednoznaczność gramatyk
- Znane niejednoznaczności można zakodować budując w odpowiedni sposób analizator, np. tak, by wybierał prawe drzewo składni w podanym przykładzie
- Inna metoda usuwania niejednoznaczności polega na redefinicji gramatyki na gramatykę równoważna bez niejednoznaczności

Niejednoznaczność

- W przypadku powyższej gramatyki wyrażeń arytmetycznych

  E -> E op E | (E) | int

  op -> + | - | * | /

  możemy dokonać podziału wyrażeń na składniki (terms, T) i czynniki (factors, F) i wymusić ich rozwinięcia w określonym porządku:

  E -> E top E | T

  top -> + | -

  T -> T fop T | F

  fop -> * | /

  F -> (E) | int

- Jest tylko jedno drzewo składni dla 10-2*5. Sprawdź!

Niejednoznaczność G

- ... ale co z 10-3-2?
  Powinno być (10-3)-2 czy 10-(3-2)?
  Łączność lewostronna!
- Prawidłowa G, równoważna, po zmianach:

```
E -> E top T | T
top -> + | -
T -> T fop F | F
fop -> * | /
F -> (E) | int
```

- Nie ma reguł ogólnych. Jednak w przypadku języków programowania jest niewiele takich przypadków, które wymagają takich zmian gramatyki.

Produkcje rekurencyjne

- Przykład.

  lista_zmiennych -> zmienna | lista_zmiennych, zmienna

- Ogólniej:

  $A \rightarrow \alpha \mid A\,\alpha$

  Jeśli z prawej strony produkcji występuje z lewej strony ten sam symbol nieterminalny, który występuje po stronie lewej produkcji, to takie produkcje nazywamy lewostronnie rekurencyjnymi (tutaj A lub lista_zmiennych). Prowadzi to do problemów w przypadku pewnych technik używanych w analizie syntaktycznej

Rekurencja

- Istnieje sposób eliminowania rekurencji lewostronnej
- Przykład: X -> Xa | Xb | AB | C | | DEF
- Metoda
  - wprowadzamy nowy symbol nieterminalny X'
  - X' dodajemy po prawej stronie wszystkich produkcji nierekurencyjnych, a więc do AB, C, DEF
  - dodajemy produkcję X' -> "odwrotność" lewych produkcji rekurencyjnych X, z X zastąpionym przez X' oraz produkcję X'-> ε .

    X -> ABX' | CX' | DEFX'
    X' -> aX' | bX' | ε
  - Otrzymaliśmy gramatykę z rekurencją prawostronną

Faktoryzacja

- Parser czyta wejście z lewej strony na prawą
- Dla każdego przeczytanego tokenu wygodnie jest zdecydować od razu, która reguła ma być użyta w procesie wyprowadzania
- W przypadku gdy z prawej strony produkcji występują wspólne symbole pojawia się problem wyboru właściwej produkcji
- Przykład

  instr -> `if` warunek `then` instr `else` instr |
  `if` warunek `then` instr | ...

Faktoryzacja

- W przykładzie wspólnym prefiksem jest
  `if warunek then instr`
- Po napotkaniu `if` parser "nie wie" której reguły użyć; prowadzi to do niezrównoważonych instrukcji warunkowych `if – else`
- Technika lewostronnej faktoryzacji pozwala usunąć problem
  - Produkcje przepisujemy tak aby deecyzję o wyborze reguły odsunąć w czasie, na moment gdy wczytamy odpowiednio wiele informacji
  - Część wspólną obu produkcji faktoryzujemy
  - Dodajemy nową regułę, która będzie stosowana później

Faktoryzacja

instr -> `if` warunek `then` instr `else` instr |
`if` warunek `then` instr | ...

zamieniamy na

instr -> `if` warunek `then` instr Opcelse| ...
Opcelse -> `else` instrukcja | ε |...

W ten sposób decyzja o wyborze części `else` lub ε zostaje odłożona na później.

Ukryta rekurencja

- Może sią zdarzyć, że rekurencja jest ukryta
- Przykład.
  S -> Ta | cd
  T -> Se | bbS
- Rekurencja staje się jawna po wstawieniu za S ciągu Ta w pierwszej produkcji T->...:
  S -> Ta | cd
  T -> Tae | cde | bbS
- Po eliminacji rekurencji (zgodnie z podanymi wcześniej regułami) otrzymamy równoważną gramatykę
  bez rekurencji lewostronnej
  S -> Ta | cd
  T -> cdeT' | bbST'
  T -> aeT' | ε

Ukryta faktoryzacja

- Zadanie
  Pokazać, że następującą gramatykę należy sfaktoryzować
  A -> da | acB
  B -> abB | daA | Af
- W zadaniu pierwsza i druga produkcja B przekrywają się z trzecią. Aby to unaocznić podstawimy A do trzeciej produkcji B
  A -> da | acB
  B -> abB | daA | daF | acBf
- Po faktoryzacji (wprowadzamy wspólne faktory M, N)
  A -> da | acB
  B -> aM | daN
  M -> bB | cBf
  N -> A | f

ALGOL

- Algol był pierwszym językiem programowania opisanym przy użyciu gramatyki bezkontekstowej
- John Backus, Peter Naur, notacja BNF lub EBNF (extended)
- Własności Algolu
  - struktury blokowe
  - silne typowanie
  - zakresy
  - procedury
  - wywołania przez wartość i referencyjne
  - rekurencja (wymaga stosu w czasie wykonania programu; brak w językach Fortran i Cobol)

Literatura

- Aho, Seti, Ullman
- Hopcroft
- Scott

Co dalej?

- Lex/Flex
- Bison/yacc
- Semantyka