

# Metody Realizacji Języków Programowania

## Analiza składniowa

Marcin Benke

MIM UW

11 października 2010

# Analiza składniowa

Celem analizy składniowej jest budowa drzewa struktury na podstawie strumienia leksemów.

Dzisiaj:

- Opis składni: gramatyki, BNF
- Wywody, drzewa wywodu
- Wieloznaczność
- Analiza top-down i bottom-up
- Analiza metodą LL(1)

# Gramatyki

Formalnie, a gramatyka bezkontekstowa jest czwórką:

$$G = \langle T, N, S, P \rangle$$

gdzie

- $T$  jest zbiorem symboli terminalnych (leksemów)
- $N$  jest zbiorem symboli nieterminalnych (rozłącznym z  $T$ )
- $S \in N$  jest symbolem początkowym
- $P$  jest zbiorem produkcji postaci  $\alpha \rightarrow \beta$  gdzie  $\alpha \in (T \cup N)^+$ ,  $\beta \in (T \cup N)^*$

**Konwencja:** symbole nieterminalne oznaczamy zwykle wielkimi literami.

Gramatyka indukuje naturalną relację przepisywania  $\rightarrow$  na zbiorze  $(T \cup N)^*$ : napisy mogą być przepisywane zgodnie z produkcjami

# BNF

*Backus-Naur Form* — notacja dla gramatyk bezkontekstowych

- Symbole terminalne są wyróżniane przez użycie cudzysłowów, lub innej czcionki.
- W produkcjach ::= zastępuje  $\rightarrow$
- Można skrótowo zapisywać zbiory produkcji:

$$E ::= E + T \mid T$$

zamiast

$$E ::= E + T$$

$$E ::= T$$

- Pierwsza produkcja wyznacza symbol startowy.
- Istnieje kilka wariacji BNF.
- Nieterminalne są czasem zpisywane w nawiasach kątowych  $\langle i \rangle$  (np:  $\langle \text{expr} \rangle$ ).

# Wywody i drzewa wywodu

Rozważmy gramatykę

$$E \rightarrow E + E \mid E - E \mid T$$

$$T \rightarrow 0 \mid 1 \dots \mid 9$$

Wyrażenie  $1 - 2 + 3$  może być wywiedzione:

$$E \Rightarrow E - E$$

$$\Rightarrow T - E$$

$$\Rightarrow 1 - E$$

$$\Rightarrow 1 - E + E$$

$$\Rightarrow 1 - T + E$$

$$\Rightarrow 1 - T + T$$

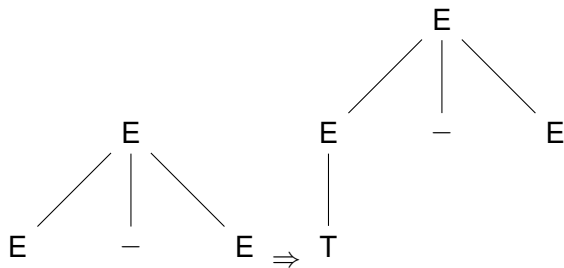
$$\Rightarrow 1 - 2 + T$$

$$\Rightarrow 1 - 2 + 3$$

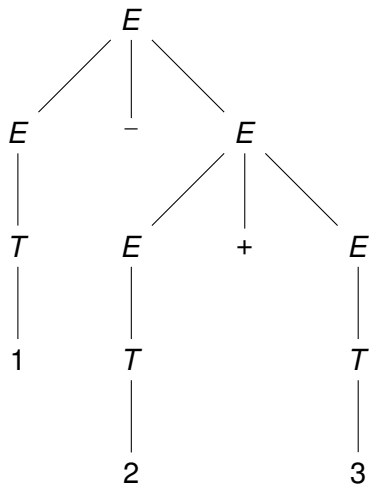
Jest to wywód lewostronny — zawsze przepisujemy najbardziej lewy nieterminal.

# Wywody i drzewa wywodu

Wywód możemy przedstawić jako drzewo:



# Wywody i drzewa wywodu



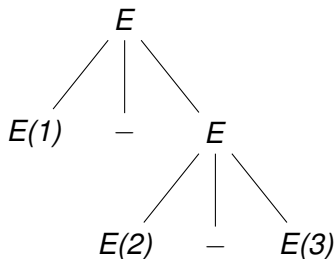
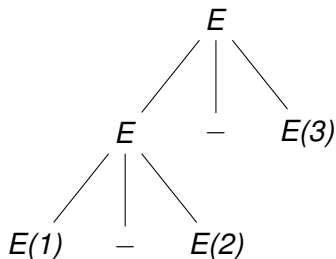
Drzewo wywodu reprezentuje *składnię konkretną*.

# Wieloznaczność

Rozważmy gramatykę:

$$E ::= E - E \mid n$$

Istnieją dwa możliwe wywody 1-2-3:



- Problem: który wywód jest ‘poprawny’?
- ‘-’ zwykle łączy w lewo, więc poprawny jest pierwszy wywód.
- Jak możemy uwzględnić ten fakt? Może trzeba nieco zmienić gramatykę?



# Łączność

- Możemy “usztynić” gramatykę, tak aby istniał tylko jeden wywód; w przypadku

$$E ::= E - E \mid T$$

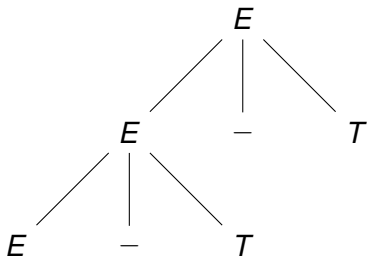
możemy wybrać albo prawe albo lewe  $E$  dla drugiego – w naszym przykładzie.

- Chcemy wymusić aby wywód “wyliczył” wszystkie minusy w kolejności występowania

$$E ::= E - T \mid T$$

Otrzymujemy następujące wyprowadzenie:

$$E \Rightarrow E - T \Rightarrow E - T - T \Rightarrow T - T - T \Rightarrow \dots$$



Dla operatora wiążącego w prawo, np potęgowania, moglibyśmy napisać:

$$E ::= T^E \mid T$$

# Priorytety operatorów

Rozważmy:

$$\begin{aligned} E &::= E + T \\ &\quad | E * T \\ &\quad | T \\ &\quad \dots \end{aligned}$$

Jak możemy opisać (w gramatyce), że “\*” wiąże silniej niż “+”?

Symbol o niższym priorytecie musi być “bliżej” symbolu startowego:

$$\begin{aligned} E &::= E + T \mid T \\ T &::= T * F \mid F \\ F &::= \dots \end{aligned}$$

# Wieloznaczność if-then-else

Rozważmy gramatykę:

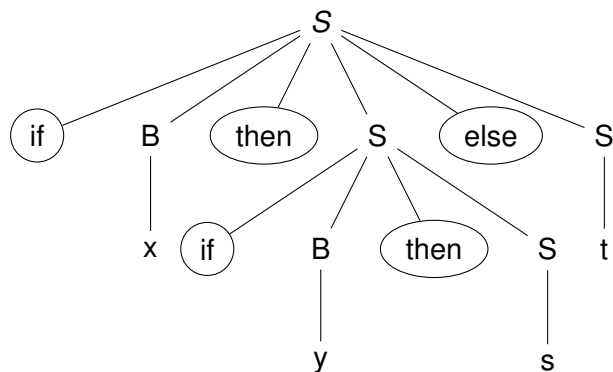
```
S ::= if B then S else S
      | if B then S
      | ...
```

wtedy konstrukcja

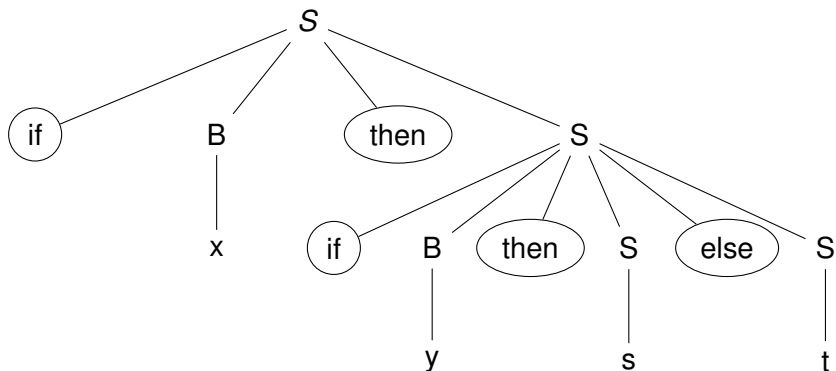
```
if x then if y then s else t
```

Ma dwa możliwe wywody:

# Wieloznaczność if-then-else



# Wieloznaczność if-then-else



# Wieloznaczność if-then-else

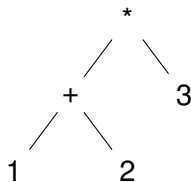
- Możemy rozwiązać problem zmieniając składnię języka:

```
S ::= if B then S else S fi
    | if B then S fi
    | ...
```

- Alternatywnie, możemy rozwiązać problem ad hoc mówiąc, że `else S` zawsze łączy się z najbliższym `if`.
- Można to wyrazić w gramatyce, ale jest to nieco kłopotliwe.

# Składnia abstrakcyjna

- Drzewa wywodu są czasem nazywane *drzewami składni konkretnej*
- Zawierają informacje zbędne na dalszych etapach kompilacji
- Zamiast tego potrzebujemy drzew struktury zawierających tylko *informacje semantyczne* (opisujące znaczenie programu).
- Drzewo struktury dla  $(1 + 2) * 3$ :



Nawiasy należą do składni konkretnej, ale nie do abstrakcyjnej.



# Składnia abstrakcyjna

Składnia abstrakcyjna to zbiór typów pozwalających reprezentować strukturę programu. Abstrahujemy od elementów takich jak nawiasy, znaki przestankowe, ...

```
Stmt = SWhile Exp Stmt  
      | SAssign Var Stmt  
      | SReturn Exp  
      | SBlock [Stmt]
```

# Analiza syntaktyczna

- Analizator syntaktyczny (*parser*) jest funkcją sprawdzającą, czy dane słowo należy do języka i, jeśli tak, budującą drzewo struktury.
- Algorytm Youngera:  $\mathcal{O}(n^3)$  i nie daje drzewa struktury.
- Istnieją efektywne algorytmy dla pewnych klas gramatyk.

Dwa zasadnicze podejścia:

- Top-down: próbujemy sparsować określoną konstrukcję (nieterminal); drzewo struktury budowane od korzenia do liści.
- Bottom-up: W danym napisie znajdujemy możliwe konstrukcje; drzewo budowane od liści do korzenia ze znalezionych kawałków.

# Analiza top-down

Schemat analizy top-down możemy zapisać jako automat:

- Jeden stan, alfabet stosowy  $\Gamma = N \cup T$ , akceptacja pustym stosem.
- Na szczycie stosu  $a \in T$ :
  - ▶ jeśli na wejściu  $a$  — zdejmij ze stosu, wczytaj następny symbol.
  - ▶ wpp — błąd: oczekiwano  $a$ .
- Na szczycie stosu  $A \in N$ , na wejściu  $a$ :
  - ▶ wybieramy produkcję  $A \rightarrow \alpha$  taką, że  $a \in SELECT(A \rightarrow \alpha)$
  - ▶ na stosie zastępujemy  $A$  przez  $\alpha$

Powyzszy automat ogląda jeden symbol z wejścia, ale łatwo go uogólnić na większą ich liczbę — automat LL(k).

Dla automatu deterministycznego, wybór produkcji jest ważny; zbiór symboli dla których wybieramy produkcję  $A \rightarrow \alpha$  nazywamy  $SELECT(A \rightarrow \alpha)$ . Teraz zajmiemy się obliczaniem tego zbioru.

# Zbiory *FIRST*

## Notacja

Niech  $w \in T^*$

$$k : w = \begin{cases} a_1 a_2 \dots a_k, & \text{jeśli } w = a_1 a_2 \dots a_k v \\ w\#, & \text{jeśli } |w| < k. \end{cases}$$

(pierwszych  $k$  znaków słowa  $w$ )

## Definicja (*FIRST*)

Niech  $w \in (T \cup N)$ .

$$FIRST_k(w) = \{\alpha : \exists \beta \in T^*, w \rightarrow^* \beta, \alpha = k : \beta\}$$

(pierwsze  $k$  znaków słów wyprowadzalnych z  $w$ ).

$$FIRST(w) = FIRST_1(w)$$

# Zbiory *FOLLOW*

## Definicja (*FOLLOW*)

Niech  $w \in N$

$$FOLLOW_k(w) = \{\alpha : \exists \beta \in T^*, S \rightarrow^* \mu w \beta, \alpha = k : \beta\}$$

(pierwsze  $k$  znaków mogących wystąpić za  $w$ ).

# Gramatyki LL(k)

Czytając od Lewej, Lewostronny wywód, widzimy ( $k$ ) symboli.

## Definicja

Gramatyka jest LL( $k$ ), jeśli dla każdego lewostronnego wyprowadzenia

$$S \rightarrow^* wA\alpha \rightarrow w\beta\alpha \rightarrow^* wx$$

$$S \rightarrow^* wA\alpha \rightarrow w\gamma\alpha \rightarrow^* wy$$

takich, że  $FIRST_k(x) = FIRST_k(y)$ , mamy  $\beta = \gamma$

“Jeżeli pierwszych  $k$  symboli wyprowadzalnych z  $A$  jest wyznaczone jednoznacznie, to także jednoznaczne jest, która produkcja dla  $A$  musi być zastosowana w wyprowadzeniu lewostronnym.”

# Gramatyki silnie LL(k)

## Definicja

$$SELECT_k(A \rightarrow \alpha) = FIRST_k(\alpha \cdot FOLLOW_k(A))$$

$$SELECT(A \rightarrow \alpha) = SELECT_1(A \rightarrow \alpha)$$

## Definicja

*Gramatyka jest silnie LL(k), jeśli dla każdej pary (różnych) produkcji  $A \rightarrow \alpha$ ,  $A \rightarrow \beta$  ich zbiory SELECT są rozłączne.*

- Dla gramatyk silnie LL(k) łatwo zbudować parser top-down.
- Każda gramatyka silnie LL(k) jest też LL(k).
- Każda gramatyka LL(1) jest silnie LL(1).

# Problemy

Gramatyka nie jest LL(1) jeśli zawiera zbiory produkcji postaci

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$

lub produkcje postaci

$$A \rightarrow A\beta$$

W drugim przypadku parser się nie zatrzyma!

Gramatykę, w której występują te problemy możemy często przekształcić do równoważnej gramatyki LL(1).



# Lewostronna faktoryzacja

Problem pierwszego rodzaju możemy rozwiązać “wyłączając przed nawias” wspólne początki produkcji:

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$

zastępujemy przez

$$A \rightarrow \alpha Z$$

$$Z \rightarrow \beta \mid \gamma$$

gdzie  $Z$  jest świeżym nieterminalem.

# Eliminacja lewostronnej rekursji

Zbiór produkcji

$$A \rightarrow A\alpha \mid \beta$$

zastępujemy

$$A \rightarrow \beta R$$

$$R \rightarrow \alpha R \mid \varepsilon$$

Na przykład, dla gramatyki

$$E \rightarrow E + T \mid T$$

otrzymujemy gramatykę

$$E \rightarrow TR$$

$$R \rightarrow +TR$$

lub, prościej

$$E \rightarrow T + E \mid T$$

Niestety, teraz '+' łączy w prawo, a nie jak chcieliśmy — w lewo.

# Wyliczanie FIRST

Dla  $t \in T$  mamy  $FIRST(t) = \{t\}$ .

Dla  $A \in N$  mamy:

$A \rightarrow \alpha_1 \mid \dots \mid \alpha_n \Rightarrow FIRST(A) \supseteq FIRST(\alpha_1) \cup \dots \cup FIRST(\alpha_n)$

Dla  $A \rightarrow X_1 \dots X_n$

- $FIRST(A) \supseteq FIRST(X_1) \setminus \{\#\}$
- $X_1 \rightarrow^* \varepsilon \Rightarrow FIRST(A) \supseteq FIRST(X_2) \setminus \{\#\}$
- $X_1 X_2 \rightarrow^* \varepsilon \Rightarrow FIRST(A) \supseteq FIRST(X_3)$
- ...
- $X_1 X_2 \dots X_n \rightarrow^* \varepsilon \Rightarrow \# \in FIRST(A)$

Prosty algorytm: zgodnie z powyższymi regułami powiększamy zbiory FIRST tak długo, jak któryś ze zbiorów się powiększa (obliczamy najmniejszy punkt stały).

# Wyliczanie *FOLLOW*

Dla każdych  $A, X \in N$ ,  $a \in T$ ,  $\alpha, \beta \in (N \cup T)^*$  mamy:

- $A \rightarrow \alpha X a \beta \in P$ , to  $a \in FOLLOW(X)$ .
- $A \rightarrow \alpha X \in P$ , to  $FOLLOW(A) \subseteq FOLLOW(X)$
- $A \rightarrow \alpha X \beta \in P$ , to  $FIRST(\beta) \setminus \{\#\} \subseteq FOLLOW(X)$
- $A \rightarrow \alpha X \beta \in P$ ,  $\beta \rightarrow^* \varepsilon$  to  $FOLLOW(A) \subseteq FOLLOW(X)$
- $\# \in FOLLOW(S)$  dla symbolu startowego  $S$ .

Prosty algorytm: zgodnie z powyższymi regułami powiększamy zbiory *FOLLOW* tak długo, jak któryś ze zbiorów się powiększa (obliczamy najmniejszy punkt stały).

# Przykład

$$E \rightarrow E + T \quad \text{SELECT}(E \rightarrow E + T) = \text{FIRST}(E) = \text{FIRST}(T)$$

$$E \rightarrow T \quad \text{SELECT}(E \rightarrow T) = \text{FIRST}(T) = \text{FIRST}(F)$$

---

$$T \rightarrow T * F \quad \text{SELECT}(T \rightarrow T * F) = \text{FIRST}(T) = \text{FIRST}(F)$$

$$T \rightarrow F \quad \text{SELECT}(T \rightarrow F) = \text{FIRST}(F) = \{ (, \mathbf{a} \}$$

---

$$F \rightarrow (E)$$

$$F \rightarrow \mathbf{a}$$

Gramatyka nie jest LL(1).

# Transformacja do postaci LL(1)

Usuwamy lewostronną rekursję:

$$E \rightarrow TE' \quad E' \rightarrow +TE' \mid \varepsilon$$

$E \rightarrow TE'$	niepotrzebne SELECT
<hr/>	
$E' \rightarrow +TE'$	$\text{SELECT}(E' \rightarrow +TE') = \{+\}$
$E' \rightarrow \varepsilon$	$\text{SELECT}(E' \rightarrow \varepsilon) = \text{FOLLOW}(E') =$ $= \text{FOLLOW}(E) = \{), \#\}$
<hr/>	
$T \rightarrow FT'$	
<hr/>	
$T' \rightarrow *FT'$	$\text{SELECT}(T' \rightarrow *FT') = \{*\}$
$T' \rightarrow \varepsilon$	$\text{SELECT}(T' \rightarrow \varepsilon) = \text{FOLLOW}(T') = \text{Follow}(T)$ $= \text{FIRST}(E' \cdot \text{Follow}(E')) = \{+, ), \#\}$
<hr/>	
$F \rightarrow (E)$	$\text{SELECT}(F \rightarrow (E)) = \{( \}$
$F \rightarrow \mathbf{a}$	$\text{SELECT}(F \rightarrow \mathbf{a}) = \{\mathbf{a}\}$

# Dla miłośników algorytmiki

Efektywny algorytm wyliczania FIRST/FOLLOW:

- 1 Budujemy graf  $G$  odpowiedniej relacji na  $N$ : dla FIRST “bycia na początku” produkcji, dla FOLLOW — “bycia na końcu”.
- 2 Obliczamy acykliczną kondensację  $G$  (graf silnie spójnych składowych).
- 3 Sortujemy topologicznie.
- 4 Obliczamy przybliżenia odpowiednich zbiorów (tak jak w jednym obrocie pętli “naiwnego” algorytmu).
- 5 Dla każdej silnie spójnej składowej przybliżenie rozszerzamy do sumy przybliżeń dla wszystkich jej wierzchołków.
- 6 Wynikowy zbiór jest sumą przybliżeń dla wszystkich poprzedzających składowych i bieżącej składowej.