AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

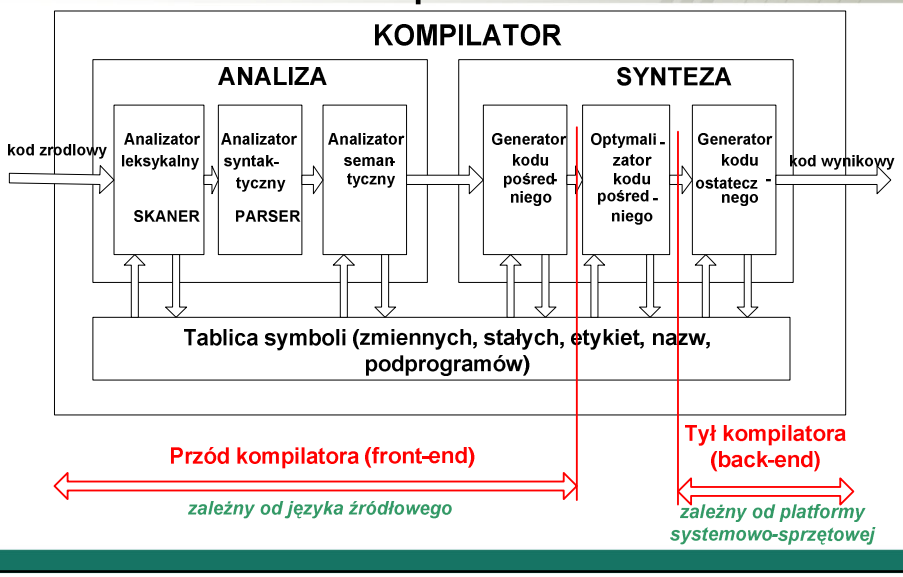
Analiza syntaktyczna

Teoria kompilacji

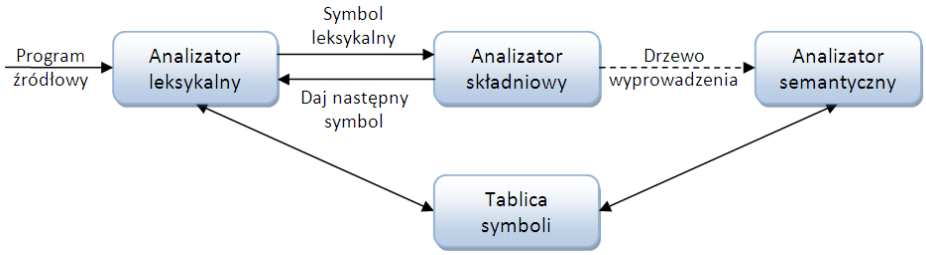
Dr inż. Janusz Majewski
Katedra Informatyki

AGH

Przypomnienie: struktura kompilatora



Współpraca parsera z innymi analizatorami



Analizator składniowy (parser) jest modułem wykonującym analizę syntaktyczną oraz zarządzającym całym przodem kompilatora, a w szczególności całym blokiem analizy. W takt pracy parsera działa zarówno analizator leksykalny (dostarczając parserowi kolejne tokeny), jak i analizator semantyczny (analizując np. poprawność typów) oraz generator kodu pośredniego (tłumacząc zanalizowane fragmenty programu do kodu pośredniego).

Zadanie analizy syntaktycznej

Analizator syntaktyczny (parser) ma za zadanie dokonać rozbioru gramatycznego tekstu źródłowego analizowanego programu. Analizator leksykalny (skaner) dokonuje zamiany tekstu wejściowego na ciąg tokenów. Parser pracuje na podstawie gramatyki syntaktycznej. Buduje on drzewo rozbioru analizowanego ciągu tokenów, czyli przeprowadza wyprowadzenie łańcucha tokenów w gramatyce składniowej. Od parsera oczekujemy jakiegoś zanotowania tego wywodu. Często stosowanym sposobem notowania wyprowadzenia jest podanie ciągu numerów produkcji, na podstawie którego można odtworzyć drzewo rozbioru.

Zadanie analizy syntaktycznej c.d.

Niekiedy równolegle z analizą syntaktyczną parser lub inicjuje wykonanie:

- pewnych akcji semantycznych, jak np. będącej czynnością o charakterze analitycznym kontroli typów, i/lub
- pewnych akcji o charakterze syntetycznym, jak np. generowania kodu pośredniego w postaci ONP, tetrad (n-tek), drzew składniowych itd.

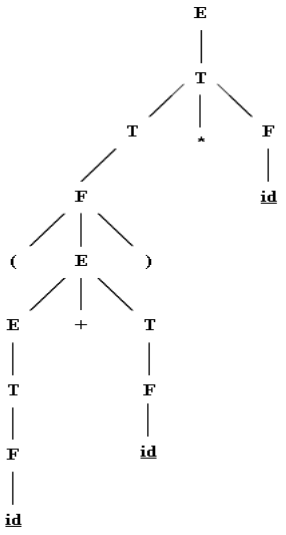
Działanie parsera:

- We: $\omega \in \Sigma_S^*$ (ω jest słowem nad alfabetem Σ_S, Σ_S jest zbiorem tokenów)
- Wy: $\pi = p_1 \ldots p_i \ldots p_n$ (p_i—numer produkcji) lub err (błąd syntaktyczny)

Analiza składniowa

Gramatyka składniowa:

1) E→E+T
2) E→T
3) T→T*F
4) T→F
5) F→(E)
6) F→id



Analizowane słowo: (id+id)*id
Do zbudowania drzewa rozbioru syntakt. (podstawa konstrukcji kodu wynikowego) niezbędna jest wiadomość, że ciąg numerów produkcji związany jest z wywodem (tutaj) lewostronnym tworzonym metodą top-down

E
(2) ⇒ T
(3) ⇒ T*F
(4) ⇒ F*F
(5) ⇒ (E)*F
(1) ⇒ (E+T)*F
(2) ⇒ (T+T)*F
(4) ⇒ (F+T)*F
(6) ⇒ (id+T)*F
(4) ⇒ (id+F)*F
(6) ⇒ (id+id)*F
(6) ⇒ (id+id)* id ⇒ acc

ciąg numerów produkcji jest wyjściem z parsera

Tworzenie parsera

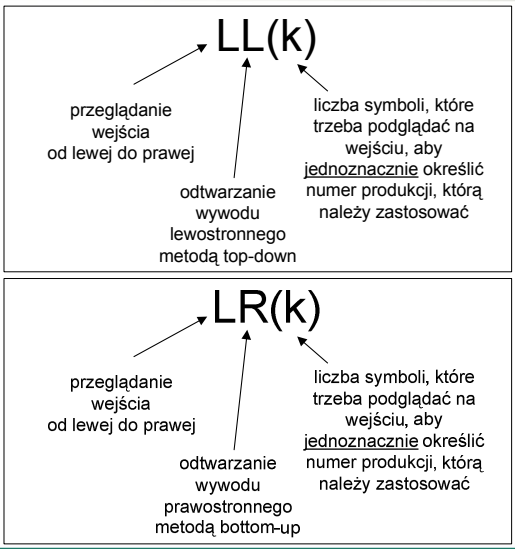
Generacja parsera:

- We: gramatyka syntaktyczna $G_S=<V_S,\Sigma_S,P_S,S_S>$ (oznaczana dalej $G=<V,\Sigma,P,S>$)
- Wy: algorytm parsera

Algorytm parsera musi być efektywny. Uniwersalny algorytm Earley'a ma dla gramatyki bezkontekstowej (bez ograniczeń na jednoznaczność) złożoność obliczeniową: czasową $O(n^3)$, pamięciową $O(n^2)$, zaś dla gramatyk jednoznacznych złożoność czasowa wynosi $O(n^2)$. Są to złożoności niedopuszczalne w praktyce. Algorytm Cocke'a-Youngera-Kasamiego (CYK) (także bez ograniczeń na jednoznaczność) ma złożoność obliczeniową: czasową $O(n^3)$, pamięciową $O(n^2)$. Musimy się posługiwać algorytmami prostszymi, ale wiąże się to z zawężeniem klasy gramatyk.

Dalej rozpatrywane będą algorytmy parsingu dla gramatyk LL(k) i LR(k). Dla gramatyk LL(k) i LR(k) złożoność obliczeniowa algorytmu parsera wynosi: czasowa O(n) oraz pamięciowa O(n).

Gramatyki LL i LR



Na ogół stosowane są gramatyki:

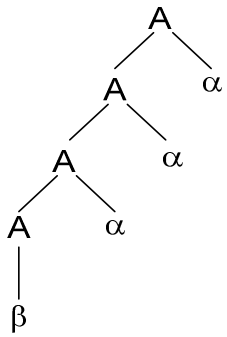
- LL(1)
- LR(1) – oraz jej podklasy: SLR(1) lub LALR(1), dające możliwość znacznego uproszczenia algorytmu parsingu przy niewielkim ograniczeniu „możliwości" gramatyki.

Usuwanie lewostronnej rekurencji

Mówimy, że gramatyka jest lewostronnie rekurencyjna, jeśli ma taki nieterminal A, że istnieje wyprowadzenie $A \Rightarrow^{+} A\alpha$ dla pewnego niepustego łańcucha α.

Jeśli mamy produkcje:

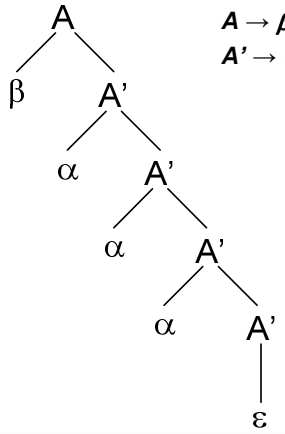
$A \rightarrow A\alpha \mid \beta$



To możemy je zastąpić przez:

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$



Przykład

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

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

1) E → E+T	1) E → TE’
2) E → T	2) E’ → +TE’
	3) E’ → ε
3) T→T*F	4) T → FT’
4) T→F	5) T’ → *FT’
	6) T’ → ε
5) F→(E)	7) F → (E)
6) F→id	8) F → id

Zbiory $FIRST_k$

Zbiór $FIRST_k(\alpha)$ jest zbiorem wszystkich terminalnych przedrostków długości *k* (lub mniejszej, jeżeli z α wyprowadza się łańcuch terminalny krótszy niż *k*) łańcuchów, które mogą być wyprowadzalne z α

Formalnie:

Niech $G = < V, \Sigma, P, S > \in \mathcal{G}_{BK}$

Definiujemy:

$$FIRST_K(\alpha) = \{x \in \Sigma^* \mid (\alpha \Rightarrow^* x\beta \wedge |x|=k) \vee (\alpha \Rightarrow^* x \wedge |x|<k);\ \alpha,\beta \in (V \cup \Sigma)^*\}$$

Zbiory $FOLLOW_k$

Zbiór $FOLLOW_k(\beta)$ zawiera terminalne łańcuchy o długości *k* (lub mniejszej—jak poprzednio), które mogą pojawić się w wyprowadzeniach jako następniki β. Uwaga: jeśli $FOLLOW_k(\beta)$ zawiera *x*, $|x|<k$, to wówczas zastępujemy *x* przez *x$*, gdzie *$*—prawy ogranicznik słowa (dla wygody).

Formalnie:

Niech $G = < V, \Sigma, P, S > \in \mathcal{G}_{BK}$

Definiujemy:

$$FOLLOW_K(\beta) = \{x \in \Sigma^*: (S \Rightarrow^* \alpha\beta\gamma \wedge x \in FIRST_K(\gamma);\ \alpha,\beta,\gamma \in (V \cup \Sigma)^*\}$$

Wyznaczanie $FIRST_1(x)$ dla $x \in (V \cup \Sigma)$

```
1)  if x∈Σ then FIRST1(x):={x};
2)  if (x→ε)∈P then FIRST1(x):= FIRST1(x)∪{ε};
3)  if x∈V and (x→y1y2...yk)∈P then
begin
    FIRST1(x):=FIRST1(x)∪(FIRST1(y1)—{ε});
    for i:=2 to k do
      if (ε∈FIRST1(y1))and...and(ε∈FIRST1(yi-1))
      then FIRST1(x):=FIRST1(x)∪(FIRST1(yi)—{ε});
    if (ε∈FIRST1(y1))and...and(ε∈FIRST1(yk)) then
      FIRST1(x):=FIRST1(x)∪{ε}
end;
```

Aby wyznaczyć FIRST1(x) dla wszystkich $x \in (V \cup \Sigma)$ należy stosować reguły (1), (2) i (3) tak długo, aż nic nowego nie da się dołączyć do któregokolwiek zbioru $FIRST_1(x)$.

Przykład (1)

1) $E \rightarrow TE'$
2) $E' \rightarrow +TE'$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$
5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$

} $FIRST_1(F)=\{(, \underline{id}\}$

Przykład (2)

1) $E \rightarrow TE'$
2) $E' \rightarrow +TE'$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$ → $FIRST_1(T)=FIRST_1(F)=\{(, \underline{id}\}$
5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$

} $FIRST_1(F)=\{(, \underline{id}\}$

Przykład (3)

1) $E \rightarrow TE'$ → $FIRST_1(E)=FIRST_1(T)=\{(, \underline{id}\}$
2) $E' \rightarrow +TE'$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$ → $FIRST_1(T)=FIRST_1(F)=\{(, \underline{id}\}$
5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$

} $FIRST_1(F)=\{(, \underline{id}\}$

Przykład (4)

1) $E \rightarrow TE'$ → $FIRST_1(E)=FIRST_1(T)=\{(, \underline{id}\}$

2) $E' \rightarrow +TE'$
3) $E' \rightarrow \varepsilon$
} $FIRST_1(E')=\{+, \varepsilon\}$

4) $T \rightarrow FT'$ → $FIRST_1(T)=FIRST_1(F)=\{(, \underline{id}\}$

5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
} $FIRST_1(T')=\{*, \varepsilon\}$

7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$
} $FIRST_1(F)=\{(, \underline{id}\}$

Wyznaczanie $FIRST_1(x_1x_2...x_n)$

$FIRST_1(x_1x_2...x_n) := FIRST_1(x_1) - \{\varepsilon\}$

for i:=2 **to** k **do**

  if $(\varepsilon \in FIRST_1(x_1))$ **and** ... **and** $(\varepsilon \in FIRST_1(x_{i-1}))$
  then

    $FIRST_1(x_1...x_n) :=$
    $FIRST_1(x_1...x_n) \cup (FIRST_1(x_i) - \{\varepsilon\})$;

if $(\varepsilon \in FIRST_1(x_1))$ **and** ... **and** $(\varepsilon \in FIRST_1(x_n))$
  then $FIRST_1(x_1...x_n) := FIRST_1(x_1...x_n) \cup \{\varepsilon\}$;

Wyznaczanie zbiorów $FOLLOW_1$

Wyznaczenie $FOLLOW_1$ dla wszystkich $A \in V$

Stosować poniższe reguły (1), (2), (3) dopóty nic nowego nie może już zostać dodane do żadnego zbioru $FOLLOW_1$.

1) $FOLLOW_1(S) := FOLLOW_1(S) \cup \{\$\}$

2) **if** $(A \to \alpha B \beta) \in P$ **then**
   $FOLLOW_1(B) := FOLLOW_1(B) \cup (FIRST_1(\beta) - \{\varepsilon\})$

3) **if** $(A \to \alpha B) \in P$ **or** $((A \to \alpha B \beta) \in P$ **and** $(\varepsilon \in FIRST_1(\beta)))$ **then**
   $FOLLOW_1(B) := FOLLOW_1(B) \cup FOLLOW_1(A)$;

Przykład (5)

1) E → TE' ⟶ $FOLLOW_1(E) \supseteq \{\$\}$
2) E' → +TE'
3) E' → ε
4) T → FT'
5) T' → *FT'
6) T' → ε
7) F → (E)
8) F → id

Przykład (6)

1) $E \rightarrow TE'$ → $FOLLOW_1(E) \supseteq \{\$\}$
2) $E' \rightarrow +TE'$ → $FOLLOW_1(T) \supseteq \{+\}$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$ → $FOLLOW_1(F) \supseteq \{*\}$
5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$ → $FOLLOW_1(E) \supseteq \{)\}$
8) $F \rightarrow \underline{id}$

Czyli w tej chwili:
$FOLLOW_1(E) \supseteq \{\$,)\}$
$FOLLOW_1(T) \supseteq \{+\}$
$FOLLOW_1(F) \supseteq \{*\}$

Przykład (7)

1) $E \rightarrow TE'$ → $FOLLOW_1(E') \leftarrow FOLLOW_1(E)$
2) $E' \rightarrow +TE'$ → $FOLLOW_1(T) \leftarrow FOLLOW_1(E)$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$ → $FOLLOW_1(T') \leftarrow FOLLOW_1(T)$
5) $T' \rightarrow *FT'$ → $FOLLOW_1(F) \leftarrow FOLLOW_1(T)$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$

Poprzednio:
$FOLLOW_1(E) \supseteq \{\$,)\}$
$FOLLOW_1(T) \supseteq \{+\}$
$FOLLOW_1(F) \supseteq \{*\}$

Ostatecznie:
$FOLLOW_1(E)=\{\$,)\}$
$FOLLOW1(E')=\{\$,)\}$
$FOLLOW_1(T)=\{+, \$,)\}$
$FOLLOW_1(T')=\{+, \$,)\}$
$FOLLOW_1(F)=\{*, +, \$,)\}$

Lewostronna faktoryzacja

Jeżeli produkcje gramatyki mające ten sam symbol po lewej stronie posiadają prawe strony rozpoczynające się od tego samego przedrostka, to można dla nich wykonać przekształcenie zwane lewostronną faktoryzacją. Polega ona na zamianie produkcji postaci:

$$A \rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_k$$

na produkcje:

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 \mid \ldots \mid \beta_k$$