Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Włodzimierz Bielecki, Piotr Błaszyński

Projektowanie kompilatorów

Teoria i praktyka

Skrypt do wykładów i zajęć laboratoryjnych

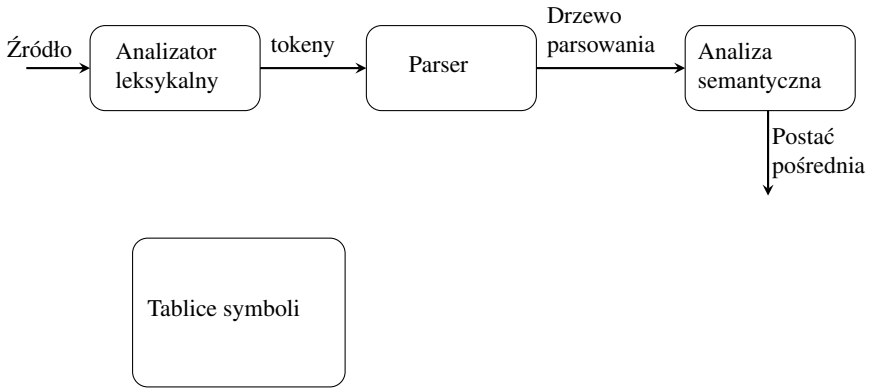
Szczecin 2018

Część I Wykłady

1. Pojęcie gramatyki, drzewo parsowania, łączność operatorów, gramatyka jednoznaczna

Składnia języka programowania. Opisuje ona właściwą strukturę programu, natomiast semantyka języka określa co program robi – czyli jaki jest jego sens.

Struktura kompilatora. Każdy kompilator ma przód i tył; struktura przodu kompilatora jest pokazana na rysunku 1.1.



Rys. 1.1. Struktura przodu kompilatora

Języki programowania

$$1 + 2 * 3 = 7 \tag{1.1}$$

$$1 + *23 = ??? \tag{1.2}$$

Czy ciągi 1.1 i 1.2 są poprawnie zbudowanym wyrażeniem arytmetycznym? Do odpowiedzi na to pytanie potrzebna jest gramatyka, czyli zbiór produkcji.

Pojęcie produkcji. Produkcja jest to para uporządkowana, na przykład: $S \to 1$.

Gramatyka. Jest to zbiór produkcji:

$$S \to AB \tag{1.3}$$

$$A \to 1 \tag{1.4}$$

$$A \to A1 \tag{1.5}$$

$$B \to 0 \tag{1.6}$$

$$B \to B0 \tag{1.7}$$

S (wzór 1.3) jest symbolem początkowym. Symbole nieterminalne $N = S, A, B$ występują po prawej stronie produkcji; mogą wystąpić również po lewej stronie produkcji (ale nie muszą). Symbole terminalne $T = 0, 1$ występują tylko po prawej stronie produkcji.

Wyprowadzenia:
- na podstawie 1.3 : $S \rightarrow AB$
- na podstawie 1.4 : $AB \rightarrow 1B$
- na podstawie 1.6 : $1B \rightarrow 10$

czyli: $S \rightarrow 10$, więc z S można wyprowadzić 10, stosując jedną produkcję lub większą liczbę produkcji.

Specyfikacja BNF. Backus-Naur Form (BNF) jest formą używaną do wyrażenia gramatyk bezkontekstowych. Nazwa wywodzi sie od nazwisk naukowców zajmujących się opisem gramatyk języków programowania; byli to John Warner Backus i Peter Naur. Specyfikacja BNF jest to zbiór produkcji o postaci: *symbol* → *expression*, gdzie *symbol* jest to nieterminal, natomiast *expression* jest to sekwencja jednego symbolu lub większej liczby symboli terminalnych i/lub nieterminalnych. Większą liczbę sekwencji oddzielamy kreską pionową '|', wskazując wybór. Symbole, które nigdy nie pojawiają się po lewej stronie produkcji, są to **terminale** (są one pogrubione). Symbole pojawiające się po lewej stronie produkcji są to **nieterminale** (są one wyróżnione kursywą). Przykład produkcji: *stmt* → if(*expr*) *stmt else stmt*.

Gramatyka bezkontekstowa (*A context-free grammar*) zawiera:
1. Zbiór symboli terminalnych (terminali). Terminale są to elementarne symbole języka zdefiniowanego przez gramatykę.
2. Zbiór symboli nieterminalnych (zmienne syntaktyczne, nieterminale). Każdy nieterminal reprezentuje sekwencję terminali w sposób, który poznamy w dalszej części.
3. Zbiór produkcji, gdzie każda produkcja składa się z nieterminala, zwanego głową lub lewą stroną produkcji, ze strzałki oraz z sekwencji terminali i/lub nieterminali, nazywanej ciałem lub prawą stroną produkcji.
4. Jeden wyznaczony symbol nieterminalny zwany symbolem startowym.

Gramatyka jest to skończony zbiór produkcji, w którym lewa strona pierwszej produkcji wskazuje symbol startowy. Przykład gramatyki:

```
list -> list + digit                          (1)
list -> list - digit                          (2)
list -> digit                                 (3)
digit -> 0 | 1| 2| 3| 4| 5| 6| 7| 8| 9        (4)
```

Ciała trzech produkcji, których lewa strona jest tym samym symbolem list, można pogrupować:

```
   list -> list + digit | list - digit | digit
```

Według przedstawionej definicji symbole terminalne są następujące: $+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ Nieterminalami są *list* i *digit*. Symbolem startowym jest *list*. Przykład gramatyki – zapis formalny :

```
G = < {list,digit}, {+,-,0,1,2,3,4,5,6,7,8,9}, P, list >
Z produkcjami P =
 list -> list + digit
 list -> list - digit
 list -> digit
 digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  8 | 9
```

Ciąg symboli (napis) jest to sekwencja składająca się z zera lub większej liczby symboli. Ciąg, który nie zawiera żadnego symbolu, jest nazywany napisem pustym ε.

Wyprowadzenia. Korzystając z gramatyki, wyprowadzamy napisy, zaczynając zawsze od symbolu startowego, i wielokrotnie zastępujemy pojedynczy nieterminal prawą stroną produkcji, której lewa strona jest zastępowanym nieterminalem. Dla gramatyki:

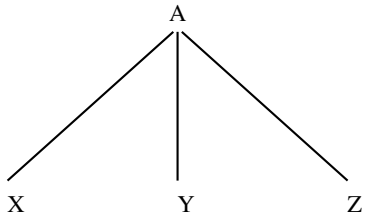
```
list -> list + digit | list - digit | digit
digit -> 0 |1| 2| 3| 4| 5| 6| 7|  8| 9
```

możemy wyprowadzić:

```
list -> digit -> 0
list -> list + digit -> digit + digit -> 1 + digit -> 1+2
```

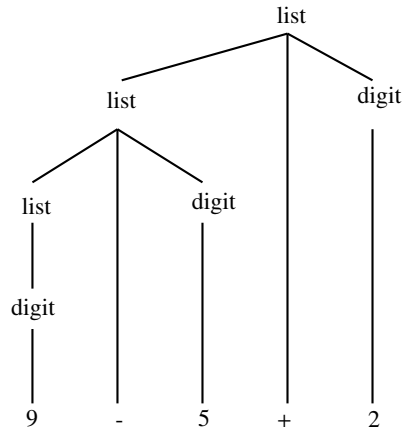
Parsowanie ma na wejściu ciąg terminali i „zastanawia się”, jak wyprowadzić ten ciąg z symbolu startowego gramatyki; jeśli nie można wyprowadzić takiego ciągu, to jest raportowany błąd składni. Drzewo parsowania pokazuje obrazowo, w jaki sposób z symbolu startowego gramatyki można wyprowadzić zdanie wejściowe.

Dla produkcji: $A \rightarrow XYZ$ drzewo parsowania ma postać jak na rysunku 1.2:



Rys. 1.2. Drzewo parsowania

Dla napisu 9 – 5 + 2 i gramatyki G drzewo parsowania ma postać jak na rysunku 1.3:



Rys. 1.3. Drzewo parsowania

Formalnie w przypadku gramatyki bezkontekstowej drzewo parsowania ma następujące właściwości:

1. Korzeń jest oznaczony symbolem startowym.
2. Każdy liść jest oznaczony przez terminal lub ε.
3. Każdy węzeł wewnętrzny jest oznaczony przez nieterminal.
4. Jeśli A jest nieterminalem, który oznacza pewien węzeł wewnętrzny, a $X1, X2, ..., Xn$ są etykietami dzieci tego węzła od lewej do prawej strony, to istnieje produkcja $A- > X1X2..Xn$, gdzie każde $X1, X2, ..., Xn$ oznacza symbol terminalny lub nieterminalny. W szczególnym przypadku, jeśli $A- > \varepsilon$ jest produkcją, to węzeł oznaczony jako A ma jedno dziecko ε.

Terminologia związana z drzewem. Drzewo składa się z jednego lub większej liczby węzłów. Węzły mogą mieć etykiety, które zazwyczaj są symbolami gramatycznymi. Gdy rysujemy drzewo, często reprezentujemy węzły tylko przez te etykiety. Dokładnie jeden węzeł jest korzeniem. Wszystkie węzły, oprócz korzenia, mają unikatowego rodzica; korzeń nie ma rodzica. Gdy rysujemy drzewo, to umiejscawiamy rodzica zawsze powyżej dziecka; rodzic i dziecko są połączone krawędzią.

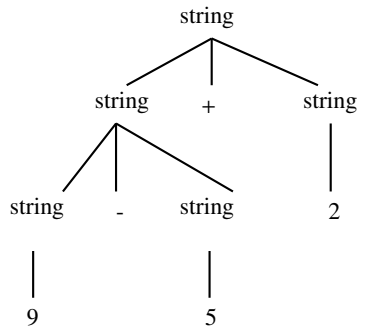
- Jeśli węzeł N jest rodzicem węzła M, to M jest dzieckiem N.
- Dzieci jednego węzła nazywane są rodzeństwem.
- Węzeł bez dzieci jest to liść.
- Inne węzły – te z jednym dzieckiem lub z większą liczbą dzieci – są to węzły wewnętrzne.
- Potomkiem węzła N jest sam węzeł N lub dziecko N, lub dziecko dziecka itd.
- Mówimy, że węzeł N jest przodkiem węzła M, jeśli M jest potomkiem N.

Niejednoznaczność. Dla danej gramatyki, dla ciągu terminali może istnieć więcej niż jedno drzewo parsowania. Taka gramatyka jest nazywana gramatyką niejednoznaczną. Ponieważ napis, dla którego istnieje więcej niż jedno

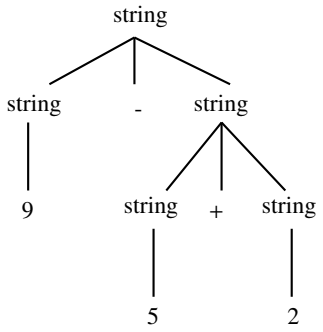
drzewo parsowania zwykle ma więcej niż jedno znaczenie, musimy zaprojektować gramatykę jednoznaczną lub używać gramatyk niejednoznacznych z dodatkowymi zasadami rozwiązywania niejednoznaczności. Korzystając z gramatyki:

```
string -> string + string | string - string| 0|1|2|3|4|5|6|7|8|9,
```

dla napisu 9 – 5 + 2 można utworzyć dwa drzewa parsowania (rysunki 1.4 i 1.5):



Rys. 1.4. Pierwsze drzewo parsowania



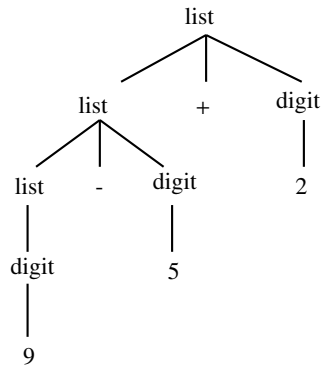
Rys. 1.5. Drugie drzewo parsowania

Łączność operatorów. Zgodnie z konwencją 9 + 5 + 2 jest równoważne z (9 + 5) + 2, natomiast 9 – 5 – 2 jest równoważne z (9 – 5) – 2. Gdy argument 5 ma operatory po jego lewej i prawej stronie, reguły są potrzebne do podjęcia decyzji, który z operatorów odnosi się do tego argumentu. Mówimy, że operator + jest łączny lewostronnie, ponieważ argument, który ma znak plus po obu jego stronach, należy do operatora po jego lewej stronie. W większości języków programowania cztery operatory arytmetyczne: dodawanie, odejmowanie, mnożenie i dzielenie są łączne lewostronnie. Niektóre operatory, takie jak potęgowanie, są łączne prawostronnie. Operator przypisania = też jest łączny prawostronnie, co oznacza, że wyrażenie $a = b = c$ traktuje się w taki sam sposób jak wyrażenie $a = (b = c)$. Ciągi takie, jak $a = b = c$, są generowane przez następującą gramatykę:

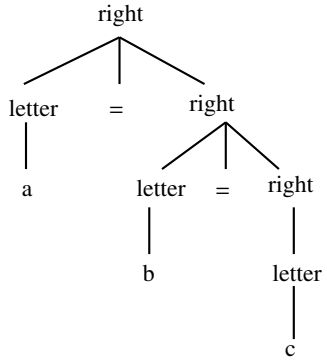
```
right -> letter = right | letter
letter -> a | b | ... | z
```

Według tych reguł zapis $a = b = c$ jest równoważny z zapisem $a = (b = c)$. Struktura drzew dla łączności lewostronnej i prawostronnej jest różna; w pierwszym przypadku drzewo rośnie – w dół i w lewo; w drugim przypadku – w dół i w prawo. Kontrast między drzewem parsowania dla operatora – łącznego lewostronnie i drzewem parsowania dla operatora = łącznego prawostronnie jest pokazany na rysunkach 1.6 i 1.7.

```
list -> list + digit || list - digit | digit
digit -> 0 | 1 | 2 | ... | 9
```



Rys. 1.6. Łączność operatorów lewostronna



Rys. 1.7. Łączność operatorów prawostronna

Według tych reguł zapis $9 - 5 + 2$ jest równoważny z zapisem $(9 - 5) + 2$.

Pierwszeństwa operatorów. Rozważmy wyrażenie $9 + 5 * 2$. Istnieją dwie możliwe interpretacje tego wyrażenia: $(9 + 5) * 2$ lub $9 + (5 * 2)$. Zasadę łączności stosuje się do wystąpień tego samego operatora, a więc nie rozwiązuje ona dwuznaczności. Gramatykę dla wyrażeń arytmetycznych można skonstruować na podstawie tabeli reprezentującej pierwszeństwa operatorów. Zaczynamy od czterech operatorów arytmetycznych i tabeli pierwszeństwa, pokazującej operatory w kolejności rosnącego priorytetu. Operatory na tej samej linii mają taką samą łączność i pierwszeństwo:

– łączne lewostronnie: + −
– łączne lewostronnie : ∗ /

Gramatyka jednoznaczna. Zasada tworzenia gramatyki jednoznacznej: należy dodatkowo wprowadzić $N + 1$ nieterminali, gdzie N jest to liczba poziomów pierwszeństwa. W naszym przykładzie $N = 2$. Tworzymy dwa nieterminale *expr* i *term* dla dwóch poziomów pierwszeństwa oraz dodatkowy nieterminal $factor$ do generowania podstawowych jednostek w wyrażeniach. Podstawowe jednostki w wyrażeniach są to cyfry i wyrażenia w nawiasach: $factor- > digit|(expr)$. Rozważmy teraz operatory binarne ∗ i /, które mają najwyższy priorytet. Odpowiednie produkcje mają postać:

```
term
    -> term * factor
     | term / factor
     | factor
```

Produkcje, które odpowiadają za wyrażenia z operatorami + i −, mają postać:

```
expr
   -> expr + term
    | expr - term
    | term
```

Poprzez połączenie powyższych gramatyk uzyskujemy następującą gramatykę jednoznaczną:

```
expr -> expr + term | expr - term | term
term -> term * factor | term / factor | factor
factor -> digit | ( expr )
```

Kroki użyte do przetwarzania wyrażenia 2 + 3 ∗ 5:

Krok 1:

```
factor -> digit | ( expr )
```

Krok 2:

```
term -> term * factor
    | term / factor
    | factor
```

Krok 3:

```
expr -> expr + term
    | expr - term
    | termactor
    | factor
```

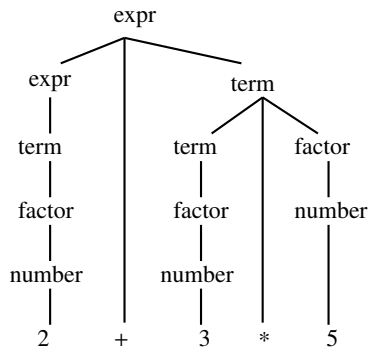
Krok 4:

```
expr -> expr + term | expr - term | term
term -> term * factor | term / factor | factor
factor -> digit | ( expr )
```

Użyte produkcje:

```
expr -> expr + term | term
term -> term * factor | factor
factor -> number | ( expr )
```

Zdanie 2+3*5 jest parsowane w sposób jednoznaczny – tak jak to przedstawiono na rysunku 1.8:



Rys. 1.8. Gramatyka jednoznaczna

2. Translacja sterowana składnią

Translacja sterowana składnią. Odbywa się poprzez dołączenie zasad (reguł) lub fragmentów kodu do produkcji w gramatyce. Na przykład w przypadku produkcji: $expr \rightarrow expr_1 + term$ możemy przetwarzać $expr$ wykorzystując strukturę produkcji zgodnie z poniższym pseudokodem:

```
1 dopasuj expr1;
2     przetwarzaj expr1;
3 dopasuj term;
4     przetwarzaj term;
5 dopasuj +;
6     przetwarzaj +;
```

Atrybut jest to pewna wartość powiązana z konstrukcją języka programowania. Przykładami atrybutów są typy danych wyrażeń, liczba instrukcji wygenerowanego kodu lub lokalizacja pierwszej instrukcji generowanego kodu.

Schemat translacji sterowany składnią jest to gramatyka, w przypadku której do każdej produkcji jest przypisany fragment kodu – akcja semantyczna. Fragmenty kodu są wykonywane, natomiast produkcja jest wykorzystywana przez parser.

Notacja postfiksowa może być zdefiniowana w następujący sposób:

1. Jeżeli E oznacza zmienną lub stałą, to notacją postfiksową dla E jest samo E.
2. Jeżeli E jest wyrażeniem postaci $E1\ op\ E2$, gdzie op jest operatorem binarnym, to notacją postfiksową dla E jest $E1'\ E2'\ op$, gdzie $E1'$ i $E2'$ są notacjami postfiksowymi odpowiednio dla $E1$ i $E2$.
3. Jeżeli E jest wyrażeniem o postaci $(E1)$, to notacja postfiksowa dla E jest taka sama jak notacja postfiksowa dla $E1$.

Przykład: Notacją postfiksową dla wyrażenia $(9 - 5) + 2$ jest $9\ 5\ -\ 2+$. Oznacza to, że przekład dla 9, 5 i 2 jest reprezentowany przez te same stałe, zgodnie z regułą (1). Tłumaczeniem $9 - 5$ zgodnie z regułą (2) jest $9\ 5-$. Tłumaczeniem dla $(9-5)$ zgodnie z regułą (3) jest $9-5$. Nawiasy nie są potrzebne w notacji postfiksowej, ponieważ pozycja i liczba argumentów operatorów w tej notacji w sposób jednoznaczny określają kolejność wykonywania operatorów.

Atrybuty syntezowane. Atrybuty kojarzymy z nieterminalami i terminalami. Żeby obliczyć atrybuty nieterminali, dodajemy reguły do produkcji. Reguły te opisują, w jaki sposób obliczane są atrybuty w węzłach drzewa parsowania.

Definicje przekładu sterowanego składnią formułuje się poprzez:

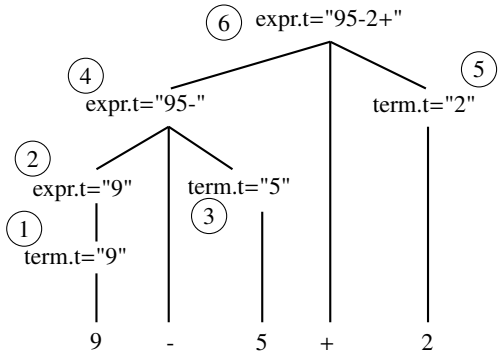
1. Zbiór atrybutów dla każdego symbolu gramatycznego.
2. Zbiór reguł semantycznych do obliczania wartości atrybutów związanych z symbolami występującymi w produkcji.

Atrybuty mogą być obliczone w następujący sposób: Dla danego ciągu wejściowego x tworzymy drzewo parsowania. Następnie stosujemy reguły semantyczne w każdym węźle drzewa parsowania w następujący sposób: Załóżmy, że węzeł N w drzewie parsowania jest oznaczony symbolem X. Wtedy zapisujemy wartość atrybutu a w tym węźle jako $X.a$. Drzewo parsowania, pokazujące wartości atrybutów w każdym węźle, nazywamy drzewem parsowania z przypisami. Wartość atrybutu syntezowanego dla węzła N w drzewie parsowania oblicza się na podstawie atrybutów jego dzieci i atrybutu własnego.

Przykład obliczania atrybutów: Poniżej podano definicję przekładu sterowanego składnią do tłumaczenia postaci infiksowej na postfiksową:

```
1  expr -> expr1 + term     expr.t := expr1.t || term.t ||"+"
2  expr -> expr1- term      expr.t := expr1.t ||term.t || "-"
3  expr -> term             expr.t := term.t
4  term -> 0                term.t := "0"
5  term -> 1                term.t := "1"
6  term -> 2                term.t := "2"
7  term -> 3                term.t := "3"
8  ...                      ...
9  ...                      ...
10 term -> 9                term.t := "9"
```

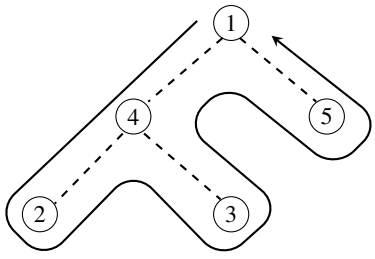
Wartości atrybutów w węzłach drzewa parsowania dla ciągu: $9 - 5 + 2$ podano na rysunku 2.1. Liczby w okręgach oznaczają kolejność dopasowania poszczególnych atrybutów:



Rys. 2.1. Atrybuty syntezowane

Przechodzenie przez drzewo jest stosowane do obliczania atrybutów oraz wykonywania fragmentów kodu (akcji) w schemacie translacji. Przechodzenie przez drzewo zaczyna się od jego korzenia, następnie odwiedza się każdy węzeł drzewa w pewnej kolejności.

Przechodzenie drzewa w głąb (ang. *depth-first traversal*) zaczyna się od korzenia; polega na odwiedzaniu dzieci każdego węzła w dowolnej kolejności, niekoniecznie od lewej do prawej. Ogólny schemat tego przechodzenia zaprezentowano na rysunku 2.2. Nazywa się ono „w głąb", ponieważ odwiedza nieodwiedzone dziecko węzła, gdy tylko może, a więc odwiedza węzły występujące jak najdalej od korzenia i tak szybko, jak jest to możliwe. Na rysunku 2.3 przedstawiono przykład przechodzenia drzewa w głąb; liczby w okręgach określają kolejność przechodzenia.

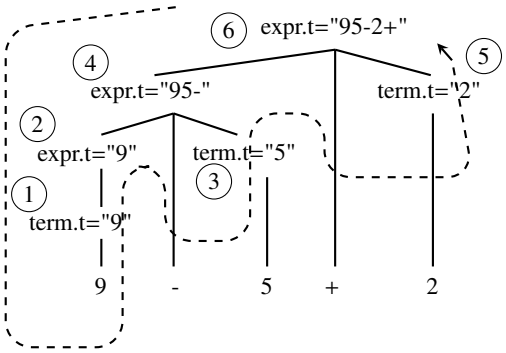


Rys. 2.2. Przechodzenie drzewa w głąb – schemat

Definicja sterowana składnią nie narzuca żadnej konkretnej kolejności obliczania atrybutów w drzewie parsowania; każda kolejność, która oblicza atrybut *a* po wszystkich innych atrybutach, od których *a* zależy, jest akceptowalna.

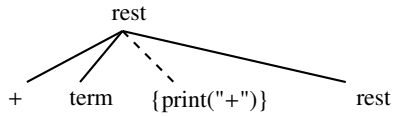
Przechodzenie drzewa w głąb:

```
procedure visit(node N)
 {
   for (each child C of N, from left to right)
         visit (C) ;
     zastosuj reguły semantyczne w węzle N;
 }
```



Rys. 2.3. Przechodzenie drzewa w głąb – przykład

Schemat translacji sterowany składnią jest to notacja dla określenia translacji, która powstaje po dołączeniu fragmentów kodu do produkcji w gramatyce. Fragmenty kodu, dodane do produkcji, nazywamy akcjami semantycznymi. Pozycja, gdzie akcja ma zostać wykonana, jest pokazana przez umieszczenie jej w klamrach po prawej stronie produkcji: $rest \rightarrow + \; term \; print(" + ") rest$. Dodatkowy węzeł reprezentuje akcję semantyczną; jest połączony na rysunku 2.4 linią przerywaną z węzłem odpowiadającym „głowie” (lewej stronie).



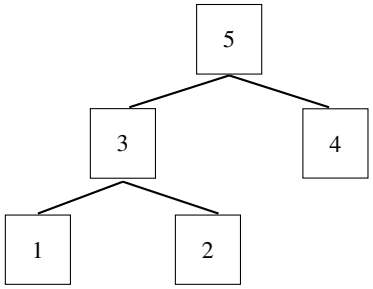
Rys. 2.4. Dodatkowy węzeł w schemacie translacji

Akcje do translacji na notację postfiksową, akcje semantyczne po prawej stronie, w nawiasach klamrowych:

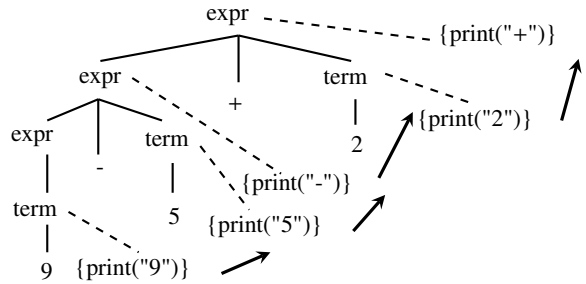
```
expr -> expr1 + term     {print('+')}
expr -> expr1- term      {print('-')}
expr -> term
term -> 0                {print ('0')}
term -> 1                {print ('1')}
...
...
term -> 9                {print ('9')}
```

Implementacja schematu translacji musi zapewnić, że akcje semantyczne zostaną wykonane w kolejności, w jakiej pojawiają się w trakcie przechodzenia drzewa *post-order*. Zanim odwiedzimy dany wierzchołek, odwiedzamy wszystkich jego potomków. Poruszamy się od najniższej generacji w górę. Liczby w wierzchołkach na rysunku 2.5, pokazującym przechodzenie drzewa w głąb, oznaczają kolejność ich odwiedzania.

Implementacja nie musi w rzeczywistości konstruować drzewa parsowania, jeżeli zapewnia, że wszystkie akcje semantyczne są wykonywane tak, jak byśmy konstruowali drzewo syntaktyczne; następnie akcje przedstawione na rysunku 2.6 są wykonywane zgodnie z przechodzeniem *post-order*. Rysunek ten pokazuje translację wyrażenia na jego notację postfiksową.



Rys. 2.5. Przechodzenie drzewa w głąb – przykład



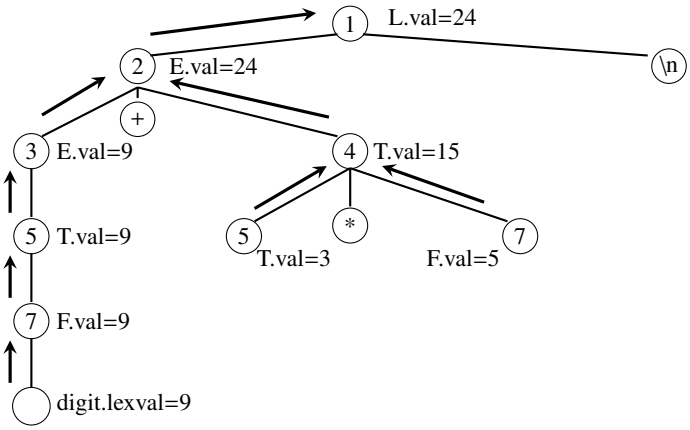
Rys. 2.6. Translacja wyrażenia 9 – 5 + 2 do postaci 9 5 – 2+

Poniższe produkcje i akcje semantyczne definiują prosty kalkulator:

```
1 L ::= E \n       L.val = E.val
2 E ::= E1 + T     E.val = E1.val + T.val
3 E ::= T          E.val = T.val
4 T ::= T1 * F     T.val = T1.val * F.val
5 T ::= F          T.val = F.val
6 F ::= ( E )      F.val = E.val
7 F ::= digit      F.val = digit.lexval
```

W powyższym kodzie produkcje występują po lewej stronie, a akcje semantyczne – po prawej. Na rysunku 2.7 jest przedstawiony sposób obliczenia wyrażenia $9+3*5\backslash n$. Liczby w okręgach oznaczają numer produkcji (numer linii) z powyższego kodu.

Uwaga: Produkcja numer 6 (F ::= (E)) nie została wykorzystana do tworzenia drzewa parsowania.



Rys. 2.7. Obliczanie wyrażenia 9 + 3 ∗ 5\n