WI

Metody kompilacji

Wykład 6, 7 - Analiza składniowa

Włodzimierz Bielecki, Piotr Błaszyński

Wydział Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego

30 marca 2020

Parsowanie

Parsowanie. Jest to proces określenia, w jaki sposób ciąg terminali może być generowany przez gramatykę. Dla każdej gramatyki bezkontekstowej istnieje parser, dla którego czas parsowania ciągu z n terminali zajmuje co najwyżej $O(n^3)$ czasu. Ale czas ten jest zazwyczaj nie do przyjęcia w praktyce.

Parsowanie

Na szczęście, dla rzeczywistych języków programowania możemy zaprojektować gramatykę, którą można szybko parsować. Dla większości języków programowania zostały opracowane algorytmy parsowania o złożoności liniowej. Analizatory składni języka programowania prawie zawsze przeglądają ciąg wejściowy od lewej do prawej strony, biorąc pod uwagę tylko jeden terminal w danym momencie. Większość metod parsowania należy do jednej z dwóch klas – metod zstępujących (*top-down*) i metod wstępujących (*bottom-up*). Terminy te odnoszą się do kolejności, w jakiej są budowane węzły w drzewie parsowania.

Parsowanie

W parserach zstępujących (*top-down*), tworzenie drzewa rozpoczyna się od korzenia i polega na przechodzeniu do liści. Natomiast w parserach wstępujących (*bottom-up*) budowa rozpoczyna się od liści i przebiega w kierunku korzenia. Popularność parserów zstępujących jest spowodowana tym, że z łatwością mogą być zbudowane wydajne parsery przy małym nakładzie pracy programisty. Parser wstępujący może obsłużyć jednak więcej bardziej złożonych gramatyk i schematów translacji, co oznacza, że jego zakres stosowalności jest szerszy.

Parsowanie

W parsowaniu zstępującym budowa drzewa syntaktycznego dokonuje się od korzenia, oznakowanego przez nieterminal startowy; następnie wielokrotnie są wykonywane następujące dwa kroki:

1. W węźle N, oznaczonym nieterminalem A, wybiera się jedną z produkcji, której lewa strona jest A, i tworzy się dzieci dla N oznaczone symbolami prawej strony tej produkcji.
2. Szuka się następnego węzła, w którym poddrzewo ma być utworzone; zazwyczaj jest to skrajny lewy nieterminal drzewa bieżącego.

Parsowanie

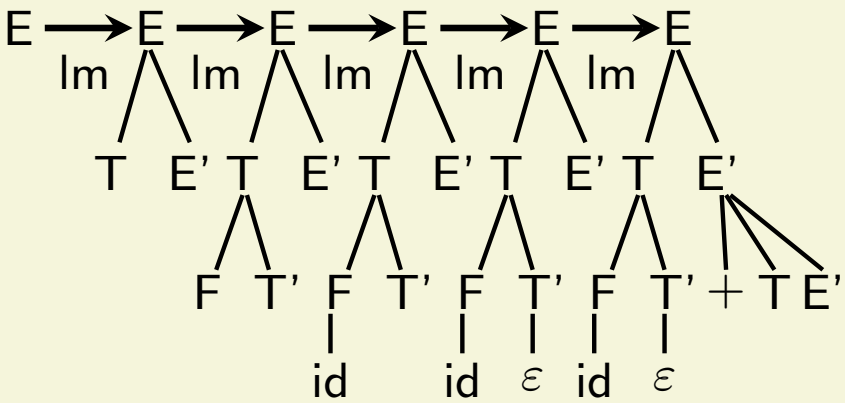
Aktualny symbol terminalny z ciągu wejściowego jest często nazywany symbolem bieżącym *lookahead*. Początkowo symbolem bieżącym jest pierwszy terminal z lewej strony ciągu wejściowego. Parser próbuje utworzyć drzewo parsujące od korzenia w kierunku liści, skanując wejście od lewej do prawej strony.

Parsowanie

Przykład: Dla wejścia $id + id * id$ jest tworzone drzewo parsowania takie jak na rysunku. Na tym rysunku *lm* oznacza wyprowadzenie lewostronne, czyli zawsze jest wybierany skrajny lewy nieterminal.
Użyte produkcje:

```
E -> TE'
E' -> +TE' | ε
T -> FT'
T' -> *FT' | ε
F -> (E) | id
```

Parsowanie



Rysunek: Parsowanie zstępujące

Parsowanie

Parsowanie zstępujące – tworzenie kodu parsera obejmuje kroki:

1. Tworzymy jedną procedurę dla wszystkich symboli terminalnych.
2. Tworzymy jedną procedurę dla każdego symbolu pomocniczego.
3. Tworzymy kod dla każdej produkcji jako ciąg procedur, zgodnie z kolejnością symboli po prawej stronie odpowiedniej produkcji.

Parsowanie

Na rysunku przedstawione zostało drzewo parsowania dla produkcji:

```
stmt->for ( optexpr; optexpr ; optexpr ) stmt
```

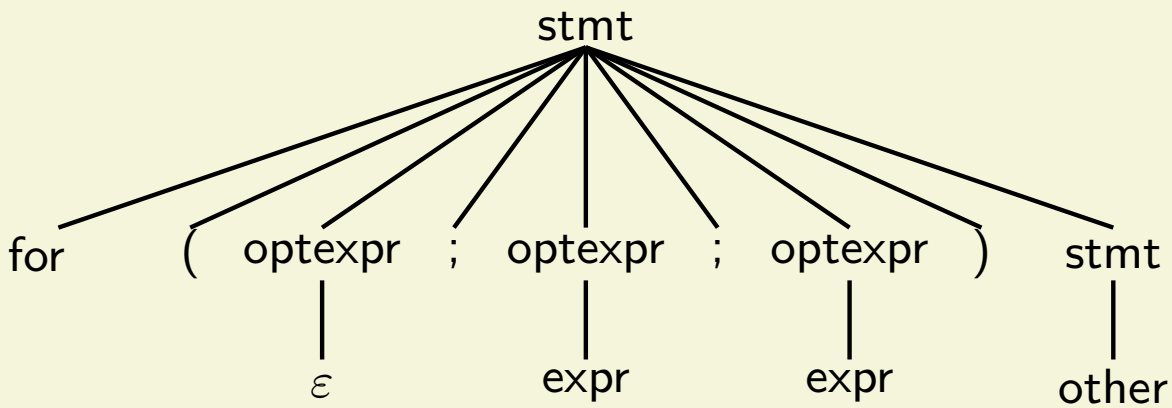
dla kodu o przykładowej strukturze:

```
for ( ; i<n ; i+=2*k)
        p=i ;
```

Parsowanie

Reguły parsowania:

```
stmt -> expr ;
      | if ( expr ) stmt
      | for ( optexpr ; optexpr ; optexpr ) stmt
      | other
optexpr -> ε
      | expr
```



Rysunek: Drzewo parsowania dla produkcji opisujących pętlę *for*

Parsowanie

W poniższym kodzie funkcja *stmt* odpowiada za parsowanie elementu *stmt* z wcześniej zaprezentowanych reguł parsowania. Funkcja *optexpr* odpowiada za zastosowanie ε-produkcji, a funkcja *match* odpowiada za sprawdzenie dopasowania terminala t do symbolu wejściowego.

Parsowanie

Pseudokod parsera, który sprawdza czy ciąg wejściowy zawiera błędy syntaktyczne, jest przedstawiony poniżej:

```
void stmt() {
  switch ( lookahead ) {
        case expr:      match(expr); match(';');
            break;
        case if:
                match(if); match('(');
                match(expr); match(')');
                stmt(); break;
        case for: match(for); match('(');
                optexpr(); match(';');
                optexpr(); match(';');
                optexpr(); match(')');
                stmt(); break;
        case other: match(other); break;
        default: report("syntax error");
  }
}
```

Parsowanie

Funkcje wykorzystywane w kodzie parsera są przedstawione poniżej:

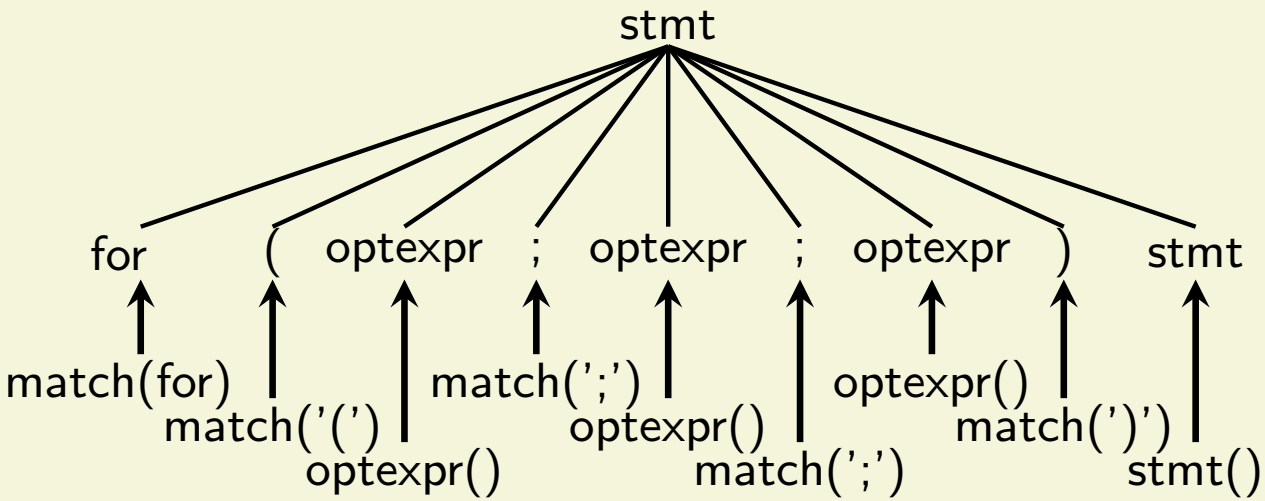
```
void optexpr() {
        if ( lookahead == expr )
                match(expr);
}

void match(terminal t) {
        if ( lookahead == t )
                lookahead = nextTerminal;
        else
                report("syntax␣error");
}
```

Parsowanie

Na pierwszym rysunku przedstawione zostało drzewo parsowania wraz z wywołaniami poszczególnych funkcji dla pętli *for*. Natomiast kolejny rysunek zawiera schemat parsowania zstępującego dla deklaracji tablicy. W momencie wystąpienia pierwszego średnika w konstukcji pętli pojawia się problem braku dopasowania. Wybierana jest wtedy produkcja: *optexpr* $\longrightarrow \varepsilon$. Wywołanie funkcji *stmt* dla *lookahead other* kończy się zakończeniem programu, bez sygnalizacji błędu syntaktycznego, co oznacza, że wejście jest poprawne.

Parsowanie



Rysunek: Drzewo parsowania dla produkcji opisujących pętlę *for*

Parsowanie

Kroki parsowania: ϵ -produkcje (produkcje, których prawa strona jest napisem pustym ϵ) wymagają specjalnego traktowania. Używamy te produkcje jako domyślne, gdy żadna inna produkcja nie może być użyta. Z nieterminalem *optexpr* i symbolem bieżącym: ; ϵ -produkcja jest wykorzystywana, ponieważ nie ma takiej produkcji, której lewa strona jest *optexpr*, a prawą stroną jest symbol: ;. Ogólnie rzecz biorąc, wybór produkcji dla nieterminala może być oparty na metodzie prób i błędów; to znaczy, że możemy spróbować jakiejś produkcji; jeśli jest ona niewłaściwa, to możemy wrócić i spróbować innej produkcji, itd.

Parsowanie

Dla gramatyki jak poniżej należy utworzyć odpowiednią liczbę (3) procedur:

```
type -> simple
      | ^ id
      | array [ simple ] of type
simple -> integer
      | char
      | num dotdot num
```

Parsowanie

Poniższy program implementuje parsowanie zstępujące:

```
procedure match(t : token);
begin
        if lookahead = t then
                lookahead := nexttoken()
        else
                error()
end;
```

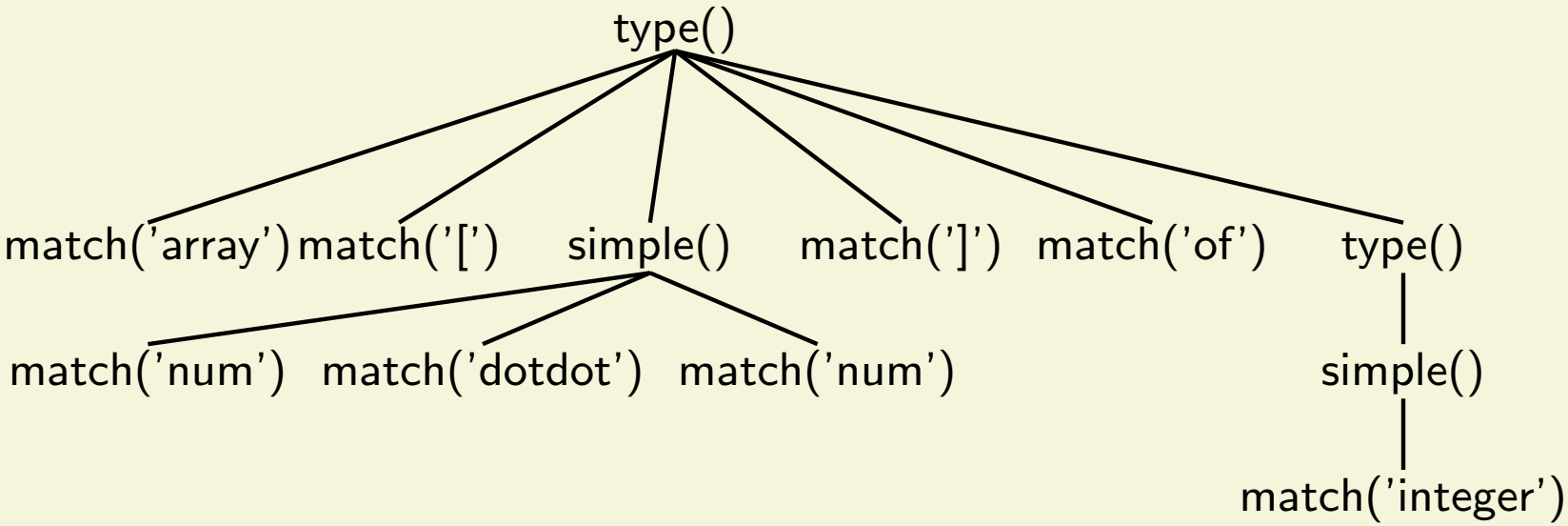
Parsowanie


```
procedure type();
begin
        if lookahead in { 'integer', 'char', 'num' }
             then
                 simple()
        else if lookahead = '^' then
                 match('^'); match(id)
        else if lookahead = 'array' then
                 match('array'); match('['); simple()
                     ;
                 match(']'); match('of'); type()
        else
                 error()
end;
```

Parsowanie


```
procedure simple();
begin
        if lookahead = 'integer' then
                match('integer')
        else if lookahead = 'char' then
                match('char')
        else if lookahead = 'num' then
                match('num');
                match('dotdot');
                match('num')
        else
                error()
end;
```

Parsowanie



Input: array [num dotdot num] of integer

Rysunek: Parsowanie zstępujące dla deklaracji tablicy

Parsowanie

Dla parsera zstępującego możliwe jest zapętlenie parsowania. Problem pojawia się, gdy korzystamy z produkcji lewostronnie rekurencyjnych, takich jak: $expr \longrightarrow expr + term$, gdzie lewy skrajny symbol ciała produkcji jest taki sam jak symbol po lewej stronie produkcji. Produkcja lewostronnie rekurencyjna może być wyeliminowana poprzez jej modyfikację.

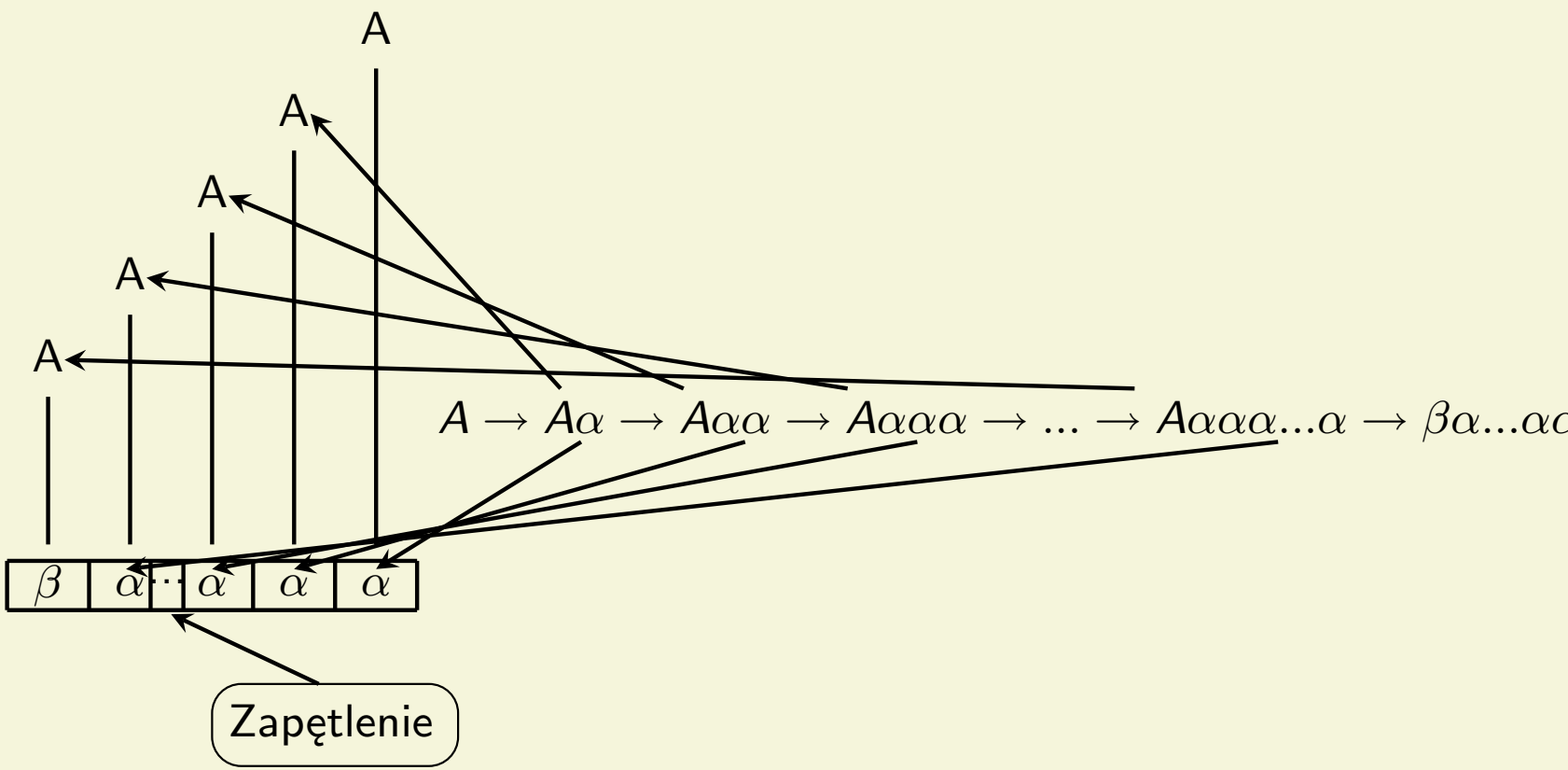
Parsowanie

Rozważmy produkcje: $A- > A\alpha|\beta$,
gdzie α i β są to ciągi terminali i nieterminali, które nie zaczynają się od A.

Parsowanie

Na przykład dla produkcji: $expr -> expr + term | term$ nieterminal $A = expr$, ciąg $\alpha = +term$, ciąg $\beta = term$. Nieterminal A i jego produkcja są lewostronnie rekurencyjne. W ogólnym przypadku gramatyka może być lewostronnie rekurencyjna, jeśli nieterminal A wyprowadza napis $A\alpha$ przez zastosowanie dwóch lub większej liczby produkcji bezpośrednich. Powtarzające się stosowanie tej produkcji tworzy sekwencję ciągu α po prawej stronie A. Jeśli w końcu symbol A zostanie zastąpiony przez β, to uzyskamy β, po którym następuje sekwencja zera lub większej liczby α. Na rysunku parser zawsze będzie wybierał pierwszą produkcję dla gramatyki: $A -> A\alpha | \beta$. Problem: A nigdy nie zostanie zastąpione przez β, ponieważ zawsze będzie wybrana pierwsza produkcja: $A -> A\alpha$.

Parsowanie



Rysunek: Rekurencja lewostronna – ogólna zasada

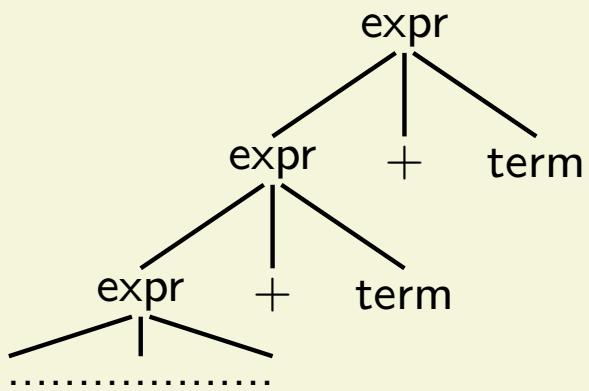
Parsowanie

Rozważmy następujący przykład gramatyki:

```
expr -> expr + term
      | term
term -> 0,1, ..., 9
```

Dla wejścia: $2 + 2$ proces budowania drzewa parsowania dla wyrażenia jest nieskończony, co zobrazowane zostało na rysunku.

Parsowanie



Rysunek: Rekurencja lewostronna – drzewo parsowania

Parsowanie

Problem można rozwiązać przez przepisanie produkcji dla A: $A -> A\alpha|\beta$ w następujący sposób przy użyciu nowego nieterminala R.

```
A -> βR
R -> αR
   | ε
```

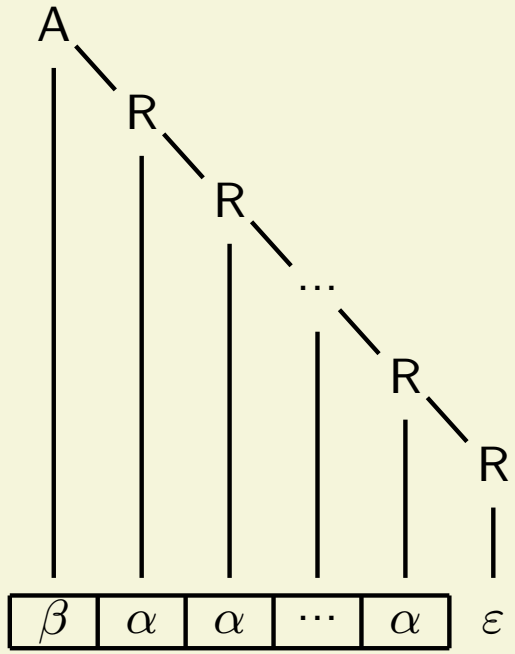
Parsowanie

Rekurencja prawostronna. Nieterminal R i jego produkcja $R->\alpha R$ są prawostronnie rekurencyjne. Produkcje prawostronnie rekurencyjne prowadzą do drzew, które rosną w dół i w prawo, jak to jest pokazane na rysunku.

Parsowanie

$$A \to \beta R$$
$$R \to \alpha R | \varepsilon$$

$$A \to \beta R \to \beta\alpha R \to \beta\alpha\alpha R \to ...$$
$$\beta\alpha\alpha...\alpha R \to \beta\alpha\alpha...\alpha\varepsilon$$



Rysunek: Rekurencja prawostronna – ogólna zasada

Parsowanie

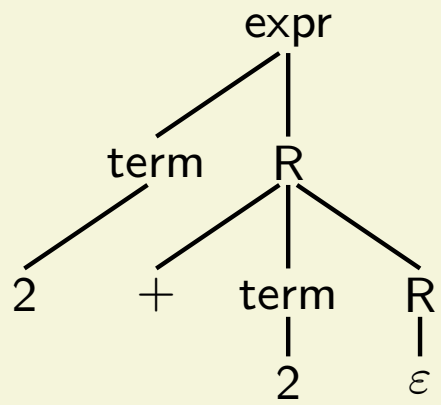
Zapis po przekonwertowaniu do powyższej postaci gramatyki z poprzedniego przykładu:

```
expr -> expr + term | term
term->0,1, ..., 9
```

```
expr -> term  R
R -> +term R
   | ϵ
term->0,1,...,9
```

Parsowanie

Dla wejścia: 2+2 drzewo parsowania wygląda jak na rysunku. Liście drzewa na tym rysunku tworzą zdanie wejściowe, więc parser kończy pracę.



Rysunek: Rekurencja prawostronna – drzewo parsowania