NetCompiler

Second-Rate[1]

Sprawozdanie:

Języki formalne i kompilatory - laboratoria.
Język „programowania” schematów sieci komputerowych.

Prowadzący: dr inż. Bartosz Sawicki
Autor: student Kamil Deryński
Nr indeksu: 197570
Grupa: II INF

Język: Java

Załączniki: - diagram klas.

[1] Second-Rate – wg. Elektronicznych słowników Ectaco znaczy tyle, co kiepski. Podtytuł w pełni obrazuje możliwości i jakość programu.

1. Cel.

Program ma być kompilatorem pseudo języka programowania, który służy do rysowania schematów sieci komputerowych.

Program ma zawierać wszystkie podstawowe elementy kompilatora, a więc analizator leksykalny, analizator składniowy jak i również generator kodu, w tym wypadku schematu. Język programowania schematów jest wymyślony przez autora programu, projekt gramatyki, założenia itd. Projekt języka ma umożliwiać podstawowe cechy języków programowania np. bloki, zmienne itp.

2. Założenia (autora).

Głównym założeniem było poznanie w praktyce zasady działania kompilatora i jego elementów. Zrozumienie problematyki zagadnienia, trudności implementacji danego „typu" gramatyk charakteryzujących się różnymi cechami.

Przedstawienie wyników programu, a więc efektów działania generatora „schematu" była drugorzędną sprawą. Rysowanie schematu sieci jest tylko wizualizacją działania w praktyce „kompilowanego" kodu programu. Z tych założeń wynikają merytoryczne przekłamania w schemacie sieci i uproszczenia w modelu generującym, co jest również poniekąd wynikiem ubogiej gramatyki projektowanego języka;)

3. Przykładowy program w „wymyślonym" języku.

```
.L1(50,10){
     .C1(0,20);
     .C2(20,0);
     switch;
}

.P1(100,10);
.P2(200,100);

main{
     add connect(.L1,.P1);
     add connect(.L1,.P2);
}
```

4. Opis przykładowego programu.

.Lxx	–	blok sieci LAN.
Switch	–	centralna część bloku LAN.
.Cxx	–	komputer.
.Pxx	–	łączę providera.
(x,x)	-	współrzędne obiektów.
main	-	główny blok (połączenia).
(Lxx,Pxx)	-	argumenty funkcji łączącej.

Uwaga1: jako pierwszy argument połączenia musi być zawsze blok LAN.
Uwaga2: w obecnej wersji programu można deklarować tylko jeden blok LAN.

5. Gramatyka.

Symbole terminalne:

$\sum$ = { main , { , } , ; , , , number , (,) , connect , switch , add , block_type , type , λ }

Symbole nieterminalne:

Γ = { S , ELEMENTS , MAIN , ELEMENT , BLOCK , BLOCKS , CONNECTIONS , CONNECT , PARM , ARGUMENT}

Produkcje P:

1. S -> BLOCKS ELEMENTS MAIN
2. BLOCKS -> BLOCK BLOCKS
3. BLOCKS -> λ
4. BLOCK -> block_type PARM {ELEMENTS switch;}
5. ELEMENTS -> ELEMENT ELEMENTS
6. ELEMENTS -> λ
7. ELEMENT -> type PARM ;
8. MAIN -> main { CONECTIONS }
9. CONNECTIONS -> add CONNECT CONNECTIONS
10. CONNECTIONS -> λ
11. CONNECT -> connect (block_type , ARGUMENT) ;
12 ARGUMENT -> type
13. ARGUMENT -> block_type
14. PARM -> (number , number)

Gramatyka (zbiór):

G = { ∑ , Γ , S , P }

Wartości lookahead:

Lookahead (1) = { block_type , type, switch, main }

BLOCKS:
Lookahead (2) = { block_type }
Lookahead (3) = { type, switch, main}

Lookahead (4) = { block_type }

ELEMENTS:
Lookahead (5) = { type }
Lookahead (6) = { switch, main }

Lookahead (7) = { type }

Lookahead (8) = { main }

CONNECTIONS:
Lookahead (9) = { add }
Lookahead (10) = { } }

Lookahead (11) = { connect }

ARGUMENT:
Lookahead (12) = { type }
Lookahead (13) = { block_type }

Lookahead (14) = { (}

Dowodzenie gramatyki LL1

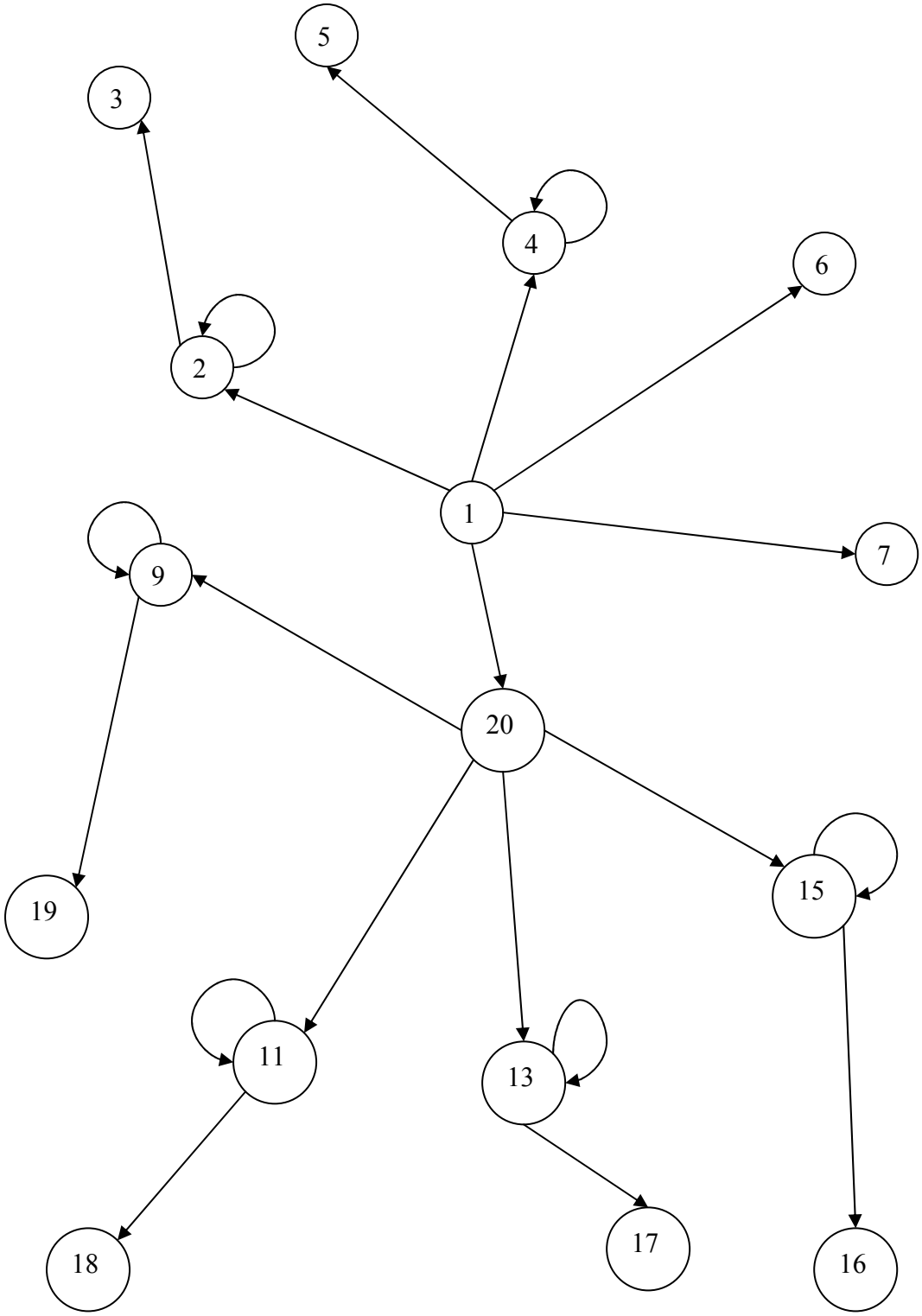
Lookahead (2) ∩ Lookahead (3) = ∅
Lookahead (5) ∩ Lookahead (6) = ∅
Lookahead (9) ∩ Lookahead (10) = ∅
Lookahead (12) ∩ Lookahead (13) = ∅

Wszystkie iloczyny logiczne dają wyniki w postaci zboru pustego, a więc gramatyka jest LL1. Gramatyka zachowuje pełną jednoznaczność.

6. Tablica symboli.

0 - liczba
1 - S
2 - block_type
3 - type
4 - BLOCKS
5 - ELEMENTS
6 - MAIN
7 - BLOCK
8 - ELEMENT
9 - CONNECTIONS
10 - CONNECT
11 - PARM
12 - ARGUMENT
13 - lambda
14 - (
15 -)
16 - {
17 - }
18 - switch
19 - ;
20 - ,
21 - main
22 - add
23 - connect

7. Graf analizatora leksykalnego.



8. Przykłady wychwytywania błędów.

Błąd analizy leksykalnej:

```
.1L1(250,100){

.L,1(250,100){
```

Błąd analizy składniowej:

```
.L1(250,100){
     switch;
     .C1(100,100);
     .C2(100,200);
     .C3(100,300);
     switch;
}


main{
     add connect(.L1,.P1);
     connect add(.L1,.P2);
}

main (brak nawiasu otwierajacego)
     add connect(.L1,.P1);
     add connect(.L1,.P2);
}
```

Błąd odczytu pliku programu:

Brak pliku z programem, jego nieprawidłowa nazwa lub inny błąd IO spowoduje wygenerowanie błędu *„Blad odczytu !”*

9. Wnioski.

Program spełnił podstawowe założenia wymienione w drugim punkcie sprawozdania. Jego funkcjonalność jest bardzo mała jednak budowa analizatorów leksykalnego leksykalnego składniowego jest bardzo elastyczna, a

dzięki rozbudowywaniu grafu można definiować nowe typy elementów sieciowych.

Model graficzny jest bardzo słaby i mało uniwersalny służył jedynie do wizualizacji działania programu.

Poprawić należałoby oczywiście ograniczenie jednego bloku w programie. Jednak jest to kwestia przemyślenia warunków analizatora leksykalnego oraz małych modyfikacji analizatora składniowego. Brak czasu spowodował oddanie wersji rozwojowej programu.