Реферат на тему:

**Побудова алгоритму LA(1)-аналізу**

**1. Правила побудови**

Нехай *G*=(*X*, *N*, *P*, *S*) – LA(1)-граматика без  -правил, можливо, розширена. Опишемо побудову програми синтаксичного аналізу слів мови *L*(*G*). Програма буде містити процедури, іменами яких є відповідні їм нетермінали граматики.

Процедура, відповідна нетерміналу *A*, описує аналіз ланцюжків, вивідних із *A*. Цими ланцюжками є слова мови або їхні підслова. Алгоритм процедури такий. Нехай *A w*1|…|*wk* – усі продукції з нетерміналом *A* ліворуч, *a*1*a*2…*an* – ланцюжок, початок якого треба виводити з *A*. Спочатку визначається, якій із множин first(*w*1), … , first(*wk*) належить символ *a*1. Нехай нею буде first(*w*1), і в найпростішому випадку *w*1=*Y*1*Y*2…*Ym*, де *Yi* – термінал або нетермінал. Початок ланцюжка має виводитися з *Y*1.

Якщо *Y*1 – термінал, то перевіряється рівність *a*1=*Y*1.

Якщо *Y*1 – нетермінал, то з *a*1 починається частина слова, вивідна з *Y*1, і для аналізу початку ланцюжка *a*1*a*2… викликається процедура *Y*1.

В обох випадках, після перевірки рівності або повернення з виклику *Y*1,за деякого *j* 2 початок непроаналізованої частини ланцюжка *ajaj*+1… повинен виводитися з *Y*2 тощо. Перший символ непроаналізованої частини ланцюжка називатимемо ***поточним***.

Отже, за правими частинами *wi* продукцій будуються фрагменти процедури *A*; вони виконуються, коли поточний символ ланцюжка міститься у відповідній множині first ( *wi* ).

Зробимо уточнення програми та правил побудови процедур. Нехай *w* – слово, що аналізується, ch – його поточний символ, функція getc задає добування наступного символу слова, змінна finch позначає спеціальний символ, що повертається функцією getc після закінчення слова *w*. Нехай ok – бульова змінна, що є ознакою належності *w L*(*G*), а процедура error задає присвоювання ok:=**false**. Тілом програми є

**begin**

ok := **true**;

ch := getc;

S; { виклик процедури, відповідної }

{ головному нетерміналу }

writeln ( (ch = finch) **and** ok )

**end**.

Нехай *A* є нетерміналом із продукціями *A w*1|…|*wk* , а *S*1,…, *Sk* позначають множини first(*w*1),…,first(*wk*), які не перетинаються. За таких умов тілом процедури *A* є складений оператор

**begin**

**if** ch **in** S1 **then** *оператор-для-w1* **else**

…

**if** ch **in** S*k* **then** *оператор-для-wk* **else**

error

**end**

Зокрема, якщо *Si* містить лише один символ *x*, то замість умови ch**in***Si* можна записати ch = *x*.

Праві частини розширених граматик є виразами, складеними з послідовностей символів алфавіту *X* і метасимволів, якими є дужки (), [], {} та символи |. Розглянемо праву частину *v* розширеного правила як послідовність виразів *Y*1 … *Yk* , в якій *Yi* є або символом з *X N*, або виразом вигляду (*u*), [*u*], чи {*u*}, що не міститься всередині інших дужок, де *u* – послідовність нетерміналів, терміналів и дужок. За правою частиною *v* будується складений оператор із послідовністю операторів, відповідних до *Y*1,…,*Yk*. Нехай *Y* позначає один із виразів *Y*1,…,*Yk*. Відповідний оператор визначається виглядом *Y* за наступними правилами.

* Якщо *Y* є першим терміналом *Y*1,то оператором є

ch:=getc.

* Якщо *Y* є терміналом, але не першим у ланцюжку *v*,то оператор має вигляд

**if** ch = *Y* **then** ch:=getc **else** error,

тобто треба перевірити збігання поточного символу з *Y* та перейти до наступного символу.

* Якщо *Y* є нетерміналом, то оператором є виклик процедури

Y.

* Якщо *Y* має вигляд (*u*1|…|*um*) і *Ti* позначає first(*ui*) при *i*=1,…,*m*, то треба визначити, до якої з множин *Ti* належить поточний символ, і виконати оператор, відповідний до *ui*:

**if** ch **in** *T*1 **then** *оператор-для-u1* **else**

…

**if** ch **in** *Tm* **then** *оператор-для-um* **else**

error.

* Якщо *Y* має вигляд [*u*], *T*=first(*u*), то за виконання умови ch *T* треба виконати оператор, відповідний до *u*:

**if** ch **in** *T* **then** *оператор-для-u*.

* Якщо *Y* має вигляд {*u*}, *T*=first(*u*), то за виконання умови ch *T* треба повторювати виконання оператора, відповідного до *u*:

**while** ch **in** T **do** *оператор-для-u*.

Зокрема, коли ланцюжок *u* починається терміналом, тобто *u*=*xu*1, де *x X*, то цикл має вигляд

**while** ch = *x* **do**

**begin**

ch := getc;

*оператор-для-u1*

**end**.

Оператори, відповідні до *u*, *u*1, … , *um*, записуються за цими ж правилами.

**2. Побудова аналізатора арифметичних виразів**

Розширена LA(1)-граматика *G*01 із продукціями *E T*{+*T*}, *T F*{\**F*}, *F* (*E*)|*a* породжує мову арифметичних виразів. Згідно з наведеними правилами запишемо процедури E, T, F:

**procedure** E;

**begin**

T;

**while** ch = '+' **do**

**begin** ch := getc; T **end**

**end**;

**procedure** T;

**begin**

F;

**while** ch = '\*' **do**

**begin** ch := getc; F **end**

**end**;

**procedure** F;

**begin**

**if** ch = '(' **then**

**begin**

ch := getc; E;

**if** ch = ')' **then** ch := getc

**else** error

**end**

**else**

**if** ch = 'a' **then** ch := getc

**else** error

**end**

Побудовані процедури ***взаємно рекурсивні***: тіло E містить виклики процедури T, тіло T – виклики F, а тіло F – виклики E. У мові Паскаль кожне ім'я повинно бути означеним вище від його вживання, тому незрозуміло, в якій послідовності треба записати процедури E, T, F. У таких випадках використовують ***конструкцію forward*** .

Якщо в програмі є взаємно рекурсивні підпрограми, то за правилами мови Паскаль спочатку в блоці охоплюючої їх програми (підпрограми) записуються *лише* *заголовки* кількох із них (зокрема, однієї), *а замість їх блоків пишеться слово* ***forward***, тобто, "попереду". Решту підпрограм розміщують так, щоб вони містили виклики підпрограм, чиї заголовки (разом із блоком чи словом **forward**) уже записано вище. Підпрограми, записані спочатку без блоку, розміщаються в кінці зі ***скороченим заголовком*** вигляду

**procedure** <ім'я>; або **function** <ім'я>.

*Список параметрів, дужки й тип функції в скороченому заголовку відсутні*. У даному прикладі процедури E, T, F не мають параметрів, тому скорочені заголовки не відрізняються від повних.

Запишемо програму аналізу арифметичних виразів, вважаючи, що вираз набирається на клавіатурі, а читання його символів задається процедурою getc із модуля Inbuf (розділ 20):

**program** Aexan ( input, output );

**uses** Inbuf;

**var** ch : char;

ok : boolean;

**procedure** error;

**begin** ok := false; ch := finch **end**;

**procedure** E; { тут повний заголовок }

**forward**;

**procedure** F;

… E … { виклик процедури E }

**end**;

**procedure** T;

… F … { виклик процедури F }

**end**;

**procedure** E; { тут скорочений заголовок }

… T … { виклик процедури T }

**end**;

**begin**

ok := **true**; ch := getc;

E; { виклик процедури, відповідної до }

{ головного нетермінала }

writeln ( (ch = finch) **and** ok )

**end**.

Як бачимо, всі виклики посилаються на процедури, чиї імена вже означено раніше.

Уживання взаємно рекурсивних підпрограм інколи називається ***непрямою рекурсією***.