Синтактичен анализ отдолу-нагоре

Синтактичният анализатор отгоре-надолу започва от стартовия символ. Построяването на синтактичното дърво е в посока от корена на дървото към листата (т.е. отгоре-надолу). Правилата се прилагат до достигане на терминални символи в листата.

Синтактичният анализатор отдолу-нагоре започва с разпознавания низ (от терминални символи). Построяването на синтактичното дърво е в посока от листата към корена (т.е. отдолу-нагоре). Правилата се прилагат в обратен ред. Синтактичният анализатор търси в разпознавания низ такъв подниз, който съответства на дясната страна на някое от правилата. Ако намери такъв подниз, анализаторът го замества с нетерминалния символ в лявата част на съответното правило, т.е. разпознаваният низ се редуцира. Целта е разпознаваният низ да бъде редуциран само до стартов символ. Това означава, че входният низ е успешно разпознат.

В общия случай синтактичният анализ отдолу-нагоре е по-мощен в сравнение с анализа отгоре-надолу, но е по-труден за реализация.

Синтактичен анализатор отдолу-нагоре използва *стек* за съхраняване на текущата позиция и *таблица* за определяне на следващата операция.

Алгоритъмът на синтактичния анализ отдолу-нагоре чете лексеми от входния поток и ги въвежда в стека, опитвайки се да построи последователности, които се срещат в дясната част на някое от правилата на граматиката. При откриване на такава последователност тя се замества в стека със съответния нетерминал от лявата страна на правилото. Построяването на синтактичното дърво продължава във възходяща посока, обратна на посоката при анализ отгоре-надолу. При успешно разпознаване би трябвало всички лексеми от входния поток да бъдат въведени в стека и евентуално да се получи последователност, съответстваща на дясната част в правилото на стартовия символ на граматиката.

Пример:

Дадени са правила на граматика:

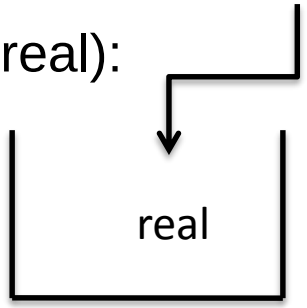
1. <S> → real <IDLIST>
2. <IDLIST> → <IDLIST> , <ID>
3. <IDLIST> → <ID>
4. <ID> → A|B|C|D

Да се получи следното изречение с използване на метод отдолу-нагоре:

real A, B, C

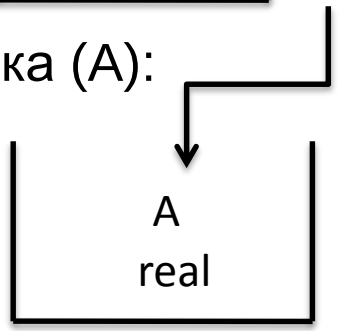
Стъпка 1. Първата лексема се записва в стека (real):

real A, B, C

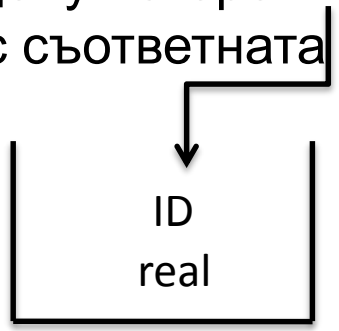


Стъпка 2. Следващата лексема се записва в стека (A):

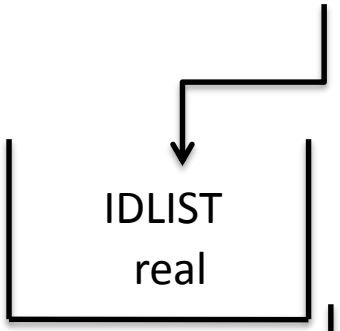
real A, B, C



Стъпка 3. Замества се A според правило 4. Тази операция се нарича *редуциране на стека*. При анализ отдолу-нагоре заместваме дясната страна на правилата със съответната лява страна.



Стъпка 4. Заместваме в стека според правило 3

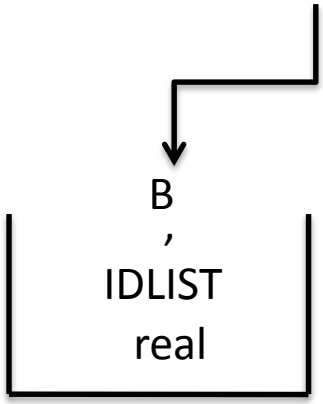


Стъпка 5. Записваме следващата лексема (,)

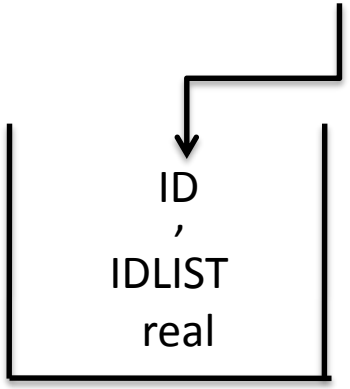
real A , B, C

Стъпка 6. Записваме следващата лексема (B)

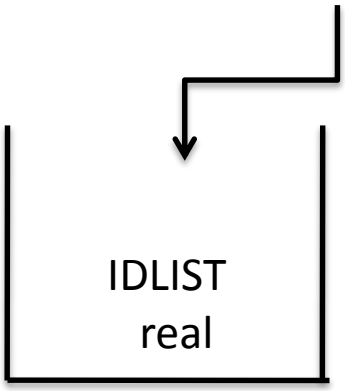
real A , B , C



Стъпка 7. Редуцираме стека според правило 4.



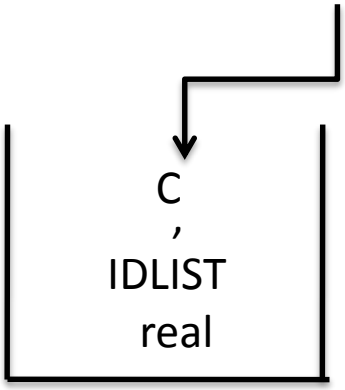
Стъпка 8. Редуцираме стека според правило 2.



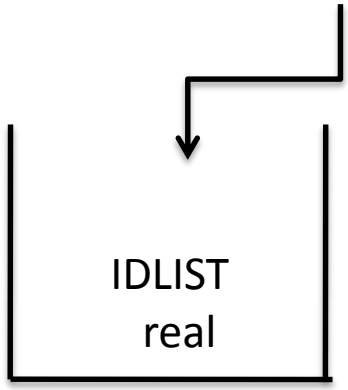
Стъпка 9. Записваме следващата лексема (C)

Стъпка 10. Четем следващата лексема

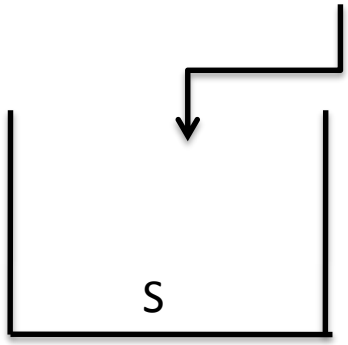
real A , B , C



Стъпка 11. Редуцираме стека според правила 4 и 2



Стъпка 12. Редуцираме стека според правило 1



- Описаният анализатор се нар. *анализатор с изместване и редуциране.* На всяка стъпка такъв анализатор изпълнява една от следните три операции:

Изместване: Ако не е възможно да се изпълни редуциране на стека и има останали лексеми във входния поток, тогава прехвърляме лексема от потока в стека. Тази операция се нар. *изместване*. Например, нека за горната граматика разгледаме стек със съдържание **(real IDLIST ,)** и входен поток **(B,C)**. Невъзможно е да се изпълни редуциране на стека, защото съдържанието му не съответства на дясната страна на никое от правилата. В този случай записваме първата лексема от потока в стека (в случая в стека се записва B, а C остава във входния поток).

Редуциране: Ако съществува правило *A –> w* и съдържанието на стека е **(qw)** за низ q (q може да е празен), тогава съкращаваме (редуцираме) стека до **(qA)**. Правилото за нетерминала A се прилага отдясно-наляво. Например, за горната граматика, ако съдържанието на стека е **(real ID)**, можем да използваме правилото IDLIST –> ID, за да редуцираме стека до **(real IDLIST)**.

Ако при редуцирането в стека се съдържа стартовият символ и няма останали лексеми във входния поток, то входният низ се разпознава като валидно изречение. Това е последната стъпка на успешен анализ.

Грешка: Ако символите, записани в стека, не съответстват на дясната страна на никое от правилата, не може да се приложи редуциране на стека. Също, ако записването на следваща лексема в стека води до последователност, която не може да бъде редуцирана до стартовия символ, няма смисъл да се прилага редуциране. Описаните случаи означават, че входният поток не е валидно изречение.

Проблем

Описаните типове граматики създават конфликти, които се нар. *изместване-редуциране* и *редуциране-редуциране.* Първият вид конфликт се среща, когато анализаторът не може да реши дали да измества или да редуцира. Вторият вид конфликти се среща, когато анализаторът има повече от едно правило за редукция.

Пример за конфликт изместване-редуциране възниква при конструкцията *if-then-else*.

Типично правилото за тази конструкция е:

S –> if E then S | if E then S else S

Разглеждаме какво ще се получи при анализатор с изместване и редуциране:

if E then if E then S else S

В даден момент от анализа стекът ще има следното състояние:

if E then if E then S

, при което *else* ще бъде следващата лексема.

Възможно е да се извърши редуциране на разпознавания низ, защото съдържанието на стека съответства на дясната страна на първото правило. Възможно е и изместване на *else* за построяване на дясната страна на второто правило. Редуцирането затваря вътрешният оператор *if,* като така свързва *else* с външния *if*. Изместването свързва вътрешния *if* с *else*.

Проблемът в примера е как анализаторът да избере операция – изместване или редуциране.

И двете операции са валидни спрямо граматиката, но получените като резултат синтактични дървета са различни.

Проблемът се свежда до избор на оператор *if*, към който принадлежи *else* – към вътрешен или към външен *if*.

В C и Java оператор *else* се отнася към най-близкия оператор *if*. При други езици като Ada и Modula този проблем се решава чрез затварящ оператор *endif*.

Проблем *редуциране-редуциране* се среща рядко и обикновено е следствие от проблем в дефинирането на граматиката.

За вземане на решение каква операция да бъде приложена се използват *детерминирани методи*.

Например, ако се използва терминиращ символ (;), тогава при срещане на този символ анализаторът трябва да редуцира, в противен случай да измества. Например в горната граматика за съдържание на стека “**real IDLIST“,** ако следващият символ е “**;”,** ще се изпълни редуциране, а ако следващият символ е “**,**”, ще се изпълни изместване.

Имайки обща представа как работи анализатор с изместване и редуциране, ще разгледаме как обработва входен низ и как определя кое правило да приложи при редуциране. За целта ще разгледаме метод за реализация нар. LR анализ. Този метод е приложим за всички детерминирани езици и граматики.

LR анализ

LR анализаторите ("L" означава, че входният низ се преглежда отляво-надясно, "R" означава дясно извеждане на изречение) са ефективни, таблично управляеми анализатори с изместване и редуциране. Класовете граматики, които могат да бъдат породени с LR метод, са по-голямо множество от класовете граматики, които са породени чрез LL. Всички програмни конструкции, които се дефинират с контекстно-свободни граматики, могат да бъдат реализирани чрез LR метод. Допълнително предимство на LR е, че при повечето граматики няма необходимост от преработване на правилата, както е при LL граматиките.

LR анализаторът използва две таблици:

1. *Таблица на операциите Action[s,a],* която указва на анализатора каква операция да изпълни при състояние **s** във върха на стека и при следващ терминален символ **a** в разпознавания низ. Възможните операции са изместване на състояние в стека, редуциране на символ във върха на стека, приемане на низ (разпознаване на изречение) или маркиране на грешка.
2. *Таблица Goto[s,X]* показва кое да е новото състояние във върха на стека след редуциране на нетерминалния символ X при състояние **s** във върха на стека.

Двете таблици се използват комбинирано, като таблицата на операциите съдържа редове за терминални символи, а Goto таблицата съдържа редове за нетерминалните символи. Обединената таблица се нарича *LR парсираща таблица*.

Пример за LR парсираща таблица

Състояние	S	IDLIST	ID	real	,	A B C D	;
1	STOP HALT			S2			
2		S5	S4			S3	
3					R4		R4
4					R3		R3
5					S6		R1
6			S4			S3	
7					R2		R2

Редовете в LR таблицата съответстват на състоянията, в които може да се намира анализаторът.

Всяко състояние съответства на позиция от граматическо правило, което е достигнато от анализатора.

Анализаторът има 2 стека – стек за символи и стек за състояния.

Таблицата се състои от 4 типа елементи:

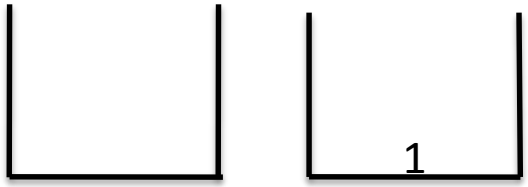
1. Изместващи елементи. Те имат форма Sx. Изместващ елемент има следното значение:
   - Да постави в символния стек символа, съответстващ на текущата колона.
   - Да постави в стека на състоянията състоянието **x** и да направи преход в състояние **x**.
   - Ако следващият входен символ е терминален, да го прочете

2. Изместващи елементи с форма Ry. Такъв елемент има следното значение:
   - Редуцира с правило **y.**
   - Ако дясната страна на правилото **y** има **n** символа, тогава извлича (pop) **n** символа от символния стек и от стека на състоянията.
   - Преход към състоянието във върха на стека на състоянията.
   - Нетерминалният символ в лявата страна на правило **y** се приема за следващия входен символ.
3. Елементи за грешка. Всяка празна позиция в таблицата съответства на синтактична грешка.
4. Елемент за край (един или повече). Тези елементи приключват анализа.

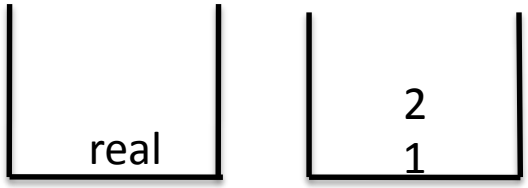
Пример

Пораждане на изречение
real A,B,C;

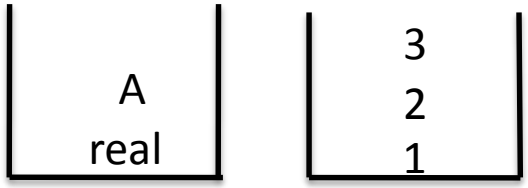
Начално състояние S1
real A,B,C;



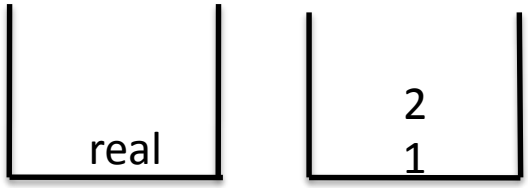
Входен символ real – S2: изместване в 2,
входен символ става A
real A,B,C



Входен символ A – S3:
real A,B,C

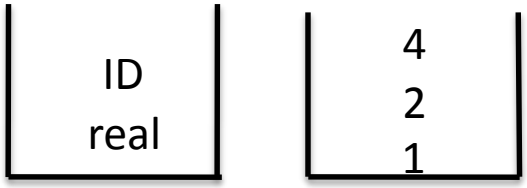


Входен символ , - R4: замества правило 4

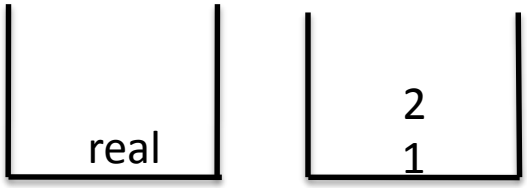


Пример

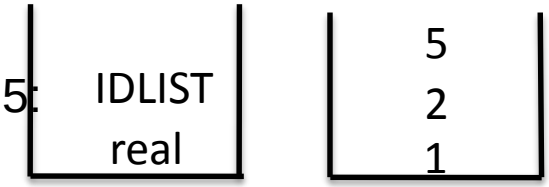
Входен символ ID, преход в състояние 4
real A,B,C;



Входен символ , – R3: замества правило 3,
real A,B,C



Входен символ IDLIST – S5: измества в съст. 5:
real A,B,C



Входен символ , - S6: изместване в състояние 6
real A,B,C