Traducerea dirijată de sintaxă

- Problematica din acest capitol aparține deja de analiza semantică.
- O modalitate concretă pentru rezolvarea problemelor de semantică în cazul în care procesul de traducere este coordonat de o GIC, constă în atașarea de **atribute semantice** la simbolurile gramaticale. Valorile acestor atribute sunt calculate prin **reguli semantice** asociate cu producțiile gramaticale. Această asociere se poate realiza prin:
  - definiții dirijate de sintaxă (DDS);
  - scheme de traducere (ST).
- Privite ca notații, DDS sunt mai evoluate decât ST. Ele nu tratează detalii de implementare și nu explicitează ordinea în care are loc traducerea. ST în schimb, indică ordinea în care sunt evaluate regulile semantice și permit să se arate unele detalii de implementare. Ambele notații se pot utiliza atât pentru specificarea verificărilor semantice cât și pentru generarea codului intermediar.
- În principiu, atât pentru DDS cât și pentru ST se parcurg etapele din fig. de mai jos, chiar dacă această etapizare nu este întotdeauna explicită.

şir de intrare → arbore sintactic → graf de dependență → ordinea de evaluare pentru regulile semantice

- Evaluarea regulilor semantice (RS) poate avea ca rezultat următoarele activități:
  - generarea de cod intermediar;
  - salvarea informațiilor calculate într-o TS;
  - emiterea unor mesaje de eroare;
  - executarea altor activități.
- DDS poate fi implementată într-o singură trecere, prin evaluarea RS în timpul AS, fără a construi explicit un arbore sintactic sau un graf arătând dependența între atribute. Deoarece implementarea într-o singură trecere conduce la o compilare eficientă, capitolul de față va fi dedicat în special acestor cazuri.
- Traducerile care pot fi executate fără construirea explicită a unui arbore sintactic sunt practic acoperite de o subclasă importantă a DDS numite definiții “L–atributate”.

Definiții dirijate de sintaxă

- O DDS este o generalizare a unui GIC în care fiecărui simbol gramatical i se asociază o mulțime de atribute. Există o mare varietate de reprezentări pentru atribute: șiruri, numere, tipuri, locații de memorie etc. Valoarea unui atribut la un nod din arborele sintactic se calculează printr-o RS asociată cu producția utilizată în acel nod. După felul în care se calculează valoarea, atributul poate să fie:
  - **sintetizat**: valoarea este calculată în funcție de atributele fiilor nodului curent;
  - **moștenit**: valoarea se calculează din valorile atributelor fraților și părintelui nodului curent.
- Între atribute se stabilesc anumite legături reprezentate printr-un graf de dependență (GD). Din GD rezultă ordinea de evaluare pentru RS. Evaluarea RS definește valorile atributelor la nodurile din arborele sintactic, pentru șirul de intrare dat. O RS poate, de asemenea, să aibă efecte colaterale: tipărirea unei valori, actualizarea unei variabile globale etc.
- Un arbore sintactic la care se indică, la fiecare nod, valorile atributelor, se numește **arbore sintactic adnotat** iar procesul de calcul aferent se numește **adnotarea arborelui sintactic**.

Forma unei DDS

Într-o DDS, fiecărei producții gramaticale **A→α** i se asociază o mulțime de reguli semantice de forma b:=f(c_1, c_2, ..., c_k) unde **f** este o funcție. În ceea ce privește **b**, se disting două cazuri:

1) **b** este un atribut sintetizat al lui **A** iar c_1, c_2, ..., c_k sunt atribute aparținând simbolurilor gramaticale din partea dreaptă a producției.

2) **b** este un atribut moștenit al unuia din simbolurilor gramaticale din partea dreaptă a producției iar c_1, c_2, ..., c_k sunt atribute aparținând lui A sau oricăruia din simbolurile gramaticale din partea dreaptă a producției.

Se spune că atributul **b** depinde de atributele c_1, c_2, ..., c_k. O **gramatică atributată** (GA) este o DDS în care funcțiile din RS nu pot avea efecte secundare. Deseori funcțiile din RS sunt exprimate ca expresii. În alte situații RS pot fi scrise ca apeluri de proceduri sau fragmente de program şi nu se pun sub formă de atribuire.

Exemplu. În figură se prezintă o DDS pentru un program gen "calculator de birou". Această definiție asociază fiecărui neterminal E, T sau F câte un atribut sintetizat cu valoare întreagă numit **val**. Pentru fiecare producție E, T sau F, prin RS, s-a calculat valoarea atributului **val** corespunzătoare neterminalului din partea stângă în funcție de valorile lui **val** ale neterminalului din partea dreaptă.

Producția	Reguli semantice
L→E**nl**	tipărește (E.val)
E→E_1+T	E.val:=E_1.val+T.val
E→T	E.val:=T.val
T→T_1∗F	T.val:=T_1.val∗ F.val
T→F	T.val:=F.val
F→(E)	F.val:=E.val
F→**cifră**	F.val:=**cifră**.lexval

DDS pentru un calculator de birou simplu

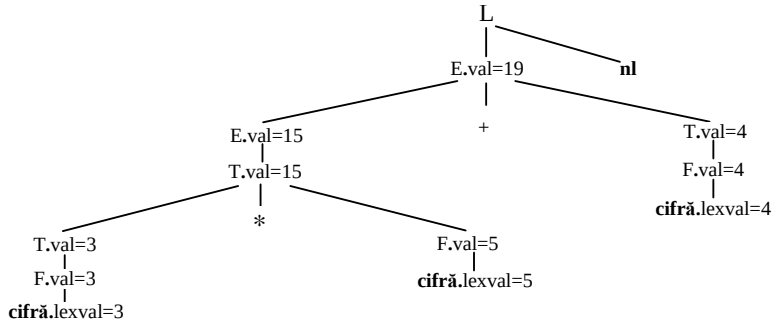
Atomul cifră are un atribut sintetizat **lexval** a cărui valoare este furnizată de analizorul lexical. RS asociată cu producția **L**→E**nl**, pentru neterminalul de start, este o procedură care tipărește valoarea expresiei aritmetice generată de **E**; se poate imagina că această regulă definește un atribut fictiv pentru neterminalul **L**.

Într-o **DDS** se presupune că terminalele au numai atribute sintetizate ale căror valori sunt furnizate de analizorul lexical şi nu de **RS** asociate definiției sintactice. În plus, de obicei, simbolul de start nu are atribute moștenite.

Atribute sintetizate

O DDS care utilizează în exclusivitate atribute sintetizate se numește definiție **S–atributată**. Un arbore sintactic pentru o definiție S–atributată poate fi întotdeauna adnotat prin evaluarea RS pentru atribute, parcurgând nodurile de jos în sus, de la frunze spre rădăcină.

Exemplu. Definiția S–atributată din exemplul anterior specifică un calculator de birou care citește o linie de intrare constând dintr-o expresie aritmetică ce conține cifre, paranteze, operatorii + şi ∗, urmată de caracterul special **nl** şi tipărește valoarea expresiei. De exemplu, dacă se dă expresia 3∗5+4**nl**, programul tipărește valoarea 19. În figură se prezintă arborele sintactic adnotat pentru această intrare. Ieşirea tipărită la rădăcina arborelui este valoarea lui E.val la primul fiu al rădăcinii.



Arborele sintactic adnotat pentru 3∗5+4**nl**

Pentru a urmări cum se calculează valorile atributelor, considerăm nodul interior din stânga jos, care corespunde utilizării producției F→**cifră**. **RS** aferentă "F.val:=**cifră**. lexval" definește atributul *F.val* la acel nod ca având valoarea 3 (valoarea lui cifră **.** lexval la fiul acestui nod).

Să considerăm acum nodul corespunzător producției **T→T_1∗F**. Valoarea atributului T.val rezultă din **RS**:

T.val:=T_1.val∗ F.val

3 (din fiul stâng) 5 (din fiul drept)

⇒ T.val=15

Atribute moştenite

În cazul în care valoarea unui atribut este definită în funcție de atributele de la părintele şi/sau fraţii acelui nod, atributul este moştenit. Atributele moştenite sunt convenabile pentru exprimarea dependenței unei construcții de limbaj de programare de contextul în care apare aceasta. De exemplu, se poate utiliza un atribut moştenit pentru a ține evidența dacă un identificator apare în stânga sau în dreapta unei asignări, spre a se putea decide dacă este necesară adresa sau valoarea identificatorului. Deşi este întotdeauna posibil să se rescrie o DDS utilizând numai atribute sintetizate, utilizarea, în anumite situații, şi a atributelor moştenite conduce la o definiție mai naturală.

În exemplul următor, un atribut moştenit furnizează tipul diferiților identificatori dintr-o declarație.

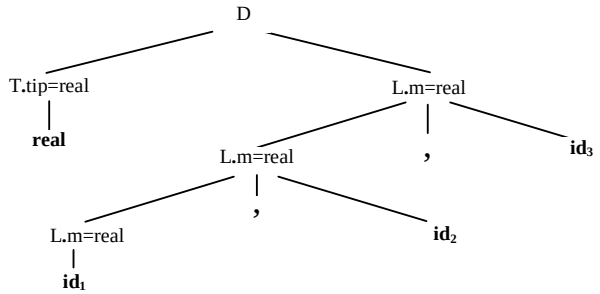
Exemplu. O declarație generată de neterminalul D din DDS prezentată în figură constă din cuvântul cheie **int** sau **real** urmat de o listă de identificatori.

Producția	Regulile semantice
D→TL	L.m:=T.tip
T→**int**	T.tip:=integer
T→**real**	T.tip:=real
L→L_1,**id**	L_1.m:=L.m adaugă tip (id.intrare, L.m)
L→**id**	adaugă tip (id.intrare, L.m)

DDS cu atributul moştenit *L.m*

Neterminalul T are un atribut sintetizat **tip** a cărui valoare este determinată de cuvântul cheie din declarație. RS "L.m:=T.tip", asociată cu producția D→TL, stabileşte atributul moştenit *L.m* la tipul de declarație.

În figura următoare este prezentat un arbore sintactic adnotat, pentru secvența de intrare: **real** id_1, id_2, id_3.



Arbore sintactic cu atributul moştenit <u>m</u> în fiecare nod etichetat *L*

Grafuri de dependență

Dacă un atribut **b** la un nod dintr-un arbore sintactic depinde de un atribut **c**, atunci RS pentru **b** la acel nod trebuie evaluată după RS prin care este definit **c**. Interdependențele între atributele moştenite şi sintetizate în nodurile unui arbore sintactic pot fi exprimate printr-un graf orientat numit **graf de dependență** (**GD**).

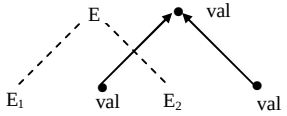
Algoritmul de construire este următorul:

for ∗fiecare nod **n** din arborele sintactic **do**
  for fiecare atribut **a** al simbolului gramatical din **n do**
    ∗ construieşte un nod în GD pentru **a**
for ∗fiecare nod **n** din arborele sintactic **do**
  for fiecare RS b:=f(c_1, c_2, ...,c_k) asociată
    cu producția utilizată la nodul **n do**
    for i:=1 **to** k **do**
      ∗ construieşte un arc de la nodul pentru $\mathbf{c_k}$ la nodul pentru **b**

Exemplu. Să considerăm producția şi RS asociată, date mai jos:

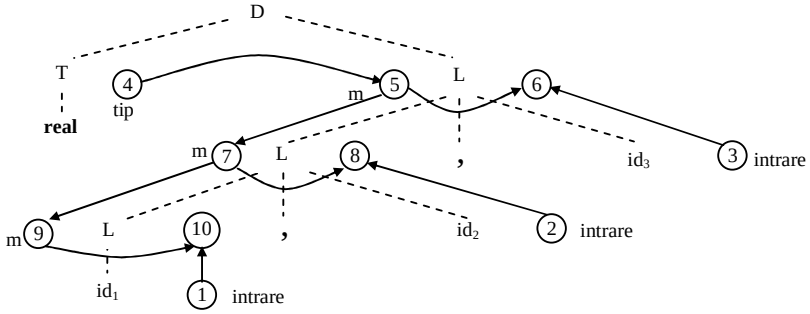
PRODUCȚIA	RS
E→E_1+E_2	E.val:=E_1.val+E_2.val

Ori de câte ori această producție este utilizată într-un arbore sintactic, la GD se vor adăuga arcele prezentate în figură.



Sintetizarea lui E.val din E_1.val şiE_2.val

Exemplu. În figură se prezintă un GD pentru arborele sintactic din figura reprezentând arborele sintactic cu atributul **m** în fiecare nod etichetat L.



GD pentru arborele sintactic din fig. AS cu atributul moştenit <u>m</u> în fiecare nod

Nodurile din GD sunt marcate prin numere.

Ordinea de evaluare

Orice ordonare $m_1, m_2,...,m_k$ a nodurilor unui graf orientat aciclic astfel că arcele merg exclusiv de la noduri precedente în ordonare spre noduri următoare, se numeşte **sortare topologică**. De exemplu, dacă $m_i \rightarrow m_j$ este un arc de la m_i la m_j, atunci m_i nu apar în ordonare înaintea lui m_j.

Orice sortare topologică a unui GD conduce la o ordine corectă în care se pot evalua RS asociate cu nodurile dintr-un arbore sintactic. În acest fel atributele $c_1,c_2,...,c_k$ dintr-o RS $b:=f(c_1,c_2,...,c_k)$ vor fi disponibile înainte ca *f* să fi fost evaluat.

Procesul de translatare specificat de o DDS poate fi descris astfel:

1) Se construieşte arborele sintactic corespunzător şirului de intrare, pe baza gramaticii iniţiale.
2) Din arborele sintactic se construieşte GD prin metoda descrisă mai sus.
3) Se sortează topologic nodurile GD rezultând ordinea de evaluare pentru RS.
4) Prin evaluarea RS în această ordine rezultă translatarea şirului de intrare.

Exemplu. Fiecare arc din GD prezentat anterior merge de la un nod cu număr mic la unul cu număr mai mare. Rezultă că scrierea nodurilor în ordinea numerelor asociate lor conduce la o sortare topologică. Din această sortare rezultă următorul program, în care prin $\mathbf{a_n}$ s-a notat atributul asociat cu nodul numerotat **n** în GD:

a_4:=real;
a_5:=a_4;
adaugă tip (id_3.intrare, a_5);
a_7:=a_5;
adaugă tip (id_2.intrare, a_7);
a_9:=a_7;
adaugă tip (id_1.intrare, a_9);

Evaluarea acestor RS înregistrează tipul **real** în TS pentru fiecare identificator.

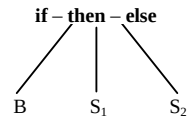
Construirea arborilor sintactici abstracţi

- Se prezintă modul de utilizare a unei DDS pentru specificarea construcţiei arborilor sintactici abstracţi (ASA) şi a altor reprezentări grafice ale construcţiilor de limbaj.
- Utilizarea ASA ca reprezentare intermediară permite ca traducerea să fie despărţită de analiza sintactică. Rutinele de traducere apelate în timpul AS trebuie să respecte două tipuri de restricţii:

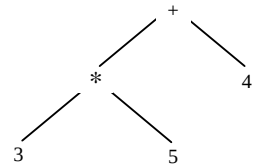
1. gramatică adecvată pentru AS poate să nu reflecte structura ierarhică naturală a construcţiilor de limbaj. De exemplu, o gramatică pentru Fortran poate privi o subrutină ca fiind formată doar dintr-o listă de instrucţiuni. Însă analiza subrutinei se poate simplifica dacă se utilizează o reprezentare de arbore care reflectă suprapunerea (încuibarea) ciclurilor DO.
2. Metoda de AS restrânge ordinea în care sunt considerate nodurile dintr-un arbore sintactic. Această ordine s-ar putea să nu corespundă cu ordinea în care devine disponibilă informaţia despre o anumită construcţie. Din acest motiv, compilatoarele pentru C construiesc arbori sintactici pentru declaraţii.

Arbori sintactici abstracţi

Un ASA este o formă condensată a unui arbore sintactic, utilă pentru reprezentarea construcţiilor de limbaj. Producţia S→**if** B **then** S_1 **else** S_2 poate apărea într-un ASA astfel:



Într-un ASA operatorii şi cuvintele cheie nu apar ca frunze ci sunt asociate cu nodul interior care este părintele acelor frunze în arborele sintactic. O altă simplificare întâlnită într-un ASA este concentrarea lanţurilor de producţii singulare. Arborele sintactic din figura reprezentând arborele sintactic adnotat pentru 3∗5+4**nl** devine ca ASA:



Traducerea dirijată de sintaxă se poate baza ori pe ASA ori pe arbori sintactici obişnuiţi. Abordarea este aceeaşi în fiecare caz: se ataşează atribute nodurilor.

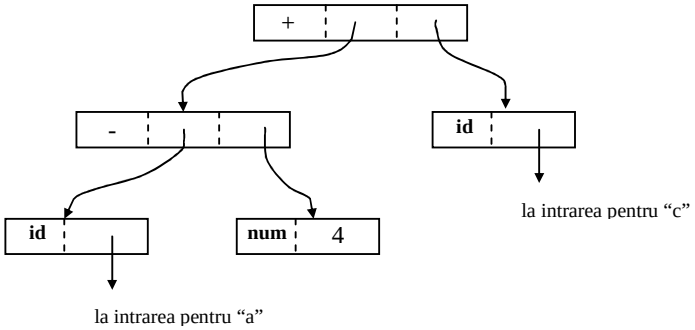
Construirea ASA pentru expresii

Operația de construire a unui ASA pentru o expresie este similară cu traducerea expresiei în formă postfix. Se construiesc subarbori pentru subexpresii creând un nod pentru fiecare operator şi operand. Fiii unui operator sunt rădăcinile subarborilor reprezentând subexpresiile ce constituie operanzii acelui operator.

Crearea nodurilor ASA pentru expresii cu operatori binari se face apelând următoarele funcții (fiecare funcție returnează un pointer la nodul nou creat):

1. **nod** (op, st, dr) creează un nod operator cu eticheta **op** şi două câmpuri conținând pointeri la stânga şi dreapta.
2. **frunză** (**id**, intrare) – creează un nod identificator cu eticheta **id** şi un câmp conținând un pointer, "intrare", spre intrarea în TS pentru identificator.
3. **frunză** (**num**, val) – creează un nod număr cu eticheta **num** şi un câmp conținând valoarea numărului.

Exemplu. Următoarea secvență de apeluri de funcții creează ASA pentru expresia a–4+c din figură.



ASA pentru a–4+c

În această secvență p_1, p_2, ... , p_5 sunt pointeri la noduri iar int-a şi int-c sunt pointeri la intrările din TS pentru identificatorii "a" şi respectiv "c".

(1) p_1:=frunză (**id**, int–a);
(2) p_2:=frunză (**num**, 4);
(3) p_3:=nod ('–',p_1, p_2);
(4) p_4:=frunză (**id**, int–c);
(5) p_5:=nod ('+',p_3, p_4);

Arborele este construit de jos în sus. p_5 indică rădăcina arborelui.

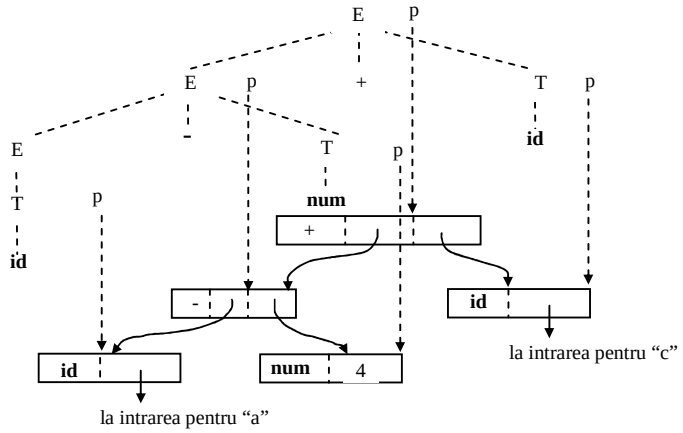
O DDS pentru construirea ASA

Figura următoare conține o definiție S–atributată de construire a unui ASA pentru o expresie ce conține operatorii + şi –. Ea utilizează producțiile gramaticii pentru a planifica apelul funcțiilor "nod" şi "frunză" care construiesc arborele. Atributul sintetizat "p" pentru E şi T ține evidența pointerilor returnați pentru apelurile de funcții.

Producția	Regulile semantice
$E \rightarrow E_1+T$	E.p:=nod ('+', E_1.p, T.p)
$E \rightarrow E_1-T$	E.p:=nod ('–', E_1.p, T.p)
$E \rightarrow T$	E.p:=T.p
$T \rightarrow (E)$	T.p:=E.p
T→**id**	T.p:=frunză (**id**, **id.**intrare)
T→**num**	T.p:=frunză (**num**, **num.**val)

DDS de construire a unui ASA pentru o expresie

Exemplu. În figură este prezentat un nou arbore sintactic adnotat înfățişând construcția unui ASA pentru expresia a–4+c.

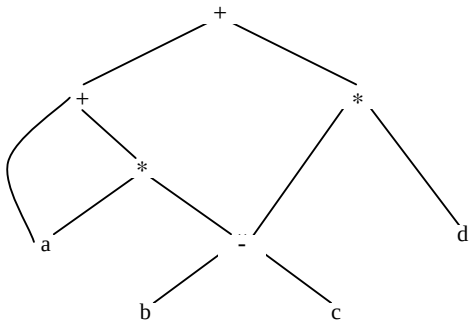


Construirea unui ASA pentru a–4+c

Arborele sintactic este desenat punctat. Nodurile etichetate cu neterminalele E şi T utilizează atributul sintetizat "p" pentru a reține un pointer la nodul din ASA pentru expresia reprezentată de neterminal.

Grafuri orientate aciclice pentru expresii

Un graf orientat aciclic (GOA) pentru o expresie identifică subexpresiile comune din expresie. Ca şi un ASA, un GOA are un nod pentru fiecare subexpresie; un nod interior reprezintă un operator iar fiii săi reprezintă operanzii. Diferența este că un nod din GOA reprezentând o subexpresie comună are mai mulți "părinți".



GOA pentru expresia: a+a∗(b−c)+(b−c)∗d

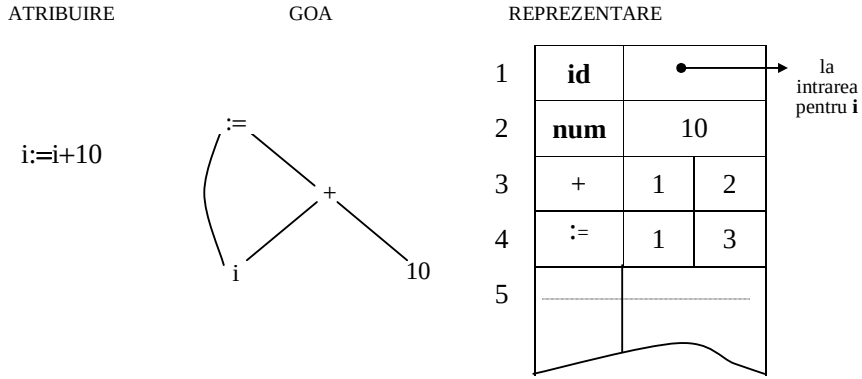
DDS din figura anterioară va construi un GOA în loc de ASA dacă se modifică operațiile pentru construirea nodurilor: funcția ce construieşte un nod verifică, mai întâi, dacă nu există deja un nod identic. Dacă da, funcția va returna un pointer la nodul anterior construit.

Exemplu. Secvența de instrucțiuni din figură construieşte un GOA cu condiția ca **nod** şi **frunză** să creeze noi noduri numai când este necesar, returnând pointeri la noduri existente ori de câte ori este posibil.

(1) p_1:=frunză (**id**, a)
(2) p_2:=frunză (**id**, a)
(3) p_3:=frunză (**id**, b)
(4) p_4:=frunză (**id**, c)
(5) p_5:=nod ('−', p_3, p_4)
(6) p_6:=nod ('∗', p_2, p_5)
(7) p_7:=nod ('+', p_1, p_6)
(8) p_8:=frunză (**id,** b)
(9) p_9:=frunză (**id**, c)
(10) p_{10}:=nod ('−', p_8, p_9)
(11) p_{11}:=frunză (**id**, d)
(12) p_{12}:=nod ('∗', p_{10}, p_{11})
(13) p_{13}:=nod ('+', p_7, p_{12})

Instrucțiuni pentru construirea GOA

În multe aplicații, nodurile sunt implementate ca înregistrări într-un tablou, ca în figură.



Noduri într-un GOA pentru i:=i+10 alocate dintr-un tablou

Referirea la un nod se realizează prin indicele său în tablou, numit **număr de valoare**.

Algoritm. Metoda numărului de valoare pentru a construi un nod în GOA.

Se presupune că nodurile sunt numerotate într-un tablou ca în figura anterioară şi că fiecare nod este referit prin numărul său de valoare. Fie **semnătura** unui nod operator un triplet <op, st, dr> constând din eticheta sa **op**, fiul stâng şi fiul drept.

Intrare: Eticheta **op**, nodul **st** şi nodul **dr**.

Ieşire: Un nod cu semnătura <op, st, dr>.

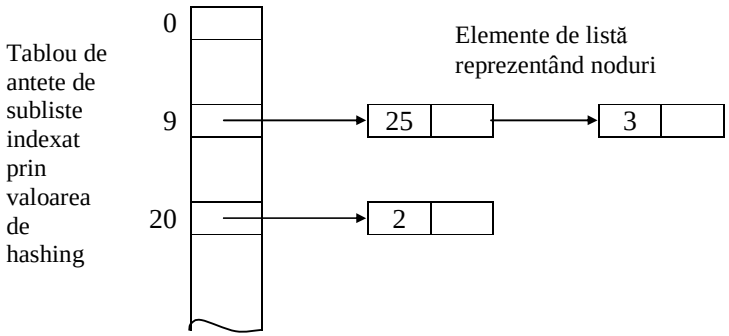
Metoda: Se caută în tablou un nod **m** cu eticheta **op**, fiul stâng **st** şi fiul drept **dr**. Dacă există, se returnează **m**; altfel se crează un nou nod **n**, cu eticheta **op**, fiul stâng **st** şi cel drept **dr** şi se returnează **n**.

Pentru a determina dacă nodul **m** este deja în tablou, toate nodurile create anterior se păstrează într-o listă şi se verifică fiecare nod din listă pentru a vedea dacă are semnătura dorită. Căutarea lui **m** poate fi mai eficientă utilizând k subliste şi o funcție de hashing **h** pentru a determina sublista.

Funcția **h** calculează numărul sublistei din valorile lui *op*, *st* şi *dr* şi, evident, pentru aceleaşi argumente va returna acelaşi număr. Dacă **m** nu e în sublista h(op, st, dr) atunci se crează un nou nod **n** şi se adaugă la această sublistă, aşa că următoarele căutări îl vor găsi aici.

Fiecare sublistă este implementată ca o listă înlănțuită. Fiecare intrare din listă reprezintă un nod. Antetele de subliste, constând din pointeri la prima intrare a listei, sunt numerotate într-un tablou. Numărul de sublistă returnat de h(op, st, dr) este un indice în acest tablou.

Acest algoritm poate fi adaptat pentru a se aplica nodurilor care nu sunt alocate secvențial, dintr-un tablou. În acest caz nodurile nu sunt secvențiale în memorie şi se referă prin pointeri.

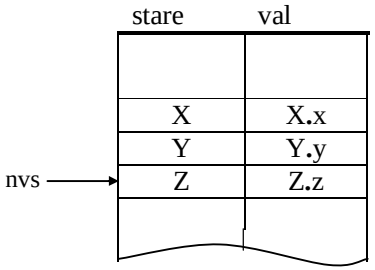


Structura de date pentru căutarea în subliste

GOA se pot, de asemenea, utiliza pentru a reprezenta mulțimi de expresii, deoarece GOA poate avea mai multe rădăcini.

Evaluarea ascendentă a definițiilor S–atributate

- După problemele legate de utilizarea DDS pentru specificarea traducerii urmează în mod logic cele de implementare a translatoarelor aferente. Este dificil de realizat un translator pentru o DDS arbitrară. Această muncă se simplifică substanțial considerând clase de DDS particulare, dar totuşi cu o largă aplicabilitate practică. Un prim exemplu sunt definițiile s–atributate (DSA).
- Un ANSIN ascendent poate evalua atributele sintetizate pe măsură ce analizează linia de intrare. Valorile acestor atribute (asociate cu simbolurile gramaticale), se pot păstra în stiva ANSIN. Valorile noilor atribute sintetizate sunt calculate la fiecare reducere, pe baza valorilor atributelor care apar în stivă pentru simbolurile gramaticale din partea dreaptă a producției cu care se realizează reducerea.
- În stiva unui ANSIN ascendent se păstrează informații despre subarborii care au fost analizați. Reținerea atributelor sintetizate se poate realiza în aceeaşi stivă în câmpuri suplimentare – varianta ASLR



Stiva ANSIN cu un câmp pentru atribute sintetizate

- Implementarea stivei este sub forma unei perechi de tablouri stare şi val. Fiecare intrare stare este un pointer la o tabelă AS LR(1). Simbolul gramatical este implicit în “stare” şi nu trebuie memorat în stivă, dar ne putem referi la “stare” prin simbolul gramatical unic, corespunzător. Dacă simbolul lui stare [i] este A, atunci val[i] va păstra valoarea atributului asociat cu nodul arborelui S corespunzător lui A.
- Pentru variantele de AS ascendente în care în stivă se memorează chiar simboluri gramaticale, asocierea este aceeaşi şi se realizează direct.
- Atributele sintetizate sunt evaluate înainte de fiecare reducere. Să presupunem că producția A→XYZ are asociată RS A.a:=f(X.x, Y.y, Z.z). Înainte ca XYZ să se reducă la A, valoarea atributului X.x este în val[vs], valoarea atributului Y.y este în val[vs–1] iar X.x în val[vs–2]. Dacă un simbol nu are atribute, atunci intrarea corespondentă în tabloul **val** este nedefinită. După reducere *vs* se decrementează cu 2, starea asociată cu A trece în **stare**[vs] şi valoarea atributului sintetizat A**.**a va fi în **val**[vs].

Exemplu. Să considerăm din nou DDS pentru un calculator de birou simplu. Atributele sintetizate din Arborele Sintactic adnotat pot fi evaluate în timpul analizei sintactice ascendente a liniei de intrare 3∗5+4**nl**.

Pentru evaluarea atributelor se modifică ANSIN prin executarea fragmentelor de cod prezentate în figură, imediat înainte de a face reducerea corespunzătoare.

Producția	Fragmente de cod
L→E**nl**	tipăreşte (val[vs−1])
E→E_1+T	val[nvs]:=val[vs−2]+val[vs]
E→T	
T→T_1∗F	val[nvs]:=val[vs−2]∗val[vs]
T→F	
F→(E)	val[nvs]:=val[vs−1]
F→**cifră**	

Implementarea unui calculator de birou cu un ASLR

În fragmentele de cod nu se arată cum se prelucrează variabilele "vs" şi "nvs". Aceasta ține de procesul de reducere, făcând parte din analiza sintactică propriu-zisă. În principiu, când se reduce o producție cu "r" simboluri în partea dreaptă, **nvs** primeşte valoarea "vs–r+1" iar după efectuarea fiecărui fragment de cod, "vs" ia valoarea lui "nvs".

Intrare	Stare sau simbol gramatical	Val	Producția utilizată
3∗5+4**nl**	—	—	
∗5+4**nl**	3	3	
∗5+4**nl**	F	3	F→**cifră**
∗5+4**nl**	T	3	T→F
5+4**nl**	T∗	3_	
+4**nl**	T∗5	3_5	
+4**nl**	T∗F	3_5	F→**cifră**
+4**nl**	T	15	T→T∗F
+4**nl**	E	15	E→T
4**nl**	E+	15_	
nl	E+4	15_4	
nl	E+F	15_4	F→**cifră**
nl	E+T	15_4	T→F
nl	E	19	E→E+T
	E**nl**	19_	
	L	19	L→E**nl**

Mişcările efectuate de translator pentru intrarea 3∗5+4

Definiții L–atributate

Când traducerea are loc în timpul AS, ordinea evaluării atributelor este legată de ordinea în care se creează nodurile prin metoda de AS. O ordine naturală care caracterizează multe metode de traducere, atât ascendente cât şi descendente, se obține prin aplicarea procedurii **preparcurgere** la rădăcina unui arbore sintactic. Această ordine de evaluare se numeşte **ordinea cu parcurgere în adâncime**. Metoda se poate aplica chiar dacă arborele sintactic nu se construieşte efectiv.

```
procedure preparcurgere (n : nod);
   begin
      for∗fiecare fiu m al lui n, de la stânga la dreapta do begin
            ∗evaluarea atributelor moştenite ale lui m;
            do preparcurgere (m);
      end;
      ∗evaluarea atributelor sintetizate ale lui n;
end;
```

Evaluarea spre adâncime a atributelor într-un arbore sintactic

Definițiile ale căror atribute se pot evalua întotdeauna "spre adâncime" se numesc definiții L–atributate (DLA L–left, pentru că informația pentru atribute pare să curgă de la stânga la dreapta). DLA includ toate DDS bazate pe gramatici LL(1).

Definiție: O DDS este **L–atributată** dacă fiecare atribut în orice RS pentru fiecare producție A→X_1 X_2 ... X_n este fie un atribut sintetizat, fie un atribut moştenit al lui X_j, 1≤j≤n depinzând numai de:

1. atributele simbolurilor X_1, X_2, ..., X_{j-1} de la stânga lui X_j în producție;
2. atributele moştenite ale lui A.

⇒ a) orice DSA este DLA

b) cerințele enunțate sunt exact cele necesare pentru evaluarea prin parcurgere în adâncime.

Exemplu. DDS din figură nu este L–atributată fiindcă atributul moştenit *Q.m* depinde de atributul *R.s* al simbolului gramatical de la dreapta sa.

Producția	Reguli semantice
A→LM	L.m:=f_1(A.m) M.m:=f_2(L.s) A.s:=f(M.s)
A→QR	R.m:=f_3(A.m) Q.m:=f_4(R.s) A.s:=f(Q.s)

O DDS ne-L–atributată

Scheme de traducere

O schemă de traducere (ST) este o GIC în care atributele sunt asociate cu simbolurile gramaticale iar acțiunile semantice, incluse între acolade, sunt inserate în părțile drepte ale producțiilor. ST sunt o notație utilă pentru specificarea traducerii în timpul AS. ST considerate în continuare pot avea atât atribute sintetizate cât şi moştenite.

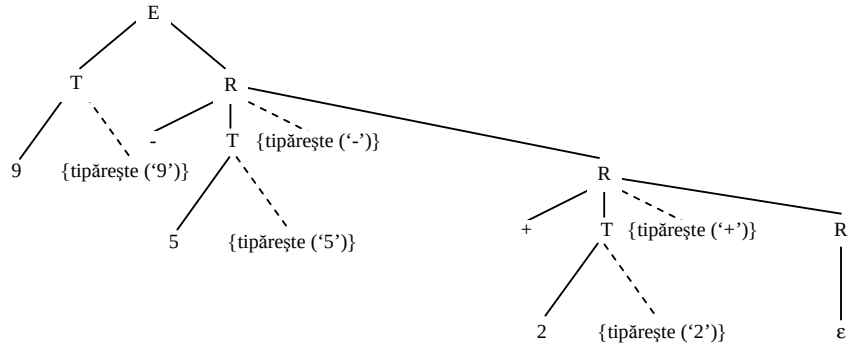
Exemplu. O ST simplă care transformă expresii infix în expresii postfix corespondente.

$E \rightarrow TR$

$R \rightarrow$ **opad** T{tipăreşte (**opad.**lexemă)}$R_1 \mid \varepsilon$

$T \rightarrow$ **num**{tipăreşte (**num.**val)}

În figură se arată arborele sintactic pentru şirul de intrare 9–5+2, cu fiecare acțiune semantică ataşată ca fiu al nodului ce corespunde părții stângi a producției. Astfel acțiunile se tratează ca şi când ar fi simboluri terminale. Numele şi operatorii sunt reprezentate efectiv în locul atomilor **num** şi **opad**. În *ordinea cu parcurgere în adâncime*, acțiunile semantice tipăresc “9 5 – 2 +”.



Arborele sintactic pentru 9–5+2 indicând acțiuni semantice

La proiectarea unei ST trebuie respectate anumite restricții pentru a asigura că o valoare de atribut este disponibilă când o acțiune se referă la ea.

Cazul cel mai simplu apare când sunt necesare numai atribute sintetizate. În acest caz se poate construi o schemă de traducere prin crearea unei acțiuni ce constă dintr-o atribuire pentru fiecare RS şi plasarea acestei acțiuni la sfârşitul părții drepte a producției asociate.

Exemplu: PRODUCȚIA REGULA SEMANTICĂ

$T \rightarrow T_1 * F$ $\quad$ $T.val := T_1.val \times F.val$

$\Rightarrow$următoarea producție şi acțiune semantică:

$T \rightarrow T_1 * F\{\ T.val := T_1.val \times F.val\ \}$

Dacă sunt atât atribute moştenite cât şi atribute sintetizate trebuie respectate următoarele trei cerințe:

1. Un atribut moştenit pentru un simbol din partea dreaptă a producției trebuie să fie calculat într-o acțiune înainte de acel simbol.
2. O acțiune nu trebuie să se refere la un atribut sintetizat al unui simbol situat în dreapta acțiunii.
3. Un atribut sintetizat pentru neterminalul din stânga poate fi calculat numai după ce au fost calculate toate atributele pe care le referă. Acțiunea prin care se calculează astfel de atribute se poate plasa, de obicei, la sfârşitul părții drepte a producției.

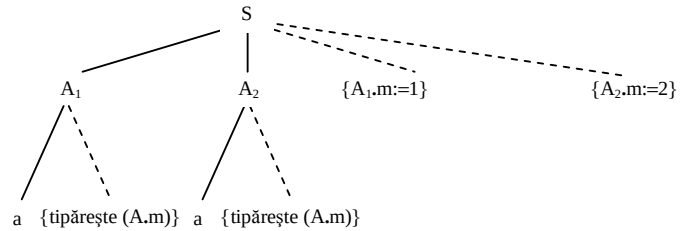
Se observă similitudinea dintre aceste cerințe şi cele impuse unei DDS pentru a fi DLA.

Exemplu: Următoarea ST nu satisface prima din cele trei cerințe:

$S \rightarrow A_1 A_2\{A_1.m := 1;\ A_2.m := 2\}$

$A \rightarrow a$ $\quad$ {tipăreşte (A.m)}

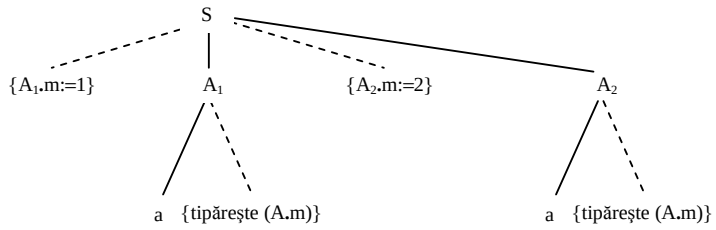
Efectul: *A.m* din a doua producție nu este definit când se încearcă să se tipărească valoarea sa, în timpul unei traversări spre adâncime a arborelui sintactic pentru şirul de intrare “aa”.



$\Downarrow$

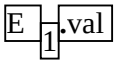
$S \rightarrow \{A_1.m := 1\}A_1\{A_2.m := 2\}A_2$

$A \rightarrow a$ {tipăreşte (A.m)}



Pentru construirea unei ST care satisface cele trei cerințe de mai sus se poate începe cu o DLA, aşa după cum rezultă din exemplul de mai jos, bazat pe limbajul pentru formule matematice **EQN**.

Pentru şirul de intrare "E inf 1.val", EQN plasează *E,1* şi *.val* în pozițiile şi mărimile relative redate în figură.



Plasarea căsuțelor dirijată de sintaxă

Exemplu. Din DLA din figura următoare se construieşte ST redat în figura reprezentând ST construită din DDS. Neterminalul C (pentru "căsuță") reprezintă o formulă. Producția "C→CC" reprezintă juxtapunerea a două căsuțe iar C→C **inf** C reprezintă plasarea celei de a doua căsuțe ca indice inferior.

Producția	Regulile semantice
S→C	C.dp:=10 S.in:=C.in
$C \to C_1\ C_2$	C_1.dp:=C.dp C_2.dp:=C.dp C.in:=max (C_1.in, C_2.in)
$C \to C_1$ **inf** C_2	C_1.dp:=C.dp C_2.dp:=reduce(C.dp) C.in:=depl (C_1.in, C_2.in)
C→**text**	C.in:=**text**.in×C.dp

DDS pentru dimensiunea şi înălțimea căsuțelor

Atributul moştenit **dp** (pentru dimensiune punct) reprezintă dimensiunea unei formule. Regula pentru producția "C→**text**" determină ca înălțimea normală a textului să fie înmulțită cu **dp** pentru a obține înălțimea efectivă. Atributul *text.in* poate să rezulte dintr-o tabelă cu dimensiunile caracterelor. Când se aplică producția "$C \to C_1C_2$", C_1 şi C_2 moştenesc **dp** de la *C* prin reguli de copiere. Înălțimea lui *C* (atributul sintetizat **in**) este maximul dintre înălțimile lui C_1 şi C_2.

Când se utilizează producția "$C \to C_1$**inf**C_2", funcția **reduce** realizează reducerea **dp** pentru C_2 cu 30%. Funcția **depl** permite deplasarea în jos a căsuței C_2 când se calculează înălțimea lui *C*. Nu sunt redate regulile care generează efectiv comenzile pentru imprimantă.

DDS din figura anterioară este L–atributată. Singurul atribut moştenit este **dp** al neterminalului *C*. Fiecare RS defineşte **dp** numai în funcție de atributul moştenit al neterminalului din stânga producției.

S→		{C.dp:=10}
	C	{S.in:=C.in}
C→		{C_1.dp:=C.dp}
	C_1	{C_2.dp:=C.dp}
	C_2	{C.in:=max (C_1.in, C_2.in)}
C→		{C_1.dp:=C.dp}
	C_1	
	inf	{C_2.dp:=reduce(C.dp)}
	C_2	{C.in:=depl (C_1.in, C_2.in)}
C→	**text**	{C.in:=**text**.in×C.dp}

ST construită din DDS din figura anterioară

ST din figura de mai sus este obținută prin inserarea în producții a atributelor ce corespund RS, conform celor trei cerințe date mai sus. Scrierea unei producții pe mai multe linii s-a făcut doar pentru a uşura citirea ST.

Traducerea descendentă

Pentru a putea explicita ordinea în care au loc acțiunile şi evaluările atributelor, în loc de DDS se utilizează ST.

Eliminarea recursivității de stânga dintr-o ST

Deoarece majoritatea operatorilor aritmetici sunt asociativi la stânga, utilizarea gramaticilor recursive la stânga pentru expresii este naturală. În continuare se va extinde algoritmul pentru eliminarea recursivității de stânga, prin includerea acțiunilor semantice în procesul de transformare a gramaticii corespunzătoare ST. Transformarea se aplică la ST cu atribute sintetizate. Ea permite implementarea majorității DDS din exemplul anterior pe baza AS predictivă.

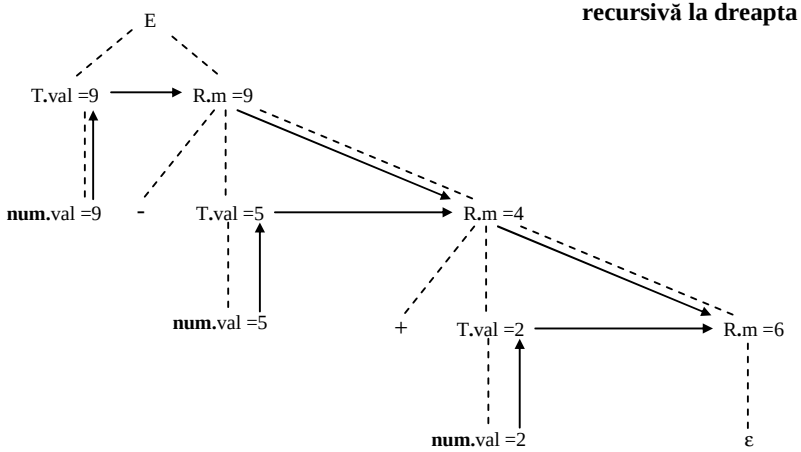
Exemplu.

$E \to E_1+T$	{E.val:=E_1.val+T.val}
$E \to E_1-T$	{E.val:=E_1.val−T.val}
$E \to T$	{E.val:=T.val}
$T \to (E)$	{T.val:=E.val}
T→**num**	{T.val:=**num**.val}

ST cu gramatica recursivă la stânga

$E \to T$	{R.m:=T.val}
R	{E.val:=R.s}
$R \to +$	
T	{R_1.m:=R.m+T.val}
R_1	{R.s:=R_1.s}
$R \to -$	
T	{R_1.m:=R.m−T.val}
R_1	{R.s:=R_1.s}
$R \to \varepsilon$	{R.s:=R.m}
$T \to ($	
E	
)	{T.val:=E.val}
T→**num**	{T.val:=**num**.val}

ST transformată cu gramatica recursivă la dreapta



Evaluarea expresiei "9-5+2"

Pentru a adapta alte ST recursive la stânga pentru AS predictivă utilizarea atributelor *R.m* şi *R.s* se va exprima mai general. Fie ST de mai jos:

$$\begin{array}{ll} A \to A_1.Y & \{A.a:=g(A_1.a,\ Y.y)\} \\ A \to X & \{A.a:=f(X.x)\} \end{array} \qquad \textbf{(7.2)}$$

Fiecare simbol gramatical are un atribut sintetizat scris cu litera mică corespunzătoare iar *f* şi *g* sunt funcții arbitrare.

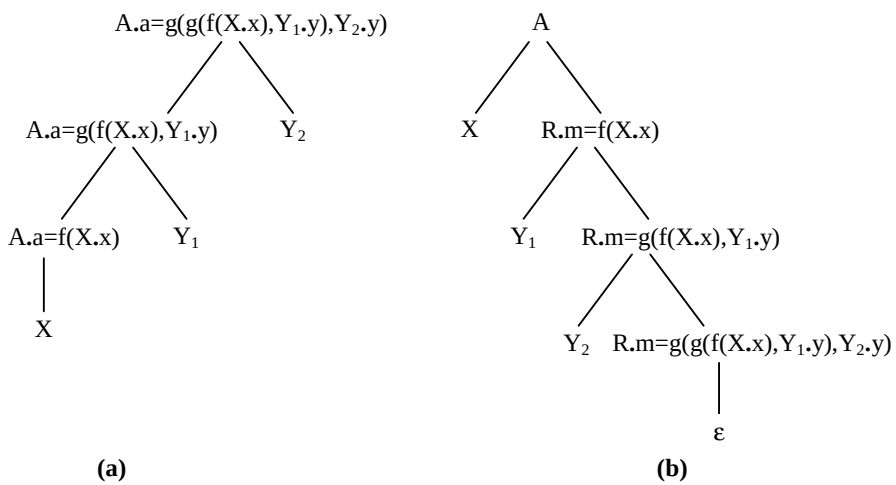
Algoritmul pentru eliminarea recursivității de stânga construieşte din **(7.2)** următoarea gramatică:

$$\begin{array}{l} A \to XR \\ R \to YR \mid \varepsilon \end{array} \qquad \textbf{(7.3)}$$

Luând în considerare acțiunile semantice, schema transformată devine:

$$\begin{array}{lll} A \to & X & \{R.m:=f(X.x)\} \\ & R & \{A.a:=R.s\} \\ R \to & Y & \{R_1.m:=g(R.m,Y.y)\} \\ & R_1 & \{R.s:=R_1.s\} \\ R \to & \varepsilon & \{R.s:=R.m\} \end{array} \qquad \textbf{(7.4)}$$

Schema transformată utilizează atributele **m** şi **s** pentru R ca în exemplul anterior Pentru a vedea echivalența rezultatelor din **(7.2)** şi **(7.4)**, se consideră cei doi arbori adnotați de mai jos corespunzători aceluiaşi şir de intrare XYY. Valoarea lui *A.a* este calculată conform cu **(7.2)**. Figura **(b)** conține calculul lui *R.m* în josul arborelui, conform cu **(7.4)**. Valoarea lui *R.m* de la bază este transferată în sus, neschimbată, ca *R.s* şi devine valoarea corectă a lui *A.a* la rădăcină (*R.s* nu e indicat în figura **(b)**).



Două moduri de a calcula valoarea unui atribut

Exemplu. Dacă DDS de construire a unui ASA pentru o expresie este transformată într-o ST, atunci producțiile şi acțiunile semantice pentru *E* devin:

$E \rightarrow E_1 + T\{E.p := nod(‘+’, E_1.p, T.p)\}$

$E \rightarrow E_1 - T\{E.p := nod(‘-’, E_1.p, T.p)\}$

$E \rightarrow T\{E.p := T.p\}$

Când se elimină recursivitatea de stânga din ST, neterminalul *E* corespunde lui *A* din **(7.2)**, iar şirurile $+T$ şi $-T$ din primele două producții corespund lui *Y*; neterminalul *T* din a treia producție corespunde lui *X*. În figură se prezintă ST transformată. Producțiile şi acțiunile semantice pentru *T* sunt similare cu cele din definiția originară.

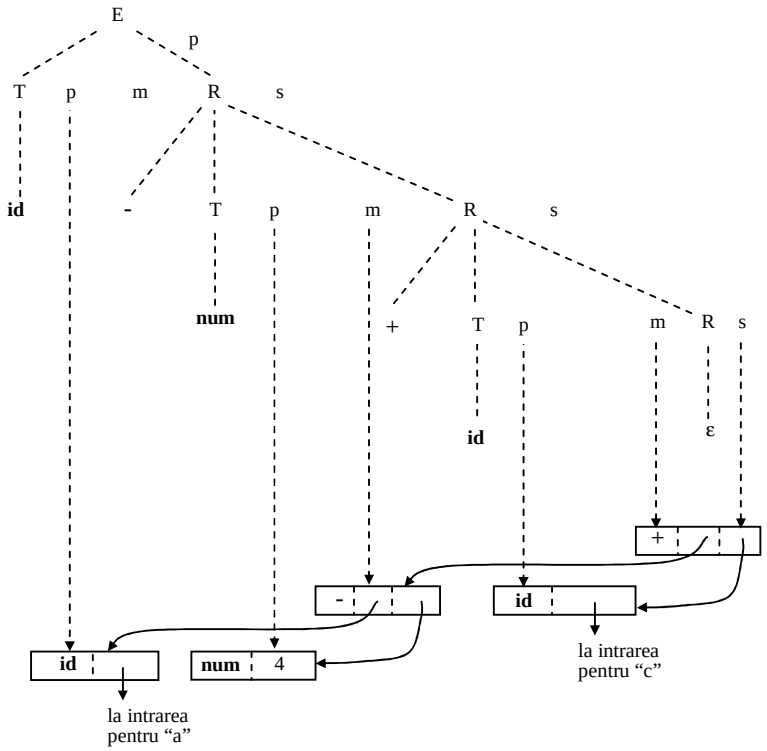
$E \rightarrow$	T	$\{R.m := T.p\}$
	R	$\{E.p := R.s\}$
$R \rightarrow$	+	
	T	$\{R_1.m := nod(‘+’, R.m, T.p)\}$
	R_1	$\{R.s := R_1.s\}$
$R \rightarrow$	–	
	T	$\{R_1.m := nod(‘-’, R.m, T.p)\}$
	R_1	$\{R.s := R_1.s\}$
$R \rightarrow$	ε	$\{R.s := R.m\}$
$T \rightarrow$	(	
	E	
	)	$\{T.p := E.p\}$
$T \rightarrow$	**id**	{T.p:=frunză(**id**,**id.**intrare)}
$T \rightarrow$	**num**	{T.p:=frunză(**num**,**num.**valoare)}

Scheme de traducere transformată pentru construirea arborelui sintactic

În figură se prezintă construirea arborelui sintactic pentru expresia “a–4+c” pe baza acțiunilor din ST. Atributele sintetizate sunt scrise în dreapta nodului pentru un simbol gramatical iar atributele moştenite, în stânga. O frunză în arborele sintactic e construită prin acțiunile asociate cu producțiile T→**id** şi T→**num**. La cel mai din stânga *T*, atributul ***T.p*** indică spre frunza pentru “a”. Un pointer la nodul pentru “a” este moştenit ca atribut ***R.m*** în partea dreaptă a lui “E→TR”.

Explicarea figurii: Când se aplică producția “R→ $-TR_1$“ la fiul drept al rădăcinii, ***R.m*** indică spre nodul pentru **a** iar ***T.p*** la nodul pentru **4**. Apoi se aplică funcția **nod** operatorului minus şi pointeri, pentru a construi nodul corespunzător lui “a–4”.

În final, când se aplică producția “R→ε”, ***R.m*** indică spre rădăcina întregului arbore sintactic. Acest pointer este returnat prin atributele **s** ale nodurilor pentru *R* (nu se indică în figură), până când devine valoarea lui ***E.p***.



Utilizarea atributelor moştenite pentru a construi arborele sintactic

Proiectarea unui translator predictiv

Următorul algoritm generalizează construirea unui analizor sintactic predictiv pentru a implementa o ST bazată pe o gramatică adecvată pentru AS descendentă (neambiguă, nerecursivă la stânga şi factorizată la stânga).

Algoritm. Construcția unui translator predictiv, dirijat de sintaxă.

Intrare: O ST dirijată de sintaxă cu o gramatică adecvată pentru AS predictivă.

Ieşire: Cod pentru un translator dirijat de sintaxă.

Metoda: Tehnica este o modificare a construcției AS predictiv.

1. Pentru fiecare neterminal A, se construieşte o funcție care are un parametru formal pentru fiecare atribut moştenit al lui *A* şi care returnează valorile atributelor sintetizate ale lui *A* (posibil ca o înregistrare, ca un pointer la o înregistrare sau utilizând mecanismul de apel prin referință pentru transferul parametrilor). Pentru simplitate, se presupune că fiecare neterminal are doar un atribut sintetizat. Funcția corespunzătoare lui *A* are câte o variabilă locală pentru fiecare atribut al fiecărui simbol gramatical care apare în producția pentru *A*.
2. Pe baza simbolului de intrare curent, codul pentru neterminalul *A* decide ce producție trebuie utilizată.
3. Codul asociat cu fiecare producție, face următoarele operații (se consideră atomii, neterminalele şi acțiunile din partea dreaptă a producției, de la stânga la dreapta):

   i. Pentru atomul *X* cu atributul sintetizat *x* se memorează valoarea lui *x* în variabila declarată pentru *X.x*. Apoi se generează un apel pentru a pune în corespondență atomul *X* şi a avansa intrarea.

   ii. Pentru neterminalul *B*, se generează o atribuire "c:=B(b_1,b_2,...,b_n)", cu un apel de funcție în partea dreaptă, unde b_1, b_2, ..., b_k sunt variabilele pentru atributele moştenite ale lui *B*, iar **c** este variabilă pentru atributul sintetizat al lui *B*.

   iii. Pentru o *acțiune*, se copiază codul în AS, înlocuind fiecare referire la un atribut prin variabila pentru acel atribut.

Exemplu. Gramatica din figura anterioară este LL(1) şi deci adecvată pentru AS descendentă. Din atributele neterminalelor gramaticii se obțin următoarele tipuri pentru argumentele şi rezultatele funcțiilor pentru *E*, *R* şi *T*. Fiindcă *E* şi *T* nu au atribute moştenite, ele nu au argumente.

function E:↑nod_arbore_sintaxă;
function R(most:↑nod_arbore_sintaxă):↑nod_arbore_sintaxă;
function T:↑nod_arbore_sintaxă;

Pentru a micşora translatorul, se pot combina cele două producții R *din* figura anterioară. Noile producții utilizează atomul **opad** pentru a reprezenta + şi –.

R→**opad**
  T {R_1**.**m:=nod(**opad.**lexemă, R**.**m, T**.**p)}
  R_1 {R**.**s:=R_1**.**s}
R→ε {R**.**s:=R**.**m}

Codul pentru *R* se bazează pe procedura de AS pură prezentată în figura următoare. Dacă simbolul de anticipare este **opad**, atunci se aplică producția "R→**opad**TR" utilizând procedura *avansează* pentru a citi următorul atom de intrare după **opad** şi apoi apelând procedurile pentru *T* şi *R*. În caz contrar, procedura nu face nimic, simulând producția R→ε. Această procedură va reprezenta scheletul funcției R din translator.

```
procedure R;
  begin
    if simb=opad then begin
      avansează (opad); T; R
    end
    else        {nu se execută nimic}
  end;
```

Procedura de AS pentru producțiile "R→<u>opad</u>TR|ε"

Funcția corespunzătoare lui R din figura următoare conține, în plus, codul pentru evaluarea atributelor. Valoarea lexicală **lexval** a atomului **opad** este memorată în **opadlexemă**, se citeşte următorul atom de intrare, se apelează *T* iar rezultatul este memorat în **p**.

Variabila m_1 corespunde atributului moştenit "$\mathbf{R_1}$**.**m" iar s_1, atributului sintetizat R_1**.**s. Instrucțiunea **return** returnează valoarea lui **s** înaintea părăsirii funcției. Funcțiile pentru *E* şi *T* sunt construite similar.

```
function R(most:↑nod_arbore_sintaxă):↑nod_arbore_sintaxă;
   var p, m1, s1, s:↑nod_arbore_sintaxă;
      opadlexemă: char;
   begin
      if simb=opad then begin
         {producția R→opadTR}
         opadlexema:=lexval;
         avansează (opad);
         p:=T;
         m1:=nod(opadlexema, most, p);
         s1:=R(m1);
         s:=s1
      end
      else s:=most; { producția R→ε}
      return s
   end;
```

Construcția prin descendență recursivă a arborilor sintactici


```
function E:↑nod_arbore_sintaxă;
   var p, pT, m, s:↑nod_arbore_sintaxă;
   begin
      pT:=T;
      m:=pT;
      s:=R(m);
      p:=s;
   return p
   end;
function T:↑nod_arbore_sintaxă;
   var p, pT:↑nod_arbore_sintaxă;
      intrare:↑TS;
      val: integer;
   begin
      if simb='(' then begin
         {producția T→(E)}
         avansează ('(');
         p:=E;
         avansează (')');
         pT:=T
      end else if simb=id then begin
            intrare:=lexval;
            avansează (id);
            pT:=frunză (id, intrare)
            end else if simb=id then begin
                  val:=lexval;
                  avansează (num)
                  pT:=frunză (num, val)
                  end else eroare();
      return pT
   end;
```