Prevođenje programskih jezika – beleške sa predavanja
Semantička analiza

Milan Banković

*Matematički fakultet,
Univerzitet u Beogradu

Jesenji semestar 2023/24.

Pregled

1 Uvod

2 Atributske gramatike

3 Stabla apstraktne sintakse

4 Provera tipova

Semantička analiza

Čemu služi semantička analiza?

Idealno bi bilo kada bismo na osnovu stabla apstraktne sintakse dobijenog nakon sintaksne analize mogli da generišemo kôd (ili izvršimo odgovarajuća izračunavanja, u slučaju interpretacije). Ovo, na žalost, nije moguće:

- programski jezici koje srećemo u praksi, zapravo nisu kontekstno slobodni jezici, u formalnom smislu
- gramatika programskog jezika, po pravilu, opisuje neki kontekstno slobodni nadskup tog programskog jezika
- to znači da ono što ,,prođe" sintaksnu analizu, ne mora da bude ispravna konstrukcija tog programskog jezika
- potrebno je primeniti dodatne provere, kako bi se eliminisale konstrukcije koje, iako sintaksno ispravne, semantički nemaju smisla

Ove dodatne provere se vrše u fazi semantičke analize.

Uvod

Uloge semantičke analize:

- provera i određivanje tipova izraza (engl. type checking,type-inference)
- provera i određivanje dometa identifikatora
- provera prava pristupa promenljivama (u OOP jezicima)
- provera ispravnosti poziva funkcija
- provera upotreba određenih naredbi (poput break i continue)
- ...

Atributske gramatike

Jedan od mehanizama za ugradnju ovih dodatnih provera u prevodilac su atributske gramatike koje opisujemo u nastavku.

Pregled

1 Uvod

2 Atributske gramatike

3 Stabla apstraktne sintakse

4 Provera tipova

Atributske gramatike

Definicija 1

Atributska gramatika je kontekstno slobodna gramatika u kojoj se simbolima pridružuju *atributi*, a pravilima se pridružuju *akcije* kojima se izračunavaju vrednosti ovih atributa:

- Atribut v simbola X označavaćemo sa $X.v$
- Akciju pridruženu pravilu $A \longrightarrow \gamma$ zapisivaćemo nakon samog pravila: $A \longrightarrow \gamma \ \{ \ akcija \ \}$
- U slučaju da se u pravilu neki simbol A pojavljuje više puta, tada se njegove pojave numerišu, kako bi se referisanje na njihove atribute moglo razlikovati (npr. $A_1.v$, $A_2.v$,...)

Napomena

U alatu YACC (Bison), atributi se u pravilima referišu sa $$ (atribut leve strane), odnosno $1, $2, . . . (atributi prvog, drugog, itd. simbola desne strane). Iako to formalno znači da svaki simbol može imati samo jedan atribut, tip tog atributa može biti strukturni tip, pa se na taj način simbolu efektivno može pridruživati više atributa.

Atributske gramatike

Upotrebe atributskih gramatika

- izračunavanje vrednosti izraza (kod interpretacije)
- određivanje stabla apstraktne sintakse
- određivanje i provera tipova izraza
- pridruživanje tipova identifikatorima u naredbama deklaracije
- određivanje dometa identifikatora
- generisanje kôda na međujeziku, na osnovu stabla apstraktne sintakse
- ...

Sintetizovani atributi

Definicija 2

Za atribut v simbola A atributske gramatike G kažemo da je sintetizovani atribut, ako se izračunava u okviru akcije pridružene pravilu $A \longrightarrow \alpha$, i to na osnovu vrednosti atributa simbola iz α.

Napomena

Sintetizovani atributi su najjednostavniji tip atributa, jer se njihove vrednosti mogu izračunavati obilaskom stabla izvođenja odozdo naviše:

- ovo znači da se vrednosti sintetizovanih atributa mogu izračunavati tokom parsiranja naviše
- upravo je ovo slučaj sa atributima $$, $1, $2, ... u YACC-u

Definicija 3

Gramatika je S-atributska, ako su svi atributi u njoj sintetizovani.

Primer – izračunavanje vrednosti izraza

Primer

Posmatrajmo sledeću atributsku gramatiku:

$$
\begin{array}{lcll}
E & \longrightarrow & E + T & \{E_1.v = E_2.v + T.v\} \\
E & \longrightarrow & T & \{E.v = T.v\} \\
T & \longrightarrow & T * F & \{T_1.v = T_2.v * F.v\} \\
T & \longrightarrow & F & \{T.v = F.v\} \\
F & \longrightarrow & (E) & \{F.v = E.v\} \\
F & \longrightarrow & a & \{F.v = val(a.lex)\}
\end{array}
$$

Primetimo da je ova gramatika S-atributska. Njome se, polazeći od vrednosti leksema koje odgovaraju tokenima u listovima (označenih sa $val(a.lex)$) dobijaju vrednosti svih podizraza, kao i vrednost celog izraza (izraženih atributom v). Ovaj primer demonstrira kako se atributske gramatike mogu koristiti u toku interpretacije, za izračunavanje vrednosti izraza na osnovu vrednosti konstanti i promenljivih (pročitanih iz tabele simbola) koje izraz sadrži.

Izračunavanje sintetizovanih atributa – parsiranje naviše

Postupak izračunavanja sintetizovanih atributa

Sintetizovani atributi se izračunavaju tokom sintaksne analize naviše:

- paralelno sa stekom simbola (stanja) održava se i stek atributa
  - pretpostavimo da za svaki simbol postoji tačno jedan sintetizovani atribut, te da između elemenata ova dva steka postoji ,,1-1" korespondencija
  - u praksi ako neki simbol ima više atributa, možemo ih smestiti u strukturu
  - ako neki simbol nema ni jedan atribut, možemo mu ,,veštački" pridružiti neki atribut čija je vrednost nebitna
- prilikom postavljanja simbola na stek, na atributski stek se postavljaju vrednosti njihovih atributa
- prilikom redukcije, izvršava se akcija pridružena pravilu po kome se vrši redukcija:
  - skidaju se simboli desne strane pravila sa steka, kao i njihovi atributi sa atributskog steka
  - na osnovu vrednosti tih atributa, izračunava se sintetizovani atribut leve strane
  - simbol leve strane se potiskuje na stek, a vrednost atributa simbola leve strane se potiskuje na atributski stek

Primer – izračunavanje atributa pri parsiranju naviše

Primer

(nastavak) Posmatrajmo atributsku gramatiku iz prethodnog primera. Pretpostavimo da prepoznajemo izraz $a + a * a$, pri čemu tokenima (koji predstavljaju celobrojne konstante), redom odgovaraju lekseme 2, 3 i 5. Neka val(a.lex) predstavlja vrednost odgovarajuće lekseme kao celobrojne konstante. Ova atributska gramatika će, tokom parsiranja naviše, izračunavati vrednost izraza $2 + 3 * 5$. Ovo parsiranje i izračunavanje atributa dato je sledećom tabelom:

Stek	*Atributski stek*	*Ulaz*	*Primenjena akcija*
$\perp_0$	$\square$	$a + a * a \dashv$	-
$\perp_0 a_5$	$\square 2$	$+ a * a \dashv$	S_5
$\perp_0 F_{10}$	$\square 2$	$+ a * a \dashv$	R_6
$\perp_0 T_3$	$\square 2$	$+ a * a \dashv$	R_4
$\perp_0 E_1$	$\square 2$	$+ a * a \dashv$	R_2
$\perp_0 E_1 +_2$	$\square 2 \square$	$a * a \dashv$	S_2
$\perp_0 E_1 +_2 a_5$	$\square 2 \square 3$	$* a \dashv$	S_5
$\perp_0 E_1 +_2 F_{10}$	$\square 2 \square 3$	$* a \dashv$	R_6
$\perp_0 E_1 +_2 T_{11}$	$\square 2 \square 3$	$* a \dashv$	R_4
$\perp_0 E_1 +_2 T_{11} *_4$	$\square 2 \square 3 \square$	$a \dashv$	S_4
$\perp_0 E_1 +_2 T_{11} *_4 a_5$	$\square 2 \square 3 \square 5$	$\dashv$	S_5
$\perp_0 E_1 +_2 T_{11} *_4 F_6$	$\square 2 \square 3 \square 5$	$\dashv$	R_6
$\perp_0 E_1 +_2 T_{11}$	$\square 2 \square 15$	$\dashv$	R_3
$\perp_0 E_1$	$\square 17$	$\dashv$	R_1
$\perp_0 E_1 \dashv_F$	$\square 17 \square$	ε	S_F

Vrednost $\square$ na atributskom steku označava ,,nebitnu" vrednost, pridruženu terminalima i neterminalima koji nemaju vrednost (poput $+$, $*$, $\perp$). Vrednost atributa koja je pridružena početnom simbolu E na kraju parsiranja predstavlja izračunatu vrednost izraza.

Etiketirano stablo

Definicija 4

Etiketirano stablo je stablo izvođenja u kome su svakom čvoru pridružene vrednosti njegovih atributa.

Napomena

U slučaju S-atributske gramatike, etiketirano stablo se može odrediti tokom parsiranja naviše, jer se prilikom kreiranja svakog čvora mogu odmah izračunati i njegovi atributi.

Primer

Za vežbu konstruisati etiketirano stablo za prethodni primer.

Nasleđeni atributi

Definicija 5

Neka je dato pravilo gramatike $A \longrightarrow \alpha B \beta$. Atribut $B.v$ je nasleđeni atribut ako se izračunava u akciji pridruženoj ovom pravilu, na osnovu atributa simbola A, kao i simbola iz $\alpha B \beta$.

Primedba

Atribut je, dakle, nasleđen ako se njegova vrednost računa u akciji pravila kod koga je odgovarajući simbol na desnoj strani pravila.

Definicija 6

Atributska gramatika je L-atributska ako su svi njeni atributi ili sintetizovani ili nasleđeni, pri čemu za svaki nasleđeni atribut $B.v$ koji se izračunava u akciji pridruženoj pravilu $A \longrightarrow \alpha B \beta$ važi da vrednost $B.v$ zavisi samo od atributa (nasleđenih ili sintetizovanih) simbola iz α, kao i nasleđenih atributa simbola A.

Izračunavanje nasleđenih atributa

Primedba

Primetimo da se nasleđeni atributi ne mogu izračunati tokom parsiranja naviše, jer mogu zavisiti od atributa svoje braće ili roditelja u stablu izvođenja:

- analiza naviše odgovara obilasku stabla od listova ka korenu, što nije odgovarajući poredak obilaska za nasleđene atribute

Postupak izračunavanja atributa kod *L*-atributskih gramatika

Najpre formiramo stablo izvođenja (parsiranjem naviše), a zatim ga obilazimo u dubinu, sa leva na desno:

- nasleđeni atributi se izračunavaju prilikom dolazne obrade odgovarajućeg čvora
  - u tom trenutku je već završena i dolazna i odlazna obrada leve braće tog čvora, kao i dolazna obrada roditelja
  - otuda su svi nasleđeni atributi leve braće i roditelja, kao i sintetizovani atributi leve braće već izračunati
- sintetizovani atributi se izračunavaju prilikom odlazne obrade odgovarajućeg čvora:
  - u tom trenutku je već završena i dolazna i odlazna obrada sve dece tog čvora
  - samim tim, poznate su sve vrednosti atributa od kojih sintetizovani atributi tog čvora mogu zavisiti

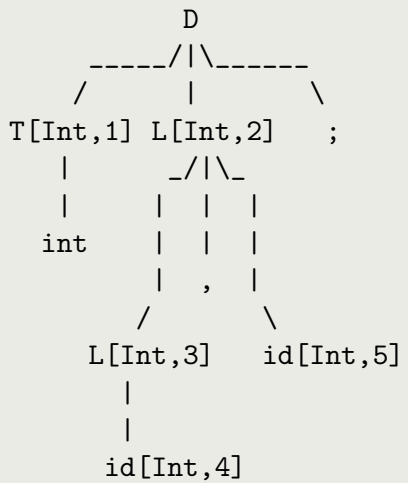
Primer – obrada deklaracija promenljivih

Primer

Posmatrajmo sledeću atributsku gramatiku:

$$
\begin{array}{lcll}
D & \longrightarrow & T\ L; & \{L.tip = T.tip\} \\
T & \longrightarrow & \texttt{int} & \{T.tip = \texttt{Int}\} \\
T & \longrightarrow & \texttt{double} & \{T.tip = \texttt{Double}\} \\
L & \longrightarrow & L, id & \{L_2.tip = L_1.tip; tabela\_simbola[id.lex].tip = L_1.tip\} \\
L & \longrightarrow & id & \{tabela\_simbola[id.lex].tip = L.tip\}
\end{array}
$$

Ova gramatika je L-atributska. Atribut $T.tip$ je sintetizovan, dok su ostali atributi nasleđeni. Ova gramatika ilustruje način na koji se u programskim jezicima poput C-a mogu identifikatorima pridruživati tipovi na osnovu deklaracije. Pritom, `Int` *i* `Double` *predstavljaju apstraktne reprezentacije tipova celih i realnih brojeva unutar prevodioca. Na primer, za deklaraciju* `int x, t;` *imali bismo sledeće anotirano stablo izvođenja:*



Notacija $X[vrednost, redosled]$ označava koja je vrednost atributa pridružena kom čvoru, kao i u kom redosledu su računati atributi.

Izračunavanje atributa pri parsiranju naniže

Šta sa sintaksnom analizom naniže?

- Izračunavanje atributa moguće je i prilikom parsiranja naniže
- Za razliku od parsiranja naviše, ovde se stablo izvođenja formira od korena ka listovima, i sa leva na desno
- Otuda se proces kreiranja stabla izvođenja može poistovetiti sa obilaskom stabla u dubinu
  - ovo je naročito vidljivo kod tehnike rekurzivnog spusta
  - dolazna obrada: na početku odgovarajućeg rekurzivnog poziva
  - odlazna obrada: na kraju rekurzivnog poziva
- Zbog toga se u slučaju parsiranja naniže izračunavanje atributa i kod *L*-atributskih gramatika može obaviti u toku parsiranja
  - setimo se da je kod parsiranja naviše to bilo moguće samo za *S*-atributske gramatike

Pregled

1 Uvod

2 Atributske gramatike

3 Stabla apstraktne sintakse

4 Provera tipova

Stabla apstraktne sintakse

Stabla apstraktne sintakse

- Ranije smo rekli da je krajnji cilj sintaksne analize kreiranje stabla apstraktne sintakse
- Za razliku od stabla izvođenja, ovo stablo ne sadrži detalje konkretne sintakse jezika (poput zagrada, separatora i ključnih reči)
- Stablo apstraktne sintakse sadrži samo informacije koje su neophodne sa stanovišta semantike (značenja) odgovarajuće jezičke konstrukcije

Kako kreirati apstraktna stabla?

- Jedan od načina da se kreiraju stabla apstraktne sintakse je da se koriste atributske gramatike
- Ispostavlja se da su takve gramatike uvek S-atributske, pa je formiranje stabla apstraktne sintakse uvek moguće tokom samog parsiranja (naniže ili naviše)
- Za razliku od stabla apstraktne sintakse, stablo izvođenja se obično ne formira eksplicitno, jer za tim nema potrebe

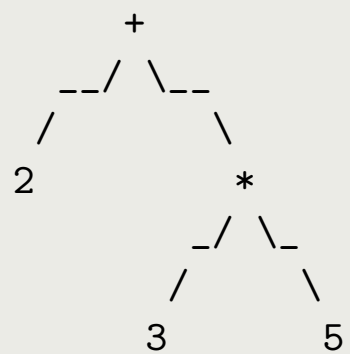
Primer – apstraktno stablo izraza

Primer

Atributska gramatika koja omogućava formiranje stabla apstraktne sintakse izraza je:

$$
\begin{array}{lcll}
E & \longrightarrow & E + T & \{E_1.pok = novi\_cvor('+', E_2.pok, T.pok)\} \\
E & \longrightarrow & T & \{E.pok = T.pok\} \\
T & \longrightarrow & T * F & \{T_1.pok = novi\_cvor('*', T_2.pok, F.pok)\} \\
T & \longrightarrow & F & \{T.pok = F.pok\} \\
F & \longrightarrow & (E) & \{F.pok = E.pok\} \\
F & \longrightarrow & a & \{F.pok = novi\_list(a.lex)\}
\end{array}
$$

Na primer, za izraz $2 + 3 * 5$, vrednost $E.pok$ na kraju parsiranja bila bi pokazivač na koren stabla:



Primer – apstraktno stablo izraza

Primer

Atributska gramatika bez leve rekurzije koja omogućava formiranje stabla apstraktne sintakse izraza je:

$$
\begin{array}{lcll}
E & \longrightarrow & TE' & \left\{ \begin{array}{l} vec = dodaj\_na\_pocetak(T.pok, E'.vec); \\ E.pok = kreiraj\_stablo(vec, '+'); \end{array} \right\} \\
E' & \longrightarrow & +TE' & \{E'_1.vec = dodaj\_na\_pocetak(T.pok, E'_2.vec);\} \\
E' & \longrightarrow & \varepsilon & \{E'.vec = prazan\_vektor();\} \\
T & \longrightarrow & FT' & \left\{ \begin{array}{l} vec = dodaj\_na\_pocetak(F.pok, T'.vec); \\ T.pok = kreiraj\_stablo(vec, '*'); \end{array} \right\} \\
T' & \longrightarrow & *FT' & \{T'_1.vec = dodaj\_na\_pocetak(F.pok, T'_2.vec);\} \\
T' & \longrightarrow & \varepsilon & \{T'.vec = prazan\_vektor();\} \\
F & \longrightarrow & (E) & \{F.pok = E.pok\} \\
F & \longrightarrow & a & \{F.pok = novi\_list(a.lex)\}
\end{array}
$$

Simboli F, E i T imaju atribut pok koji sadrži pokazivač na podstablo apstraktnog stabla koje odgovara tom simbolu. Sa druge strane, simboli E' i T' imaju atribut vec koji sadrži vektor pokazivača na podstabla koja odgovaraju podizrazima koji se sabiraju (odnosno množe) pod tim simbolom. Inicijalno, ti vektori se inicijalizuju kao prazni (u akcijama pridruženim pravilima $E' \longrightarrow \varepsilon$ i $T' \longrightarrow \varepsilon$). Pri svakoj primeni pravila $E' \longrightarrow +TE'$ i $T' \longrightarrow *FT'$ pokazivač na stablo koje odgovara simbolu T (odnosno F) se dopisuje na početak vektora. Najzad, prilikom primene pravila $E \longrightarrow TE'$ i $T \longrightarrow FT'$ vrši se dopisivanje odgovarajućeg stabla koje odgovara simbolu T (odnosno F) na početak vektora, a zatim se stabla u vektoru organizuju u levo asocijativno stablo primenom funkcije kreiraj_stablo():

```
kreiraj_stablo(vec, op):
   t = skini_sa_pocetka(vec)
   while not vec.empty():
      tp = skini_sa_pocetka(vec)
      t = kreiraj_cvor(op, t, tp)
   return t
```

Čitaocu ostavljamo da se uveri da je apstraktno stablo izraza $2+3*5$ isto kao u slučaju standardne gramatike izraza (prethodni slajd).

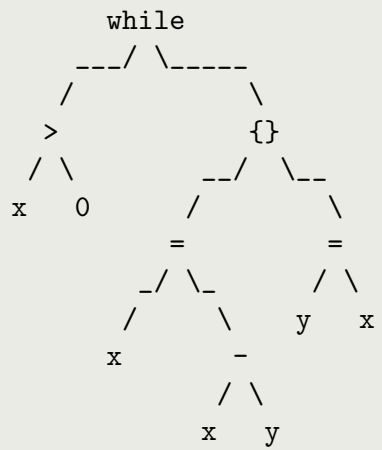
Primer – apstraktno stablo naredbi

Primer

U nastavku je dat fragment atributske gramatike koji omogućava formiranje stabla apstraktne sintakse naredbi:

$$
\begin{array}{lcll}
S & \longrightarrow & E; & \{S.pok = E.pok\} \\
S & \longrightarrow & while(E)\ S & \{S_1.pok = novi\_while\_cvor(E.pok, S_2.pok)\} \\
S & \longrightarrow & if(E)\ S\ else\ S & \{S_1.pok = novi\_if\_cvor(E.pok, S_2.pok, S_3.pok)\} \\
S & \longrightarrow & \{L\} & \{S.pok = L.pok\} \\
L & \longrightarrow & L\ S & \{L_1.pok = dodaj\_dete(L_2.pok, S.pok)\} \\
L & \longrightarrow & \varepsilon & \{L.pok = novi\_cvor\_slozene\_naredbe()\}
\end{array}
$$

while čvor ima dva deteta (stablo uslova i stablo naredbe), dok if čvor ima tri deteta (stablo uslova i stabla naredbi obe grane). Čvor složene naredbe može imati 0 ili više deteta, a to su stabla naredbi koje čine složenu naredbu. Ovaj čvor se inicijalno kreira bez dece, a onda se funkcijom dodaj_dete dodaju nova deca. Pretpostavka je da u ostatku gramatike za simbol E postoje akcije koje formiraju apstraktno stablo izraza (kao u prethodnim primerima). Na primer, stablo naredbe `while(x > 0) { x = x - y; y = x; }` *bi bilo:*



Oznaka `{}` *predstavlja čvor složene naredbe.*

Primer – izračunavanje vrednosti izraza

Primer

Ukoliko imamo formirano apstraktno stablo sintakse izraza, vrednost izraza možemo jednostavno izračunati sledećom funkcijom:

```
izracunaj_vrednost(cvor):
   if list(cvor):
     return val(cvor.lex)
   else:
     l = izracunaj_vrednost(cvor.levi)
     d = izracunaj_vrednost(cvor.desni)
     return primeni_operaciju(cvor.op, l, d)
```

Ako je čvor list, tada vraćamo vrednost pridruženu leksemi (promenljivoj ili konstanti) koja odgovara čvoru (u slučaju promenljive, tu vrednost čitamo iz tabele simbola). U suprotnom, rekurzivno izračunavamo vrednosti levog i desnog podstabla i primenjujemo operator koji odgovara čvoru na izračunate vrednosti.

Napomena

Primetimo da funkcija *primeni_operaciju*() može uključivati i bočne efekte – npr. u slučaju operatora dodele vrednost *d* biće upisana u tabelu simbola kao nova vrednost promenljive koja odgovara čvoru *cvor.levi*, dok će vrednost *l* biti ignorisana.

Primer – izvršavanje naredbi

Primer

(nastavak) Sada možemo nad apstraktnim stablom naredbi izvršiti sledeću funkciju:

```
izvrsi_naredbu(cvor):
   if izraz(cvor):
      izracunaj_vrednost(cvor)
   elsif while_naredba(cvor):
      b = izracunaj_vrednost(cvor.uslov)
      while b == True:
         izvrsi_naredbu(cvor.naredba)
         b = izracunaj_vrednost(cvor.uslov)
   elsif if_naredba(cvor):
      b = izracunaj_vrednost(cvor.uslov)
      if b == True:
         izvrsi_naredbu(cvor.if_grana)
      else:
         izvrsi_naredbu(cvor.else_grana)
   elsif slozena_naredba(cvor):
      for dete in deca(cvor):
         izvrsi_naredbu(dete)
```

Ova funkcija u okviru interpretatora proizvodi efekat odgovarajućih naredbi. Ako pretpostavimo da je program niz naredbi (tj. jedna složena naredba), tada bi se efekat celog programa proizveo pozivom ove funkcije za tu jednu složenu naredbu.

Pregled

1 Uvod

2 Atributske gramatike

3 Stabla apstraktne sintakse

4 Provera tipova

Provera tipova

Provera tipova

- Ubedljivo najznačajniji zadatak semantičke analize je provera i utvrđivanje tipova izraza
  - ovo je neophodno da bismo mogli da utvrdimo semantičku ispravnost, kao i preciznu semantiku svake od naredbi
- Tipovi izraza su po pravilu određeni tipovima promenljivih koje u njima učestvuju
  - kod statički tipiziranih jezika, tipovi promenljivih fiksirani su deklaracijama i poznati su u fazi prevođenja programa
  - kod dinamički tipiziranih jezika, tip promenljive određen je tipom vrednosti koja mu je dodeljena (i može se menjati tokom izvršenja programa).

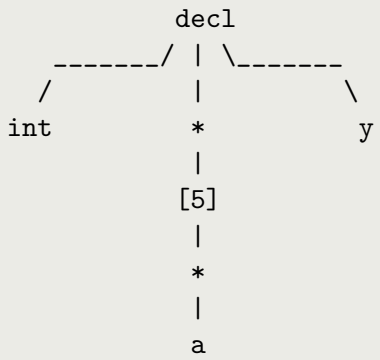
Primer – obrada deklaracija promenljivih

Primer

Sledeći primer predstavlja fragment gramatike C-ovskih deklaracija koja uključuje nizovske i pokazivačke tipove:

$$
\begin{array}{lcll}
D & \longrightarrow & T\ L; & \{D.pok = novi\_decl\_cvor(T.pok, L.lista);\} \\
T & \longrightarrow & \texttt{int} & \{T.pok = novi\_int\_list()\} \\
T & \longrightarrow & \texttt{double} & \{T.pok = novi\_double\_list()\} \\
L & \longrightarrow & L, P & \{L_1.lista = dodaj\_u\_listu(L_2.lista, P.pok)\} \\
L & \longrightarrow & P & \{L.lista = nova\_lista(P.pok)\} \\
P & \longrightarrow & *P & \{P_1.pok = novi\_pok\_cvor(P_2.pok)\} \\
P & \longrightarrow & N & \{P.pok = N.pok\} \\
N & \longrightarrow & N[c] & \{N_1.pok = novi\_niz\_cvor(c.lex, N_2.pok)\} \\
N & \longrightarrow & Z & \{N.pok = Z.pok\} \\
Z & \longrightarrow & (P) & \{Z.pok = P.pok\} \\
Z & \longrightarrow & id & \{Z.pok = novi\_id\_list(id.lex)\}
\end{array}
$$

Ova gramatika je S-atributska i izračunava stablo apstraktne sintakse naredbe deklaracije. decl čvor se kreira funkcijom novi_decl_cvor() i njegova deca biće stablo koje odgovara osnovnom tipu (`int` *ili* `double`*), kao i stabla svih deklaratora u toj deklaraciji. Lista stabala deklaratora se formira u atributu lista simbola L. Na primer, za deklaraciju* `int * (*a)[5], y;` *imali bismo sledeće stablo apstraktne sintakse deklaracije:*



Reprezentacija tipova

Reprezentacija tipova

Unutar prevodioca, tipovi se mogu predstaviti svojim apstraktnim stablima koja se formiraju na sledeći način:

- primitivnim tipovima (`int`, `char`, `double`,...) pridružujemo listove koji sadrže oznaku tipa (`Int`, `Char`, `Double`, ...)
- strukturnim (unijskim) tipovima pridružujemo čvor `Struct` (`Union`) čija su deca reprezentacije tipova članova strukture (unije)
- pokazivačkim tipovima pridružujemo čvor `Pok` čije je podstablo reprezentacija tipa na koji pokazuje pokazivač
- nizovskim tipovima pridružujemo čvor `Niz` koji sadrži celobrojnu vrednost dužine niza, a čije je podstablo reprezentacija tipa elementa niza

Primer – utvrđivanje tipa promenljive

Primer

(nastavak) Nakon formiranja stabla apstraktne sintakse, tipove identifikatorima je moguće pridružiti tako što se najpre odredi apstraktna reprezentacija tipa koji odgovara prvom detetu decl čvora (kod nas je to list `Int` *ili* `Double`*, ali u realnim situacijama može biti i npr. stablo koje predstavlja strukturu ili uniju), a zatim se za svu ostalu decu pozove sledeća funkcija (sa tim tipom kao argumentom):*

```
odredi_tip(cvor, tip):
   if list(cvor):
      tabela_simbola[cvor.lex].tip = tip
   elsif pok_cvor(cvor):
      odredi_tip(cvor.dete, Pok(tip))
   elsif niz_cvor(cvor):
      odredi_tip(cvor.dete, Niz(cvor.dim, tip))
```

Na primeru stabla sa prethodnog slajda, za prvi deklarator bismo, idući od korena ka listu, krenuli od `Int`*, pa bismo imali* $Pok(\texttt{Int})$*, pa* $Niz(5, Pok(\texttt{Int}))$ *i najzad* $Pok(Niz(5, Pok(\texttt{Int})))$*.*

Primer – određivanje i provera tipova izraza

Primer

(nastavak) Sada kada u tabeli simbola imamo informacije o deklarisanim tipovima promenljivih, možemo pristupiti proveri i određivanju tipova izraza u nastavku programa. Za programski jezik C, ovo možemo uraditi, npr. sledećom funkcijom (koja se primenjuje na čvor izraza u apstraktnom stablu):

```
tip_izraza(cvor):
   if list(cvor):
      return tabela_simbola[cvor.lex].tip
   elsif binarni_operator(cvor):
      tl = tip_izraza(cvor.levi)
      tr = tip_izraza(cvor.desni)
      if cvor.operator == '=':
         return tl
      else:
         return siri_tip(tl, tr)
   elsif unarni_operator(cvor):
      tp = tip_izraza(cvor.dete)
      if cvor.operator == '&':
         return pok(tp)
      elsif cvor.operator == '*':
         return tp.dete
      else:
         return tp
```

NAPOMENA: Funkcija nije kompletna, već služi samo za ilustraciju.