Limbaje formale și tehnici de compilare

analiza de tipuri

În cadrul analizei de domeniu (AT) se verifică dacă expresiile și instrucțiunile respectă regulile semantice (RS) specifice lor. Deoarece limbajul AtomC simplifică multe aspecte din C, există unele diferențe între el și regulile semantice din C:

- Deoarece nu există pointeri expliciți, vectorii fără dimensiune (ex: int v[]) pot fi folosiți pentru a transfera adresele vectorilor în apelurile de funcții
- Nu există aritmetica pointerilor aplicată la vectori (ex: adunare dintre un vector și un întreg)
- Vectorii pot fi folosiți doar pentru indexare (ex: v[i]) sau la apeluri de funcții, fără ca adresa lor să poată fi convertită la alte tipuri de date, gen **int**

Valori stânga și dreapta (left-value, right-value)

Valorile stânga (**lval**) au asociată o adresă la care se poate stoca o valoare. Din acest motiv, ele pot să apară în partea stângă a unei operații de atribuire. *Valorile dreapta* (**rval**) sunt doar valori, fără o adresă asociată și deci în ele nu se poate stoca o valoare. Din acest motiv, ele pot să apară doar în partea dreaptă a unei atribuiri.

```
int x;
double v[10];
x=1;    // corect: x este lval, deci la adresa lui se poate stoca o valoare
3=x;    // eroare: 3 este rval, deci nu are o adresa la care sa se poate stoca o
valoare
v[x]=3.14;    // corect: v[x] este o lval
```

Algoritmul de analiză de tipuri în AtomC

În AtomC AT se desfășoară la nivelul expresiilor și a instrucțiunilor care includ expresii, de exemplu condiția lui **while**. Algoritmul este următorul:

- Fiecărei reguli sintactice pentru expresii (ex: *exprAdd*) i se adaugă un atribut sintetizat, având tipul **Ret** și care returnează informațiile necesare AT, cum sunt:
  - tipul returnat de expresie
  - dacă este sau nu o valoare stângă
  - dacă este sau nu o valoare constantă
- Când se face AT pentru o expresie oarecare, prima oară se colectează *Ret*-urile de la subexpresiile sale, iar apoi se verifică dacă acestea sunt conforme cu regulile semantice. De exemplu, în *exprPostfix*, pentru operația de indexare, se preia *Ret*-ul pentru valoarea de indexat, care se verifică să fie un vector și *Ret*-ul pentru index, care se verifică să fie un tip convertibil la **int**.
- Se calculează (sintetizează) *Ret*-ul pentru expresia curentă și se returnează. De exemplu, în *exprAdd*, dacă un operand este de tip **int**, iar celălalt operand este de tip **double**, *Ret*-ul va avea tipul **double**.

Toate regulile și acțiunile necesare pentru AT sunt descrise în fișierul ”AtomC - analiza de tipuri”. Aici apar toate regulile sintactice pe care va trebui să le modificăm în analizorul sintactic, prin includerea în ele a codului necesar pentru AT. Fiecare regulă sintactică este precedată de comentarii care explică regulile semantice care îi sunt necesare.

Atenție: în acest fișier nu sunt repetate regulile de la etapa anterioară (analiza de domeniu). Din acest motiv, codul nou trebuie adăugat pe lângă codul implementat anterior, de la analiza de domeniu.

Toate funcțiile din AT sunt declarate în fișierul *at.h* și implementate în fișierul *at.c*.

După ce s-au implementat regulile semantice, nu mai trebuie făcut nimic altceva pentru includerea AT în compilator, deoarece acest modul nu are nevoie de alte inițializări suplimentare.

Implementarea regulilor semantice din regulile sintactice recursive la stânga

Regulile sintactice recursive la stânga, conform formulei de eliminare a recursivității stângi, se transformă în două reguli sintactice. În acest caz, regulile semantice vor trebui și ele distribuite în cele două reguli sintactice rezultate. Aceasta se realizează conform următorului algoritm, în care considerăm că avem o regulă sintactică A, din care vor rezulta regulile A și A':

- Ambele reguli sintactice rezultate vor avea aceiași parametri sintetizați și moșteniți (ex: "*out Ret *r*")
- Regulile semantice se vor distribui conform ramurilor din care fac parte, ca și cum aceste reguli sunt fragmente sintactice din ramurile respective. Astfel, regulile semantice din ramurile recursive se vor considera în fragmentele *alfa*, iar regulile semantice din ramurile nerecursive, se vor considera în fragmentele *beta* ale formulei de eliminare a recursivității stângi.
- Peste tot în formulele rezultate unde se include A', acestuia i se vor aplica atributele cu care s-a intrat în regula respectivă (ex: "*A[out Ret *r]: …. A'[r] …*")

Vom exemplifica acest algoritm prin implementarea regulilor semantice din regula sintactică *exprOr*:

```
// Ambii operanzi trebuie sa fie scalari si sa nu fie structuri
// Rezultatul este un int
exprOr[out Ret *r]: exprOr[r] OR {Ret right;} exprAnd[&right]
    {
    Type tDst;
    if(!arithTypeTo(&r->type,&right.type,&tDst))tkerr(iTk,"invalid operand type for
||");
    *r=(Ret){{TB_INT,NULL,-1},false,true};
    }
    | exprAnd[r]
```

După eliminarea recursivității stângi și considerând regulile semantice {...} ca făcând parte din ramurile recursive sau nerecursive, obținem următoarele reguli sintactice:

```
exprOr[out Ret *r]: exprAnd[r] exprOrPrim[r]

exprOrPrim[out Ret *r]: OR {Ret right;} exprAnd[&right]
    {
    Type tDst;
    if(!arithTypeTo(&r->type,&right.type,&tDst))tkerr(iTk,"invalid operand type for
||");
    *r=(Ret){{TB_INT,NULL,-1},false,true};
    }
    exprOrPrim[r]
    | eps
```

Se poate constata că peste tot unde apare regula derivată (*exprOrPrim*), aceasta a fost apelată cu atributul regulii în care apare (*"r"*).
În mod normal nu trebuie repetată în codul existent operația de eliminare a recursivității stângi, fiindcă aceasta deja a fost realizată. Este necesar doar să fie identificate locurile în care trebuie inserate regulile semantice. De exemplu, implementarea lui *exprOrPrim* poate fi:

```
bool exprOrPrim(Ret* r){
    if(consume(OR)){
        Ret right;
        if(exprAnd(&right)){
            Type tDst;
            if(!arithTypeTo(&r->type,&right.type,&tDst))tkerr(iTk,"invalid operand
type for ||");
            *r=(Ret){{TB_INT,NULL,-1},false,true};
            exprOrPrim(r);
            return true;
            }else tkerr(iTk,"invalid expression after ||");
        }
    return true;
    }
```

Testarea analizei de tipuri

Pentru a se testa AT, se folosește fișierul de intrare "testat.c". Acest fișier este corect din punctul de vedere al AT, deci nu trebuie să apară niciun mesaj de eroare.

Ulterior, acest fișier se va modifica astfel încât să se încalce câte o regulă semantică din AT (regulile semantice din fișierul "AtomC - analiza de tipuri"). De exemplu, pentru a se testa regula semantică "*Doar un array poate fi indexat*" din *exprPostfix*, se poate în funcția "*f*" înlocui "*text[i]*" cu "*ch[i]*", iar în acest caz va trebui să apară mesajul de eroare "*only an array can be indexed*".