Blokkok és alprogramok

Dr. Horpácsi Dániel
ELTE Informatikai Kar
2023-2024-2

Az előadás témája

Blokkok, hatókör, láthatóság, alprogramok

Emlékeztető: formális szemantikadefiníciók

- A programozási nyelvek szemantikája általában informálisan kerül leírásra, továbbá praktikusan a nyelv fordítóprogramja definiálja
- Ezek közül egyiket sem tekintjük formális definíciónak, nem használhatóak bizonyításhoz

Megoldás: használjuk a jó öreg matematikát és logikát!

Két alapvető megközelítés:

- Operációs (műveleti)
  - Strukturális

    Minden lépés modellezése
  - Természetes

    A kezdő- és végállapotok közötti reláció felállítása
- **Denotációs (leíró)**

  A jelentést matematikai objektumok hozzárendelésével adja meg

A következő lépés

- A magnyelvnek megadtuk az operációs és denotációs szemantikáját is
- És további nyelvi elemeknek is megmutattuk a formális szemantikáját:
  - Abort, nemdeterminisztikus választás, összefésülés, kivételkezelés

- Habár az **if** és **while** utasítások is blokkokat képeznek (magukba foglalnak újabb utasításokat), blokkra lokális változókat nem lehetett definiálni
- Most megnézzük, hogyan lehet leírni a blokkszerkezetet, illetve deklarációkat, névtereket
- Hatókör, láthatóság, élettartam? Alprogramok és azok hívása?

Kapcsolódó fogalmak

Gondoljuk át a következő fogalmakat:

- Utasítás
- Utasításblokk
- Változó
- Alprogram
- Deklaráció, definíció
- Névkötés
- Névtér
- Statikus/dinamikus scope (hatókör, láthatóság)

Szintaktikus kategóriák és metaváltozóik

n	$\in$	Num	(számliterálok)
x	$\in$	Var	(változók)
p	$\in$	Proc	(alprogramnevek)
a	$\in$	Aexp	(aritmetikai kifejezések)
b	$\in$	Bexp	(logikai kifejezések)
S	$\in$	Stm	(utasítások)
D_V	$\in$	$\mathrm{Dec_V}$	(változódefiníciók)
D_P	$\in$	$\mathrm{Dec_P}$	(alprogramdefiníciók)

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid \text{-}a \\
b & ::= & \textbf{true} \mid \textbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & \textbf{skip} \mid x := a \mid S_1; S_2 \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \mid \textbf{while}\ b\ \textbf{do}\ S \\
 & & \mid\ \textbf{begin}\ D_V\ D_P\ S\ \textbf{end} \mid \textbf{call}\ p \\
D_V & ::= & \textbf{var}\ x := a\ ;\ D_V \mid \varepsilon \\
D_P & ::= & \textbf{proc}\ p\ \textbf{is}\ S\ ;\ D_P \mid \varepsilon
\end{array}
$$

(Névkötés)

$x := 1 \; ; \; y := x + 1$

begin var $x := 1 \; ;$ **var** $y := 1 \; ; \; y := x + 1$ **end**

```
begin var x := 1 ; var y := 1 ; var z := 1 ;
    begin var x := 2 ;
        y := x + 1 ;
        x := x + 4
    end ;
    z := x + 1
end
```

(Statikus/dinamikus hatókör)

```
begin
    var x := 1 ; var y := 1 ; var z := 1 ; proc p is x := 0 ;
    begin
        var x := 2 ;
        call p ; y := x + 1
    end ;
    z := x + 1
end
```

Finomítsuk a memóriafogalmat: location (cím)

Az eddigi memóriállapotokban a változókhoz rendeltük az értékeket:

$$\text{State} = \text{Var} \to \mathbb{Z}$$

Mostantól a blokkok (lokális változók) miatt egy név több különböző változót jelölhet. A névhez rendelt érték függ a scope-tól.

- Az értékek helyett társítsunk memóriacímeket a nevekhez:

$$\text{Env}_\text{V} = \text{Var} \to \text{Loc}$$

- És vezessük be a tár fogalmát, amely a címhez értéket rendel:

$$\text{Store} = \text{Loc} \to \mathbb{Z}$$

- Az egyszerűség kedvéért a címeket most egész számokkal reprezentáljuk:

$$\text{Loc} = \mathbb{Z}$$

Változókörnyezet

- A változónevekhez címeket rendelünk:

$$\mathrm{Env_V} = \mathrm{Var} \to \mathrm{Loc}$$

- És a tárral adjuk meg, milyen érték van az adott címen:

$$\mathrm{Store} = \mathrm{Loc} \cup \{\mathit{next}\} \to \mathbb{Z}$$

(Trükk: a Loc halmazhoz itt hozzáveszünk egy extremális elemet, a *next* szimbólumot, amelyhez nem változóértéket rendelünk majd, hanem memóriacímet.)

- A következő szabad címet a *new* függvény számítja ki:

$$\mathit{new} : \mathrm{Loc} \to \mathrm{Loc}$$

$$\mathit{new}\ n = n + 1$$

Változó értékének lekérdezése

A változó értékét mostantól a változókörnyezet (env_V) és a tár (sto) kombinációból nyert állapot (s) alapján határozzuk meg.

$$s = sto \circ env_V$$

Ezt az összefüggést fogalommá emeljük, ez lesz a változólekérdezés (lookup) függvény:

$$lookup : \mathrm{Env_V} \to \mathrm{Store} \to \mathrm{State}$$

$$lookup\ env_V\ sto = sto \circ env_V$$

ahol

$$env_V \in \mathrm{Var} \to \mathrm{Loc} \quad \text{és} \quad sto \in \mathrm{Loc} \cup \{next\} \to \mathbb{Z}$$

$$\mathcal{S}'_{ds} : \mathrm{Stm} \to \mathrm{Env_V} \to (\mathrm{Store} \hookrightarrow \mathrm{Store})$$

Címekkel bővített denotációs szemantika

$$\mathcal{S}'_{ds}[\![\textbf{skip}]\!]env_V = id_{\mathrm{Store}}$$

$$\mathcal{S}'_{ds}[\![x := a]\!]env_V\ sto = sto[l \mapsto \mathcal{A}[\![a]\!](lookup\ env_V\ sto)]$$

$$\text{ahol } l = env_V(x)$$

$$\mathcal{S}'_{ds}[\![S_1; S_2]\!]env_V = (\mathcal{S}'_{ds}[\![S_2]\!]env_V) \circ (\mathcal{S}'_{ds}[\![S_1]\!]env_V)$$

$$\mathcal{S}'_{ds}[\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2]\!]env_V =$$

$$cond(\mathcal{B}[\![b]\!] \circ (lookup\ env_V), \mathcal{S}'_{ds}[\![S_1]\!]env_V, \mathcal{S}'_{ds}[\![S_2]\!]env_V)$$

$$\mathcal{S}'_{ds}[\![\textbf{while } b \textbf{ do } S]\!]env_V = \mathrm{FIX}\ F$$

$$\text{ahol } F\ g = cond(\mathcal{B}[\![b]\!] \circ (lookup\ env_V), g \circ (\mathcal{S}'_{ds}[\![S]\!]env_V), id_{\mathrm{Store}})$$

$$cond : (\mathrm{Store} \to \mathrm{Boolean}) \times (\mathrm{Store} \hookrightarrow \mathrm{Store}) \times (\mathrm{Store} \hookrightarrow \mathrm{Store}) \to (\mathrm{Store} \hookrightarrow \mathrm{Store})$$

A While kibővített szemantikája

- A memóriaállapot valamivel kevésbé absztrakt modelljével lehetővé tettük scope-ok bevezetését, blokkra lokális változószimbólumok deklarációját
- Bizonyítható, hogy a fent leírt szemantika ekvivalens a korábbi denotációs szemantikával:

$$\mathcal{S}_{ds}[\![S]\!] \circ (\mathit{lookup}\ \mathit{env}_V) = (\mathit{lookup}\ \mathit{env}_V) \circ (\mathcal{S}'_{ds}[\![S]\!]\mathit{env}_V)$$

(minden env_V környezetre)

- Hogyan írjuk le a szemantikáját a blokkoknak és az alprogramhívásoknak?

$$\mathcal{S}_{ds}[\![\textbf{begin}\ D_V\ D_P\ S\ \textbf{end}]\!] = ?$$
$$\mathcal{S}_{ds}[\![\textbf{call}\ p]\!] = ?$$

Változódeklarációk szemantikája

- A szintaxisban definiáltunk két új kategóriát: a változódeklarációkat és az eljárásdeklarációkat
- A denotációs szemantikában megszokott módon ezekhez egy-egy szemantikus domain és szemantikus függvény fog tartozni

- Definiáljuk a változódeklaráció szemantikáját:

$$\mathcal{D}^V_{ds} : \mathrm{Dec_V} \to \mathrm{Env_V} \times \mathrm{Store} \to \mathrm{Env_V} \times \mathrm{Store}$$

$$\mathcal{D}^V_{ds}[\![\textbf{var } x := a \ ; \ D_V]\!](env_V, sto) =$$
$$\quad \mathcal{D}^V_{ds}[\![D_V]\!](env_V[x \mapsto l], sto[l \mapsto v][next \mapsto new\ l])$$
$$\qquad \text{ahol } l = sto\ next \quad \text{és} \quad v = \mathcal{A}[\![a]\!](lookup\ env_V\ sto)$$

$$\mathcal{D}^V_{ds}[\![\varepsilon]\!] = id_{\mathrm{Env_V} \times \mathrm{Store}}$$

Alprogramkörnyezet

Hasonlóan a kivételekhez, az eljárások meghívásakor (odaugráskor) tudnunk kell, mi az alprogram jelentése. Ehhez bevezetjük az alprogramkörnyezet fogalmát:

$$\mathrm{Env_P} = \mathrm{Proc} \to (\mathrm{Store} \hookrightarrow \mathrm{Store})$$

És a változódeklarációkhoz hasonlóan definiáljuk az eljárások deklarációinak szemantikáját is:

$$\mathcal{D}^P_{ds} : \mathrm{Dec_P} \to \mathrm{Env_V} \to \mathrm{Env_P} \to \mathrm{Env_P}$$

$$\mathcal{D}^P_{ds}[\![\textbf{proc } p \textbf{ is } S \ ; \ D_P]\!] env_V \ env_P = \mathcal{D}^P_{ds}[\![D_P]\!] env_V \ (env_P[p \mapsto g])$$

ahol $g = \mathcal{S}_{ds}[\![S]\!] env_V \ env_P$

$$\mathcal{D}^P_{ds}[\![\varepsilon]\!] env_V = id_{\mathrm{Env_P}}$$

A kiterjesztett nyelv leíró szemantikája

A kiegészített nyelven írt programok jelentése függ a változó- és alprogramkörnyezettől. Emiatt az előző előadáson látottakhoz hasonlóan kibővítjük a szemantikus függvényünket újabb paraméterekkel:

$$S_{ds} : \mathrm{Stm} \to \mathrm{Env_V} \to \mathrm{Env_P} \to (\mathrm{Store} \hookrightarrow \mathrm{Store})$$

A környezeteket a deklarációs részek állítják be:

Címekkel bővített denotációs szemantika

$$\mathcal{S}_{ds}[\![\textbf{begin } D_V\ D_P\ S\ \textbf{end}]\!] env_v\ env_P\ s = \mathcal{S}_{ds}[\![S]\!] env'_V\ env'_P\ s'$$
$$\text{ahol } \mathcal{D}^V_{ds}[\![D_V]\!](env_V, s) = (env'_V, s')$$
$$\text{és } \mathcal{D}^P_{ds}[\![D_P]\!] env'_V\ env_P = env'_P$$

$$S_{ds} : \mathrm{Stm} \to \mathrm{Env_V} \to \mathrm{Env_P} \to (\mathrm{Store} \hookrightarrow \mathrm{Store})$$

Címekkel bővített denotációs szemantika

$$\mathcal{S}_{ds}[\![\textbf{skip}]\!]e_v\ e_p = id_{\mathrm{Store}}$$

$$\mathcal{S}_{ds}[\![x := a]\!]e_v\ e_p\ sto = sto[l \mapsto \mathcal{A}[\![a]\!](lookup\ e_v\ sto)]$$

$$\text{ahol } l = e_v(x)$$

$$\mathcal{S}_{ds}[\![S_1; S_2]\!]e_v\ e_p = (\mathcal{S}_{ds}[\![S_2]\!]e_v\ e_p) \circ (\mathcal{S}_{ds}[\![S_1]\!]e_v\ e_p)$$

$$\mathcal{S}_{ds}[\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2]\!]e_v\ e_p =$$

$$cond(\mathcal{B}[\![b]\!] \circ (lookup\ env_V), \mathcal{S}_{ds}[\![S_1]\!]e_v\ e_p, \mathcal{S}_{ds}[\![S_2]\!]e_v\ e_p)$$

$$\mathcal{S}_{ds}[\![\textbf{while } b \textbf{ do } S]\!]e_v\ e_p = \mathrm{FIX}\ F$$

$$\text{ahol } F\ g = cond(\mathcal{B}[\![b]\!] \circ (lookup\ e_v), g \circ (\mathcal{S}_{ds}[\![S]\!]e_v\ e_p), id_{\mathrm{Store}})$$

$$\mathcal{S}_{ds}[\![\textbf{begin } D_V\ D_P\ S \textbf{ end}]\!]e_v\ e_p\ s = \mathcal{S}_{ds}[\![S]\!]e'_v\ e'_p\ s'$$

$$\text{ahol } \mathcal{D}^V_{ds}[\![D_V]\!](e_v, s) = (e'_v, s')$$

$$\text{és } \mathcal{D}^P_{ds}[\![D_P]\!]e'_v\ e_p = e'_p$$

$$\mathcal{S}_{ds}[\![\textbf{call } p]\!]e_v\ e_p = e_p(p)$$

- Meghívhatja egy eljárás saját magát? Ha igen, akkor az eljáráskörnyezetnek már tartalmaznia kell az eljárást, amikor még nem is dolgoztuk fel a deklarációját...

$$\mathcal{D}^P_{ds}[\![\textbf{proc } p \textbf{ is } S\ ;\ D_P]\!]env_V\ env_P = \mathcal{D}^P_{ds}[\![D_P]\!]env_V\ (env_P[p \mapsto g])$$
$$\text{ahol } g = \mathcal{S}_{ds}[\![S]\!]env_V\ (env_P[p \mapsto g])$$

A ciklus szemantikájának definíciójában is egy rekurzív formulából indultunk ki!

- Megoldás: fixpont kombinátor

$$\mathcal{D}^P_{ds}[\![\textbf{proc } p \textbf{ is } S\ ;\ D_P]\!]env_V\ env_P = \mathcal{D}^P_{ds}[\![D_P]\!]env_V\ (env_P[p \mapsto FIX\ F])$$
$$\text{ahol } F\ g = \mathcal{S}_{ds}[\![S]\!]env_V\ (env_P[p \mapsto g])$$

$$\mathcal{D}^P_{ds}[\![\varepsilon]\!]env_V = id_{\mathrm{Env_P}}$$

Az eredeti példánk

Legyen S a következő program:

```
begin
    var x := 1 ; proc p is x := 0 ;
    begin
        var x := 2 ;
        call p
    end
end
```

Tegyük fel, hogy létezik env_V és env_P kezdeti környezetek és egy tár, sto, amelyre $sto\ next = 12$.

$$\mathcal{S}_{ds}[\![S]\!] env_V\ env_P\ sto = ?$$

begin var $x := 1$; **proc** p **is** $x := 0$; … **end**

- A változódeklaráció szemantikája:

$$\mathcal{D}^V_{ds}[\![\textbf{var } x := 1 \ ; \ \varepsilon]\!](env_V, sto) = \mathcal{D}^V_{ds}[\![\varepsilon]\!](env_V[x \mapsto 12], sto[12 \mapsto 1][next \mapsto 13]) = (env_V[x \mapsto 12], sto[12 \mapsto 1][next \mapsto 13])$$

- Az eljárás szemantikája:

$$\mathcal{D}^P_{ds}[\![\textbf{proc } p \textbf{ is } x := 0 \ ; \ \varepsilon]\!] env_V[x \mapsto 12] \ env_P = \mathcal{D}^P_{ds}[\![\varepsilon]\!] env_V[x \mapsto 12] \ env_P[p \mapsto g] = env_P[p \mapsto g]$$

ahol $g \ sto = \mathcal{S}_{ds}[\![x := 0]\!] env_V[x \mapsto 12] \ env_P \ sto = sto[12 \mapsto 0]$

Most már meghatároztuk, hogy a külső blokk szemantikája a következő:

$$\mathcal{S}_{ds}[\![\textbf{begin var } x := 1 \ ; \ \textbf{proc } p \textbf{ is } x := 0 \ ; \quad \ldots \quad \textbf{end}]\!] env_V \ env_P \ sto = \mathcal{S}_{ds}[\![\ldots]\!] env_V[x \mapsto 12] \ env_P[p \mapsto g] \ sto[12 \mapsto 1][next \mapsto 13]$$

ahol ... a **begin var** $x := 2$; **call** p **end** helyett áll

Határozzuk meg a belső blokk jelentését is, hogy megkapjuk a teljes szemantikát:

$$\mathcal{S}_{ds}[\![\textbf{begin var } x := 2 \ ; \ \textbf{call } p \textbf{ end}]\!] \ env_V[x \mapsto 12] \ env_P[p \mapsto g] \ sto[12 \mapsto 1][next \mapsto 13]$$

begin var $x := 2$; **call** p **end**

- A változódeklaráció:

$$\mathcal{D}^V_{ds}[\![\textbf{var } x := 2 \ ; \ \varepsilon]\!](env_V[x \mapsto 12], sto[12 \mapsto 1][next \mapsto 13]) = \\ \mathcal{D}^V_{ds}[\![\varepsilon]\!](env_V[x \mapsto 13], sto[12 \mapsto 1][13 \mapsto 2][next \mapsto 14]) = \\ (env_V[x \mapsto 13], sto[12 \mapsto 1][13 \mapsto 2][next \mapsto 14])$$

- Az eljárásdeklaráció:

$$\mathcal{D}^P_{ds}[\![\varepsilon]\!] env_V[x \mapsto 13] \ env_P[p \mapsto g] = env_P[p \mapsto g]$$

$$\mathcal{S}_{ds}[\![\textbf{begin var } x := 2 \ ; \ \textbf{call } p \ \textbf{end}]\!] \\ (env_V[x \mapsto 12]) \ (env_P[p \mapsto g]) \ (sto[12 \mapsto 1][next \mapsto 13]) = \\ \mathcal{S}_{ds}[\![\textbf{call } p]\!] \\ (env_V[x \mapsto 13]) \ (env_P[p \mapsto g]) \ (sto[12 \mapsto 1][13 \mapsto 2][next \mapsto 14])$$

Az eredeti példánk

$$\mathcal{S}_{ds}[\![\textbf{begin var } x := 2 \ ; \ \textbf{call } p \ \textbf{end}]\!]$$
$$\quad (env_V[x \mapsto 12]) \ (env_P[p \mapsto g]) \ (sto[12 \mapsto 1][next \mapsto 13]) =$$
$$\mathcal{S}_{ds}[\![\textbf{call } p]\!]$$
$$\quad (env_V[x \mapsto 13]) \ (env_P[p \mapsto g]) \ (sto[12 \mapsto 1][13 \mapsto 2][next \mapsto 14]) =$$
$$g \ (sto[12 \mapsto 1][13 \mapsto 2][next \mapsto 14]) =$$
$$sto[12 \mapsto 0][13 \mapsto 2][next \mapsto 14]$$

Mivel 12 a külső x címe és 13 a belső x címe, látható, hogy az alprogramhívás a külső változót módosítja, tehát statikus hatóköri szabályokat alkalmaz a szemantikadefiníciónk.

λ

A While nyelv, a fent bemutatott eljárások és blokkszerkezet végrehajtható denotációs szemantikája elérhető a kurzus anyagai között. A leíró jellegű szemantikát Haskellben definiáltuk.