Nyelvi kiegészítések az operációs szemantikában

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

Programozási nyelvi elemek szemantikája

Abort, nemdeterminisztikusság, konkurencia

Előszó

- Az előző előadásokban definiáltuk az alapvető imperatív konstrukciók operációs szemantikáját

- A valódi programozási nyelvek ennél sokkal komplexebbek
- Rengeteg nyelvi elemet tartalmaznak, amelyek a programozó által használt absztrakciókat implementálják
- Ezeknek köszönhető, hogy tömören, kényelmesen tudunk kifejező, megbízható programokat írni

- Gondoljuk át, melyik nyelvben hogyan írhatunk
  - Elágazásokat és ciklusokat
  - Deklarációkat, blokkokat, alprogramokat
  - Terminálást, kivételeket, párhuzamosítást
  - Rekurzív és névtelen függvényeket
  - Lusta vagy mohó módon kiszámítható kifejezéseket
  - Adattípusokat és típuskonverziókat

- Egy program végrehajtása általában akkor ér véget, ha az összes utasítását végrehajtottuk
- Néha szükség van azonban arra, hogy a programot “abnormálisan” leállítsuk
- Bevezetünk erre egy nyelvi elemet, az abortálást

- Érdemes megjegyezni, hogy az abortálás nem ugyanaz, mint a **skip** vagy a végtelen ciklus, nem tudjuk velük kifejezni

- A C++ például az *exit(int)* és *abort()* függvényekkel implementálja
  - Persze ezek bonyolultabb szemantikával bírnak
  - Kezelik a tárral és memóriával kapcsolatos kérdéseket is
- Java-ban *System.exit(int)*
  - Ez minden végrehajtási szálat terminál
  - És a teljes virtuális gép leállítást levezényli

Szintaktikus kategóriák és metaváltozóik

$$
\begin{array}{llll}
n & \in & \mathrm{Num} & \text{(számliterálok)} \\
x & \in & \mathrm{Var} & \text{(változók)} \\
a & \in & \mathrm{Aexp} & \text{(aritmetikai kifejezések)} \\
b & \in & \mathrm{Bexp} & \text{(logikai kifejezések)} \\
S & \in & \mathrm{Stm} & \text{(utasítások)}
\end{array}
$$

Produkciós szabályok

$$
\begin{array}{lll}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid \text{-}a \\
b & ::= & \textbf{true} \mid \textbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & \textbf{skip} \mid x := a \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \mid \textbf{while } b \textbf{ do } S \mid \\
 & & \textbf{abort}
\end{array}
$$

- Trükk: nem változtatunk semmit a szemantikán, nem írunk fel következtetési szabályt az abortra
- Ennek köszönhetően mindkét operációs szemantika azt fogja mutatni, hogy a végrehajtás ott megakad
- Azzal, hogy nem csinálunk semmit, jó szemantikát kapunk!

Vajon mindig jól működik?

$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto 10] \rangle$

$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto -10] \rangle$

$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \,;\, \textbf{abort}, \quad [x \mapsto 10] \rangle$

$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \,;\, \textbf{abort}, \quad [x \mapsto -10] \rangle$

$\langle \textbf{abort} \,;\, \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto 10] \rangle$

$\langle \textbf{abort} \,;\, \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto -10] \rangle$

Az *abort* utasítást tartalmazó programok mindig terminálnak?

Beragadt vagy zsákutca konfiguráció?

Vajon mindig jól működik?

$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto 10] \rangle$

$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto -10] \rangle$

$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \, ; \, \textbf{abort}, \quad [x \mapsto 10] \rangle$

$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \, ; \, \textbf{abort}, \quad [x \mapsto -10] \rangle$

$\langle \textbf{abort} \, ; \, \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto 10] \rangle$

$\langle \textbf{abort} \, ; \, \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto -10] \rangle$

Az *abort* utasítást tartalmazó programok mindig terminálnak?

Beragadt vagy zsákutca konfiguráció?

Vajon mindig jól működik?

$$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto 10] \rangle$$

$$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto -10] \rangle$$

$$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \ ; \ \textbf{abort}, \quad [x \mapsto 10] \rangle$$

$$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \ ; \ \textbf{abort}, \quad [x \mapsto -10] \rangle$$

$$\langle \textbf{abort} \ ; \ \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto 10] \rangle$$

$$\langle \textbf{abort} \ ; \ \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto -10] \rangle$$

Az *abort* utasítást tartalmazó programok mindig terminálnak?

Beragadt vagy zsákutca konfiguráció?

Vajon mindig jól működik?

$$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto 10] \rangle$$

$$\langle \textbf{if } x < 0 \textbf{ then abort else } x = x - 1, \quad [x \mapsto -10] \rangle$$

$$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \ ; \ \textbf{abort}, \quad [x \mapsto 10] \rangle$$

$$\langle \textbf{if } x < 0 \textbf{ then skip else } x = x - 1 \ ; \ \textbf{abort}, \quad [x \mapsto -10] \rangle$$

$$\langle \textbf{abort} \ ; \ \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto 10] \rangle$$

$$\langle \textbf{abort} \ ; \ \textbf{if } x < 0 \textbf{ then skip else } x = x - 1, \quad [x \mapsto -10] \rangle$$

Az *abort* utasítást tartalmazó programok mindig terminálnak?

Beragadt vagy zsákutca konfiguráció?

- A small-step szemantikában beragadt konfigurációk jelzik a program elakadását, hibás megállását
  - A konfiguráció beragadt, ha nem tudunk átmenetet levezetni hozzá
  - Formálisan, $\langle S, s \rangle$ beragadt, ha nincs olyan c konfiguráció, amelyre $\langle S, s \rangle \Rightarrow c$

- Nem vettünk fel olyan szabályt, amelynek a konklúziója megadná valamely **abort** utasítást tartalmazó konfiguráció átmenetét
- Tehát azok a konfigurációk, amelyek az **abort** utasítás végrehajtását írják elő a következő lépésben, beragadnak
- Továbbá a valahol **abort**ot tartalmazó, és azt végrehajtó programok véges levezetési láncokat eredményeznek, amelyek beragadt konfigurációban érnek véget

- A big-step szemantikában nincsenek beragadó konfigurációk
- A programok vagy sikeresen terminálnak, vagy nem terminálnak, sikertelen terminációt nem értelmezünk
- Az **abort** utasítást tartalmazó programoknak nem számítható ki a szemantikája, “nem futtathatóak”

- Érdekesség: a fentiek következménye, hogy a természetes szemantikában az **abort** és a **while true do skip** utasítások szemantikusan ekvivalensek
- Mivel nem tud különbséget tenni az abnormális terminálás és a divergálás között (hacsak nem vezetünk be egy extremális *hiba* konfigurációt) — v.ö. $\mathcal{S}_{sos}[\![S]\!]$

Nemdeterminisztikusság

- “A nemdeterminisztikus algoritmus egy olyan algoritmus, amely különböző futtatások esetén különböző viselkedést mutathat.”
- Mi most egy egyszerű konstrukcióval foglalkozunk: nemdeterminisztikus választás két utasítás között
- A valószínűségi nyelvek (*probabilistic programming language*) tartalmaznak véletlen értékkiválasztást is

- Az elterjedt programozási nyelvek általában nem implementálnak nemdeterminisztikus választást...
- ...viszont nemdeterminisztikus viselkedést eredményez a konkurencia, mert ekkor az ütemezésen múlik, milyen sorrendben hajtódnak végre az utasítások

- “A nemdeterminisztikus algoritmus egy olyan algoritmus, amely különböző futtatások esetén különböző viselkedést mutathat.”
- Mi most egy egyszerű konstrukcióval foglalkozunk: nemdeterminisztikus választás két utasítás között
- A valószínűségi nyelvek (*probabilistic programming language*) tartalmaznak véletlen értékkiválasztást is

- Az elterjedt programozási nyelvek általában nem implementálnak nemdeterminisztikus választást...
- ...viszont nemdeterminisztikus viselkedést eredményez a konkurencia, mert ekkor az ütemezésen múlik, milyen sorrendben hajtódnak végre az utasítások

Szintaktikus kategóriák és metaváltozóik

$$
\begin{array}{lll}
n & \in & \mathrm{Num} \quad \text{(számliterálok)} \\
x & \in & \mathrm{Var} \quad \text{(változók)} \\
a & \in & \mathrm{Aexp} \quad \text{(aritmetikai kifejezések)} \\
b & \in & \mathrm{Bexp} \quad \text{(logikai kifejezések)} \\
S & \in & \mathrm{Stm} \quad \text{(utasítások)}
\end{array}
$$

Produkciós szabályok

$$
\begin{array}{lll}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid \text{-}a \\
b & ::= & \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & \mathbf{skip} \mid x := a \mid S_1; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S \mid \\
 & & S_1\ \mathbf{or}\ S_2
\end{array}
$$

Nemdeterminisztikusság: példa

Az informális szemantikánk szerint ennek az utasításnak:

$$x := 1 \;\textbf{or}\; \left(x := 3 \;;\; x := x - 1\right)$$

két különböző eredménye lehet: olyan állapotban is véget érhet, amelyben x értéke 1, illetve olyanban is, ahol x értéke 2.

Még érdekesebb a következő utasítás szemantikája:

$$x := 1 \;\textbf{or}\; \left(\textbf{while true do skip}\right)$$

mivel azt várjuk tőle, hogy vagy az $x = 1$ állapotban terminál, vagy pedig egyáltalán nem terminál.

Hogyan tudjuk ezt elérni a formális szemantika definícióban?

Nemdeterminisztikus választás

$$\frac{\langle S_1, s\rangle \to s'}{\langle S_1 \ \mathbf{or}\ S_2, s\rangle \to s'}$$

$$\frac{\langle S_2, s\rangle \to s'}{\langle S_1 \ \mathbf{or}\ S_2, s\rangle \to s'}$$

Ezzel levezethető mindkettő:

$$\langle x := 1 \ \mathbf{or}\ (x := 3\ ;\ x := x - 1), s\rangle \to s[x \mapsto 1]$$

és

$$\langle x := 1 \ \mathbf{or}\ (x := 3\ ;\ x := x - 1), s\rangle \to s[x \mapsto 2]$$

(bármely s állapot esetén).

Mit mondhatunk az $x := 1$ **or** (**while true do skip**) utasítás big-step szemantikájáról?

Az biztosan igaz, hogy

$$\langle x := 1 \textbf{ or } (\textbf{while true do skip}), s\rangle \to s[x \mapsto 1]$$

De a második alternatívával nem tudunk átmenetet levezetni.

Ez azért van, mert a végtelen ciklus esetében nem tudunk levezetési fát építeni és így a nemdet. választás második szabályát nem tudjuk alkalmazni. A big-step szemantika nem fogja a második alternatívát választani: elkerüli a végtelen ciklust és az abnormális terminálást.

Mit mondhatunk az $x := 1$ **or** $\left(\textbf{while true do skip}\right)$ utasítás big-step szemantikájáról?

Az biztosan igaz, hogy

$$\langle x := 1 \textbf{ or } \left(\textbf{while true do skip}\right), s\rangle \to s[x \mapsto 1]$$

De a második alternatívával nem tudunk átmenetet levezetni.

Ez azért van, mert a végtelen ciklus esetében nem tudunk levezetési fát építeni és így a nemdet. választás második szabályát nem tudjuk alkalmazni. A big-step szemantika nem fogja a második alternatívát választani: elkerüli a végtelen ciklust és az abnormális terminálást.

Nemdeterminisztikus választás

$$\frac{}{\langle S_1 \textbf{ or } S_2, s\rangle \Rightarrow \langle S_1, s\rangle}$$

$$\frac{}{\langle S_1 \textbf{ or } S_2, s\rangle \Rightarrow \langle S_2, s\rangle}$$

Ahogy az informális szemantika alapján várjuk, levezethető mindkét eredmény:

$$\langle x := 1 \textbf{ or } (x := 3\ ;\ x := x - 1), s\rangle \Rightarrow^* s[x \mapsto 1]$$

és

$$\langle x := 1 \textbf{ or } (x := 3\ ;\ x := x - 1), s\rangle \Rightarrow^* s[x \mapsto 2]$$

(bármely s állapotra).

Ha az $x := 1$ **or** (**while true do skip**) utasítás small-step szemantikát vizsgáljuk, a következőt kapjuk.

Előáll egy véges levezetési lánc:

$$\langle x := 1 \textbf{ or } (\textbf{while true do skip}), s\rangle \Rightarrow^* s[x \mapsto 1]$$

és egy végtelen levezetési lánc is:

$$\langle x := 1 \textbf{ or } (\textbf{while true do skip}), s\rangle \Rightarrow^* \infty$$

Tehát a small-step szemantika a “rossz” úton is el tudja végezni a végrehajtást. Ez a formális szemantika jobban egybevág az informális szemantikával.

Konkurens programok

- Bevezetünk egy parbegin-parend jellegű nyelvi elemet, amely párhuzamos végrehajtást tesz lehetővé
- Mi két utasítás konkurens végrehajtását fogjuk lehetővé tenni
- Fontos: az általunk megadott egyszerű szemantika összefésüléses, azonban a gyakorlatban ennél bonyolultabb a konkurens végrehajtás szemantikája (lásd pl. gyengén rendezett memóriamodellek)

$$x := 1 \ \textbf{par} \ \big(x := 3 \ ; \ x := x - 1\big)$$

A fenti utasítás ami egyszerű modellünkben x-hez három különböző értéket rendelhet: 0, 1 vagy 2.

A kiterjesztett nyelv szintaxisa

Szintaktikus kategóriák és metaváltozóik

$$\begin{array}{lll} n & \in & \mathrm{Num} \quad \text{(számliterálok)} \\ x & \in & \mathrm{Var} \quad \text{(változók)} \\ a & \in & \mathrm{Aexp} \quad \text{(aritmetikai kifejezések)} \\ b & \in & \mathrm{Bexp} \quad \text{(logikai kifejezések)} \\ S & \in & \mathrm{Stm} \quad \text{(utasítások)} \end{array}$$

Produkciós szabályok

$$\begin{array}{lll} a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid \text{-}a \\ b & ::= & \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \wedge b_2 \\ S & ::= & \mathbf{skip} \mid x := a \mid S_1 ; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S \mid \\ & & S_1\ \mathbf{par}\ S_2 \end{array}$$

$$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 \textbf{ par } S_2, s\rangle \Rightarrow \langle S_1' \textbf{ par } S_2, s'\rangle}$$

$$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 \textbf{ par } S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

$$\frac{\langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle}{\langle S_1 \textbf{ par } S_2, s\rangle \Rightarrow \langle S_1 \textbf{ par } S_2', s'\rangle}$$

$$\frac{\langle S_2, s\rangle \Rightarrow s'}{\langle S_1 \textbf{ par } S_2, s\rangle \Rightarrow \langle S_1, s'\rangle}$$

$$\langle\, x := 1 \;\textbf{par}\; (x := 3 \,;\; x := x - 1), \qquad s \,\rangle \Rightarrow$$
$$\langle\, x := 3 \,;\; x := x - 1, \qquad s[x \mapsto 1] \,\rangle \Rightarrow$$
$$\langle\, x := x - 1, \qquad s[x \mapsto 3] \,\rangle \Rightarrow$$
$$s[x \mapsto 2]$$

$$\langle\, x := 1 \;\textbf{par}\; (x := 3 \,;\; x := x - 1), \qquad s \,\rangle \Rightarrow$$
$$\langle\, x := 1 \;\textbf{par}\; x := x - 1, \qquad s[x \mapsto 3] \,\rangle \Rightarrow$$
$$\langle\, x := x - 1, \qquad s[x \mapsto 1] \,\rangle \Rightarrow$$
$$s[x \mapsto 0]$$

$$\langle\, x := 1 \;\textbf{par}\; (x := 3 \,;\; x := x - 1), \qquad s \,\rangle \Rightarrow$$
$$\langle\, x := 1 \;\textbf{par}\; x := x - 1, \qquad s[x \mapsto 3] \,\rangle \Rightarrow$$
$$\langle\, x := 1, \qquad s[x \mapsto 2] \,\rangle \Rightarrow$$
$$s[x \mapsto 1]$$

Természetes szemantika

- A természetes, big-step szemantikában a részkifejezések mindig teljesen kiértékelésre kerülnek egy átmenettel, így összefésülést nem tudunk leírni
- Következésképp a párhuzamos konstrukciónknak nem tudunk big-step szemantikát definiálni, vagy legalábbis összefésüléssel nem

- Továbbá a big-step szemantikában a nemdeterminisztikus választás elkerüli a végtelen ciklusokat és az abort szemantikusan ekvivalens a végtelen ciklussal
- Ezek a példák jól mutatják, hogy a természetes szemantika jóval absztraktabb, mint a strukturális

Show me the code!

$\mathbb{K}$

A foreach-jellegű ciklus végrehajtható szemantikája elérhető a kurzus anyagai között. A small-step stílusú operációs szemantikát a $\mathbb{K}$ keretrendszerben definiáltuk, kipróbálható a 3.5 és a 3.6 verziókkal.

λ

A switch-case jellegű elágazás végrehajtható szemantikája elérhető a kurzus anyagai között. A leíró szemantikát Haskellben adtuk meg.