# A kivételkezelés szemantikája

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

Folytatásos denotációs szemantika:

A kivételkezelés modellezése

# 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) — átmenetrelációval
    - Strukturális

        Minden lépés modellezése

    - Természetes

        A kezdő- és végállapotok közötti reláció felállítása

- **Denotációs (leíró)** — matematikai objektumokkal

    A jelentést denotációk hozzárendelésével adja meg

# A haladás iránya

- Az előző előadásokon definiáltuk a While nyelv (direkt) denotációs szemantikáját
- Megkülönböztetett figyelemmel a rekurzív szerkezetekre, a fixpont-elméletre

- A direkt szemantika minden utasításhoz hozzárendelt egy függvényt, amely karakterizálja az utasítás jelentését, végrehajtásának hatását
- A magnyelvhez megfelelő, de bizonyos programozási nyelvi elemek jelentésének kifejezésére nem alkalmas, nem elég kifejező

- **Milyen domainnel definiálhatjuk a kivételkezelés és más ugró (nem strukturált) utasítások leíró szemantikáját?**

# A kivételkezeléssel kiegészített While

## Szintaktikus kategóriák és metaváltozóik

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

## Produkciós szabályok

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

# Informális szemantika, motiváció

Miért nem jó az eddigi direkt jelentésleíró modellünk? Hogyan adnánk meg az alábbiak jelentését kompozicionálisan?

- $\left(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2\right) \,;\, S_3$
- $\left(\textbf{try } S_1 \textbf{ catch } e : S_2\right) \,;\, S_3$
- $\left(\textbf{try throw } e \textbf{ catch } e : S_2\right) \,;\, S_3$
- $\left(\textbf{try throw } e \,;\, S_1 \textbf{ catch } e : S_2\right) \,;\, S_3$
- $\left(\textbf{try } S_1 \,;\, \textbf{throw } e \textbf{ catch } e : S_2\right) \,;\, S_3$
- $\left(\textbf{try throw } e_1 \,;\, S_1 \textbf{ catch } e_2 : S_2\right) \,;\, S_3$
- $\left(\textbf{try } \left(\textbf{if } b \textbf{ then } S_1 \textbf{ else throw } e\right) \textbf{ catch } e : S_2\right) \,;\, S_3$

(Az $S_n$ elemek utasítások a *magnyelvben*)

(Vezérlésfolyam)

**try**
  **while true do**
    **if** $x > 0$ **then**
      **throw** *exit*
    **else**
      $x := x + 1$
**catch** *exit* : $y := 1$

Tehát egy végtelen ciklus (*while true do*… ) is lehet véges (ezt kell formalizálni a modellben is).

- A *try...catch* szerkezet kivételkezelő blokkot vezet be
- Az el nem kapott kivételek továbbterjednek külsőbb blokkok felé

- A *throw* utasítás használható kivétel kiváltására
- Megszakítja a blokk futását, a hátralévő utasítások nem kerülnek végrehajtásra
- A vezérlés átadódik egy megfelelő kivételkezelő kódra
- A kivételkezelő blokk végrehajtása után “normálisan” folytatódik a kiértékelés, a kivételkezelt blokkot követő programrészlettel

# A folytatásos szemantika alapötlete

- A kivételek (és általában az ugró utasítások) hatásának leírásához más denotációs szemantikus formalizmusra lesz szükségünk
- Amikor kiváltódik egy kivétel, tudnunk kell, hogyan folytatódik a programvégrehajtás a kivételkezelő blokk után (hogyan folytatódna a kivételkezelt blokk után)
- Ehhez folytatásos stílusban (continuation-passing style, CPS) adjuk meg a szemantikus függvényt

$$S''_{cs} : \mathrm{Stm} \rightarrow \mathrm{State} \rightarrow (\mathrm{State} \hookrightarrow \mathrm{State}) \hookrightarrow \mathrm{State}$$

Ami nekünk most fontos: a CPS alkalmas a vezérlésfolyam tiszta függvényekkel való modellezésére.

# A folytatásos szemantika alapötlete

- Először a magnyelvhez definiálunk folytatásos leíró szemantikát:

$$\mathcal{S}''_{cs} : \text{Stm} \rightarrow \text{State} \rightarrow (\text{State} \hookrightarrow \text{State}) \hookrightarrow \text{State}$$

- De a folytatást használjuk első argumentumként (máskor általában az utolsó szokott lenni):

$$\mathcal{S}'_{cs} : \text{Stm} \rightarrow (\text{State} \hookrightarrow \text{State}) \rightarrow \text{State} \hookrightarrow \text{State}$$

- Használjunk rövidítást a folytatásra:

$$\text{Cont} = \text{State} \hookrightarrow \text{State}$$

- Tehát a magnyelv folytatásos szemantikus függvény típusa:

$$\mathcal{S}'_{cs} : \text{Stm} \rightarrow (\text{Cont} \rightarrow \text{Cont})$$

# A While folytatásos szemantikája

$$S'_{cs} : \mathrm{Stm} \to (\mathrm{Cont} \to \mathrm{Cont})$$

## Folytatásos szemantika

$$\begin{aligned}
\mathcal{S}'_{cs}[\![\textbf{skip}]\!] &= id_{\mathrm{Cont}} \\
\mathcal{S}'_{cs}[\![x := a]\!]c\ s &= c(s[x \mapsto \mathcal{A}[\![a]\!]s]) \\
\mathcal{S}'_{cs}[\![S_1; S_2]\!] &= \mathcal{S}'_{cs}[\![S_1]\!] \circ \mathcal{S}'_{cs}[\![S_2]\!] \\
\mathcal{S}'_{cs}[\![\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2]\!]c &= cond(\mathcal{B}[\![b]\!], \mathcal{S}'_{cs}[\![S_1]\!]c, \mathcal{S}'_{cs}[\![S_2]\!]c) \\
\mathcal{S}'_{cs}[\![\textbf{while}\ b\ \textbf{do}\ S]\!] &= \mathrm{FIX}\ G \\
\text{ahol } (G\ g)\ c &= cond(\mathcal{B}[\![b]\!], \mathcal{S}'_{cs}[\![S]\!](g\ c), c)
\end{aligned}$$

# A While folytatásos szemantikája

Szignatúra:
$\mathrm{Stm} \to (\mathrm{Cont} \to (\mathrm{State} \hookrightarrow \mathrm{State}))$ vs $\mathrm{Stm} \to (\mathrm{State} \hookrightarrow \mathrm{State})$

Skip:
$id_{\mathrm{Cont}}$ vs $id_{\mathrm{State}}$

Szekvencia:
$\mathcal{S}'_{cs}[\![S_1]\!] \circ \mathcal{S}'_{cs}[\![S_2]\!]$ vs $\mathcal{S}_{ds}[\![S_2]\!] \circ \mathcal{S}_{ds}[\![S_1]\!]$

Ciklus:
$G\ g\ c = cond(\mathcal{B}[\![b]\!], \mathcal{S}'_{cs}[\![S]\!](g\ c), c)$ vs $F\ g = cond(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{ds}[\![S]\!], id_{\mathrm{State}})$

# A denotációs szemantikák kapcsolata

- A direkt szemantikában azt adjuk meg, mi az adott konstrukció végrehajtásának hatása
- A folytatásos szemantikában “visszafelé” haladva állítjuk elő a konstrukciók jelentését: a folytatás függvényében adjuk meg, mi az utasítás jelentése, ha a megadott folytatás követi a végrehajtásban

- Mégis, a két megközelítés között tiszta összefüggés írható fel
- Bizonyítható, hogy az $\mathcal{S}'_{cs}$ valóban az $\mathcal{S}_{ds}$ folytatásos kifejezése:

$$\mathcal{S}'_{cs}[\![S]\!]c = c \circ \mathcal{S}_{ds}[\![S]\!]$$

- Következésképp

$$\mathcal{S}'_{cs}[\![S]\!]\ id_{\mathrm{State}} = \mathcal{S}_{ds}[\![S]\!]$$

## Folytatásos szemantika

$$\mathcal{S}'_{cs}[\![\textbf{try } S_1 \textbf{ catch } e : S_2]\!] = ?$$
$$\mathcal{S}'_{cs}[\![\textbf{throw } e]\!] = ?$$

- A kivételkezelt blokk tetszőlegesen komplex lehet
- A *throw* utasítás működése/hatása függ attól, hogy a környezetében (tetszőlegesen távol) milyen kivételkezelő blokkokat definiáltunk
- A külső környezetből minket most a definiált kivételkezelők (*catch* szekciók) érdekelnek

**A ’throw’ viselkedése csak a környezet jelentésének ismeretében adható meg: ahány kivételkezelő blokk, annyi lehetséges folytatás.**

- A kivételek “jelentését” kivételkörnyezetben tartjuk nyilván:

$$\mathrm{Env}_E = \mathrm{Exception} \to \mathrm{Cont}$$

- Definiálja, mi történik a kivétel kiváltása esetén (mi a hatása a hátralévő programrésznek)
- Ez természetesen függ a környezettől, a kivételkezelők definiálják ezt a környezetet

A kivételkörnyezettől függ a jelentés, így paraméterezzük vele a leíró szemantikus függvényt:

$$S_{cs} : \mathrm{Stm} \to \mathrm{Env}_E \to (\mathrm{Cont} \to \mathrm{Cont})$$

$$S_{cs} : \mathrm{Stm} \to \mathrm{Env}_E \to (\mathrm{Cont} \to \mathrm{Cont})$$

## A folytatásos szemantikai szabályai

$$
\begin{aligned}
\mathcal{S}_{cs}[\![\textbf{skip}]\!]\ env_E\ &= id_{\mathrm{Cont}}\\
\mathcal{S}_{cs}[\![x := a]\!]\ env_E\ c\ s &= c(s[x \mapsto \mathcal{A}[\![a]\!]s])\\
\mathcal{S}_{cs}[\![S_1; S_2]\!]\ env_E\ &= (\mathcal{S}_{cs}[\![S_1]\!]\ env_E\ ) \circ (\mathcal{S}_{cs}[\![S_2]\!]\ env_E\ )\\
\mathcal{S}_{cs}[\![\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2]\!]\ env_E\ c &= cond(\mathcal{B}[\![b]\!], \mathcal{S}_{cs}[\![S_1]\!]\ env_E\ c, \mathcal{S}_{cs}[\![S_2]\!]\ env_E\ c)\\
\mathcal{S}_{cs}[\![\textbf{while}\ b\ \textbf{do}\ S]\!]\ env_E\ &= \mathrm{FIX}\ G\\
\text{ahol } (G\ g)\ c &= cond(\mathcal{B}[\![b]\!], \mathcal{S}_{cs}[\![S]\!]\ env_E\ (g\ c), c)\\
\mathcal{S}_{cs}[\![\textbf{try}\ S_1\ \textbf{catch}\ e : S_2]\!]\ env_E\ c &= \mathcal{S}_{cs}[\![S_1]\!](\ env_E\ [e \mapsto \mathcal{S}_{cs}[\![S_2]\!]\ env_E\ c])\ c\\
\mathcal{S}_{cs}[\![\textbf{throw}\ e]\!]\ env_E\ c &= \ env_E\ e
\end{aligned}
$$

Tegyük fel, hogy van egy kezdeti $env_E$ kivételkörnyezetünk.

$$\begin{aligned}
&\mathcal{S}_{cs}[\![\textbf{try while true do}\\
&\qquad\qquad \textbf{if } x > 0 \textbf{ then throw } exit \textbf{ else } x := x+1\\
&\qquad \textbf{catch } exit : y := 1]\!]\ env_E\ id_{\mathrm{State}} =\\
&\mathcal{S}_{cs}[\![\textbf{while true do } \ldots]\!]\ env_E[exit \mapsto c_{exit}]\ id_{\mathrm{State}} = (FIX\ G)\ id_{\mathrm{State}}
\end{aligned}$$

$$\begin{aligned}
G\ g\ c\ s = {}& cond(\mathcal{B}[\![\textbf{true}]\!]\\
&\qquad cond(\mathcal{B}[\![x > 0]\!], c_{exit}, \mathcal{S}_{cs}[\![x := x+1]\!]\ env_E[exit \mapsto c_{exit}](g\ c)),\\
&\qquad c)\ s =\\
={}& cond(\mathcal{B}[\![x > 0]\!], c_{exit}, \mathcal{S}_{cs}[\![x := x+1]\!]\ env_E[exit \mapsto c_{exit}](g\ c))\ s =\\
={}& \begin{cases} c_{exit}\ s & \text{if} \quad s[x] > 0\\ (g\ c)(s[x \mapsto s[x]+1]) & \text{if} \quad s[x] \leq 0 \end{cases}
\end{aligned}$$

$$c_{exit}\ s = id_{\mathrm{State}} s[y \mapsto 1] = s[y \mapsto 1]$$

$$(FIX\ G)\ id_{\mathrm{State}}\ s = \begin{cases} s[y \mapsto 1] & \text{if} \quad s[x] > 0\\ s[x \mapsto 1][y \mapsto 1] & \text{if} \quad s[x] \leq 0 \end{cases}$$

- Az kivételkezelés operációs szemantikáját csak vázlatosan adjuk meg
- Hogyan lehetne big-step szemantikát definiálni a kivételkezeléshez?

**Throw**

$$\frac{?}{\langle \textbf{throw}\ e, s\rangle \rightarrow ?}\ ?$$

**Try-catch**

$$\frac{?}{\langle \textbf{try}\ S_1\ \textbf{catch}\ e : S_2, s\rangle \rightarrow ?}\ ?$$

- Kiegészítve a lehetséges konfigurációkat kivétel-állapot párosokkal:

**Throw**

$$\frac{}{\langle \textbf{throw}\ e, s\rangle \rightarrow \langle e, s\rangle}$$

**Try-catch**

$$\frac{\langle S_1, s\rangle \rightarrow s'}{\langle \textbf{try}\ S_1\ \textbf{catch}\ e : S_2, s\rangle \rightarrow s'}$$

$$\frac{\langle S_1, s\rangle \rightarrow \langle e, s'\rangle \quad \langle S_2, s'\rangle \rightarrow c}{\langle \textbf{try}\ S_1\ \textbf{catch}\ e : S_2, s\rangle \rightarrow c}$$

**Szekvencia szabály (módosított)**

$$\frac{\langle S_1, s\rangle \to s' \qquad \langle S_2, s'\rangle \to c}{\langle S_1; S_2, s\rangle \to c}$$

**Szekvencia szabály (hozzáadott)**

$$\frac{\langle S_1, s\rangle \to \langle e, s'\rangle}{\langle S_1; S_2, s\rangle \to \langle e, s'\rangle}$$

*A szemantikadefiníció befejezéséhez az összes utasítás esetében delegálni kell a kivételeket (az elágazás és ciklus szabályát is igazítani kell).*

$\lambda$

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