Zk 1

forEach(aL, f) = λi. loop(aL, f, i) // aL sekvence všech prvků A ∈ A*
i = stále volnou cifru

loop : (A* × (A × State → State) × State) → State

loop(ε, f, s) = s // po poslední iteraci vrátí výsledný stav

loop({a, aL}, f, s) = loop(aL, f, f(a)s) // po provedení akce a
po neposlední iteraci aplikuje funkci na další akci a,
výsledná funkce (State → State) aplikuje na výsledný stav a výsledek znovu pošle do loopu se zbytkem akcí

objekty typu z → z

Teorie programovacích jazyků, zkouška 21. 12. 2012

1. Nadefinujte funkci *forEach* tak, aby vrátila akci, která postupně projde všechny prvky sekvence zadané v prvním parametru funkce *forEach* a na každém z nich vykoná akci zadanou v druhém parametru funkce *forEach*.

$$forEach : A^* \times (A \times State \to State) \to (State \to State)$$

2. Dokažte následující tvrzení: pokud $(\vdash t : T) \wedge (t \Rightarrow t')$, pak $\vdash t' : T$.

$$t ::= \text{true} \mid \text{false} \mid 0 \mid \text{succ } t \mid \text{if } t \text{ then } t \text{ else } t \quad (1)$$

$$\frac{}{\text{if true then } t_1 \text{ else } t_2 \Rightarrow t_1} \quad (2)$$

$$\frac{}{\text{if false then } t_1 \text{ else } t_2 \Rightarrow t_2} \quad (3)$$

$$\frac{t_1 \Rightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \Rightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3} \quad (4)$$

$$\frac{t_1 \Rightarrow t_1'}{\text{succ } t_1 \Rightarrow \text{succ } t_1'} \quad (5)$$

$$\frac{}{\vdash \text{true} : Bool} \quad (6)$$

$$\frac{}{\vdash \text{false} : Bool} \quad (7)$$

$$\frac{}{\vdash 0 : Nat} \quad (8)$$

$$\frac{\vdash t : Nat}{\vdash \text{succ } t : Nat} \quad (9)$$

$$\frac{\vdash t_1 : Bool \qquad \vdash t_2 : T \qquad \vdash t_3 : T}{\vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (10)$$

3. Dokažte, že přepisovací relace $\Rightarrow$ definovaná v předcházejícím úkolu příkladů je silně normalizující (vždy terminuje).

4. Sémantika S má vstupní funkci $\lambda z, e.(z, e)$, výstupní funkci $\lambda(z, z').z'$, množinu konfigurací $Z \times Expr$, množinu finálních konfigurací $Z \times Z$ a přepisovací relaci $\Rightarrow$. Napište ekvivalentní denotační sémantiku, jako sémantickou doménu použijte $Z \to Z$.

$$Expr ::= Num \mid Expr + Expr \mid Expr - Expr \mid \ldots \quad (11)$$

Teorie programovacích jazyků, zkouška 21. 12. 2012

z = hodota závorek

$$\frac{}{(z, \Delta z') \Rightarrow (z, -z')} \quad (12)$$

$$\frac{}{(z, z' \oplus z'') \Rightarrow (z, z' + z'')} \quad (13)$$

$$\frac{(z, e) \Rightarrow (z, e')}{(z, \Delta e) \Rightarrow (z, \Delta e')} \quad (14)$$

$$\frac{(z, e) \Rightarrow (z, e')}{(z, e \otimes e'') \Rightarrow (z, e' \otimes e'')} \quad (15)$$

$$\frac{(z, e'') \Rightarrow (z, e')}{(z, e \otimes e'') \Rightarrow (z, e \otimes e')} \quad (16)$$

$$\frac{}{(z, \square) \Rightarrow (z, z)} \quad (17)$$

$[\![z]\!] = \lambda z . z$

$[\![\square]\!] = \lambda z . z$

$[\![\Delta e]\!] = [\![\Delta]\!][\![e]\!]$

$[\![e \otimes e']\!] = [\![\otimes]\!]([\![e]\!][\![e']\!])$

$[\![\Delta]\!] = \lambda e . \lambda z . -e(z)$

$[\![\otimes]\!] = \lambda e, e' . \lambda z . e(z) + e'(z)$

operace typu $(z \to z) \to (z \to z)$

5. Pravidlo 18 je pravidlo pro podtypování rekurzivních typů. Na příkladu vysvětlete, proč by pravidlo 19 nefungovalo. Svoje zdůvodnění opřete o vlastnosti programovacích jazyků jako je soundness, progress apod.

$$\frac{\Gamma \vdash \mu X.A \quad \Gamma \vdash \mu Y.B \quad \Gamma \cup \{Y <: Top, X <: Y\} \vdash A <: B}{\Gamma \vdash \mu X.A <: \mu Y.B} \quad (18)$$

$$\frac{\Gamma \vdash \mu X.A \quad \Gamma \vdash \mu Y.B \quad \Gamma \cup \{Y <: Top, X <: Top\} \vdash A <: B}{\Gamma \vdash \mu X.A <: \mu Y.B} \quad (19)$$

příklad nedůležitosti
Lacha <: Živce

class ŽivecItem {
  ŽivecItem next;
  Živce živce;
}

class LachaItem extends ŽivecItem {
  LachaItem next;
  Lacha živce
}

⇒ po aplikaci (19) by už ale nešlo „extends" a tudíž by nešlo říci, že LachaItem <: ŽivecItem

$\forall e \in Expr : \exists n \in \mathbb{N} \wedge$

$\forall e, e' \in Expr : e \rightsquigarrow e' \Rightarrow$

$\vdash \mu X.X <: \mu Y.Y \ldots \to$

→ dokazuje se $\forall$:

$\{Y <: Top, X <: Top\} \vdash X <: Y$, což už ale neplatí

Zk 2

Teorie programovacích jazyků, zkouška 8. 1. 2013

1. Napište výraz lambda kalkulu implementující funkci f. Pro přehlednost používejte symbolická jména standardních kombinátorů (např. 0) a jejich definice rozepište vedle. Nápověda: nejprve zkonstruujte generátor funkce f.

$$f(1, x, y) = x(1, y(1)) \quad (1a)$$
$$f(n, x, y) = x(n, f(n-1, x, y)) \quad \text{pro } n > 1 \quad (1b)$$

2. V jazyce Featherweight Java formálně odvoďte, že `new X().g(new X().f())` $\longrightarrow^*$ `new Object()`. Při každém redukčním kroku napište nad šipku název použitého redukčního pravidla.

```
class X extends Object {
  X() { super(); }
  X f() { return this.f(); }
  Object g(X x) { return new Object(); }
}
```

3. V jazyce Featherweight Java formálně odvoďte, že třída `X` z předchozího příkladu je správně otypovaná. Vedle každé "zlomkové" čáry napište název použitého odvozovacího pravidla.

4. Relace $\longrightarrow$ ve Featherweight Javě definuje sémantiku malého kroku. Předpokládejte, že relace $\Longrightarrow$ definuje ekvivalentní sémantiku velkého kroku a dopište následující pravidlo:

$$\frac{}{e_0.m(e_1, \ldots, e_n) \Longrightarrow} \quad (2)$$

5. Vyhodnocování výrazů ve Featherweight Javě se může za určitých okolností zaseknout. Rozšiřte FJ o hodnotu `error` tak, aby k zaseknutí nedocházelo. Neměňte existující pravidla typového systému a sémantiky, pouze přidávejte nová.

① $f = (\lambda f.\lambda n.\lambda x.\lambda y.\ \text{if}\ (\text{iszero}\ (\text{Dec}^n\ n))\ (x\ 1\ (\ldots\ 0))\ (x\ n\ (f\ (\text{Dec}\ n)\ x\ y))$

+ zápis základních kombinátorů: True: $\lambda x.\lambda y.\ x$
False: $\lambda x.\lambda y.\ y$
and, or, not, if..

② `new X().g(new X(),f())` $\xrightarrow{\text{pravidlo R-INVK}}$ $[\text{new X().f()}/x\ ,\ \text{new X()}/\text{this}]$ $\xrightarrow{\text{substitice}}$ new Object()
→ nutno ještě těžkat toto pravidlo pomocí „method body lookup"
A. IGARASHI, page 12

③ napíše, že je správně fundělání i nedělá; page 9
class → class typing

④ $e_0 \Rightarrow \text{new } C_0(\bar{e}_0) = V_0 \qquad e_1 \ldots e_n \Rightarrow V_1 \ldots V_n$
$\text{mbody}(m, C_0) = \bar{x}.\ e_m \qquad [V_1 \ldots V_n/\bar{x},\ V_0/\text{this}]\, e_m \Rightarrow V_r$
$e_0.m(e_1, \ldots e_n) \Rightarrow V_r$
(bude mě — nepošlu by nepotřebě vyčíslení se argumenty)

⑤ FJ se zaseknou jen při dynamickém přetypování → přidáme pravidlo:

(R-CAST): $\dfrac{C\ !<:\ D}{(D)(\text{new } C(e)) \longrightarrow \text{error}}$

+ pravidla pro kongruence → nanpř.: RC-FIELD-ERROR $\dfrac{e_0 \longrightarrow \text{error}}{e_0.f \longrightarrow \text{error}}$

1. Napište co nejobecnější typ funkce f. Nápověda: typ proměnné a se dá odvodit z kontextu, přesně typy proměnných b, c a d neznáte, proto použijte parametrický polymorfismus. Měli byste nějak reflektovat to, že c je použito jako parametr funkce b a že výsledek volání funkce f je buď $b(c)$ nebo d.

$$f(a, b, c, d) = \text{if } a \text{ then } b(c) \text{ else } d \tag{1}$$

2. Rozšiřte definici funkce m (implementující rozpoznávání regulárních výrazů) o případ $m(r \times n, \langle a_1, \ldots a_n \rangle, k)$ tak, aby výraz $r \times n$ byl, neformálně řečeno, syntaktická zkratka za

$$\underbrace{r \cdot r \cdot \ldots \cdot r}_{n}, \quad \text{kde } n \geq 0. \tag{2}$$

$$\begin{aligned} RegExp ::= {} & \epsilon \\ & A \\ & RegExp \cdot RegExp \\ & RegExp + RegExp \\ & RegExp \times N \end{aligned} \tag{3}$$

$$A^\perp = A^* \cup \{\perp\} \tag{4}$$

$$m : RegExp \times A^\perp \times (A^\perp \to A^\perp) \to A^\perp \tag{5}$$

$$m(r, \_, k) = k(\perp) \tag{6}$$

$$m(\epsilon, \langle\rangle, k) = k(\langle\rangle) \tag{7}$$

$$m(\epsilon, \langle a_1, \ldots a_n \rangle, k) = k(\langle a_1, \ldots, a_n \rangle) \tag{8}$$

$$m(a, \langle\rangle, k) = k(\perp) \tag{9}$$

$$m(a, \langle a_1, a_2, \ldots a_n \rangle, k) = k(\langle a_2, \ldots a_n \rangle) \quad \text{pokud } a = a_1 \tag{10}$$

$$m(a, \langle a_1, \ldots a_n \rangle, k) = k(\perp) \quad \text{pokud } a \neq a_1 \tag{11}$$

$$m(r_1 \cdot r_2, \langle a_1, \ldots a_n \rangle, k) = m(r_1, \langle a_1, \ldots a_n \rangle, \lambda x.m(r_2, x, k)) \tag{12}$$

$$\begin{aligned} m(r_1 + r_2, \langle a_1, \ldots, a_n \rangle, k) = m(r_1, \langle a_1, \ldots, a_n \rangle, \\ \lambda x.\text{if } k(x) = \langle\rangle \text{ then } \langle\rangle \text{ else } m(r_2, \langle a_1, \ldots a_n \rangle, k)) \end{aligned} \tag{13}$$

$$match : RegExp \times A^* \to Boolean \tag{14}$$

$$match(r, \langle a_1, \ldots, a_n \rangle) = true \quad \text{pokud } m(r, \langle a_1, \ldots a_n \rangle, \lambda x.x) = \langle\rangle \tag{15}$$

$$match(r, \langle a_1, \ldots, a_n \rangle) = false \quad \text{jinak} \tag{16}$$

3. Nadefinujte funkci *doWhile* tak, aby vrátila akci, která reprezentuje do ... while cyklus. Pozor: v tomto cyklu se podmínka testuje až po provedení těla cyklu, každý do ... while cyklus proto proběhne alespoň jednou.

$$doWhile : (State \to State) \times (State \to Bool) \to (State \to State) \tag{17}$$

$$doWhile(body, condition) = \lambda s.\ \text{if } (cond(s)) \text{ then } (doWhile(body, cond)(body(s))) \text{ else } (body(s))$$

4. Průnik typů A a B, značený $A \cap B$, je typ, jehož instance mají zároveň typ A i B. Formálně:

$$\times \quad \frac{e : A \quad e : B}{e : A \cap B}. \quad \checkmark \tag{18}$$

Rozhodněte, zda platí $A \cap B <: A$ nebo $A <: A \cap B$ (případně oboje). Svoje tvrzení dobře zdůvodněte. Nápověda: zamyslete se nad subsumpcí.

5. Napište definici konfluence a dokažte konfluenci relace $\Rightarrow$ definované níže.

$$\begin{aligned} Expr ::= {} & Num \mid \\ & \ominus Expr \mid \\ & Expr \oplus Expr \end{aligned} \tag{19}$$

$$\frac{}{\ominus n \Rightarrow -n} \tag{20}$$

$$\frac{}{n \oplus n' \Rightarrow n + n'} \tag{21}$$

$$\frac{e \Rightarrow e'}{\ominus e \Rightarrow \ominus e'} \tag{22}$$

$$\frac{e_1 \Rightarrow e_1'}{e_1 \oplus e_2 \Rightarrow e_1' \oplus e_2} \tag{23}$$

$$\frac{e_2 \Rightarrow e_2'}{e_1 \oplus e_2 \Rightarrow e_1 \oplus e_2'} \tag{24}$$

Zk 4

or = λleft, right. λs. if (λb,s. b(left(s))) then left(s)
else right(λb,s. b(left(s)))

right je zhotovení nad stavem z left

innerNodes = λtree. if (isNull(unfold(tree))) then 0
else if (isLeaf(unfold(tree))) then 0
else 1 + innerNodes(first(asNode(unfold(tree)))) +
+ — " — second — " —

isLeaf = λtree. isNull(first(asNode(unfold(tree))))
and — " — second — " —

- isNull = isLeft
- asNode = asRight

1. Napište definici funkce *or*. Vrácená akce by měla svůj výsledek vyhodnocovat líně, tzn. vyhodnocovat druhý operand pouze pokud je to nutné. Pozor: druhý operand byste neměli vyhodnocovat nad počátečním stavem.

$$or : (State \to Bool \times State) \times (State \to Bool \times State) \to (State \to Bool \times State) \quad (1)$$

2. Napište tělo funkce $innerNodes : (\mu A. \circ + (A \times A)) \to Number$, která vrátí počet vnitřních uzlů (tzn. ne listů) zadaného stromu. Nápověda: pravděpodobně budete potřebovat některé z operátorů *unfold*, *inLeft*, *inRight*, *asLeft*, *asRight*, *first* a *second*.

3. Upravte relaci $\Rightarrow$ tak, aby sémantika zůstala stejná, ale vyhodnocování výrazů probíhalo striktně zleva doprava.

$$\begin{array}{rl} Expr ::= & Num \mid \\ & \triangle Expr \mid \\ & Expr \oplus Expr \end{array} \quad (2)$$

Konvence: $e, e', e_1, e_2 \in Expr$ a $n, n' \in Num$.

$$\frac{}{\triangle n \Rightarrow -n} \quad (3)$$

$$\frac{}{n \oplus n' \Rightarrow n + n'} \quad (4)$$

$$\frac{e \Rightarrow e'}{\triangle e \Rightarrow \triangle e'} \quad (5)$$

$$\frac{e_1 \Rightarrow e'}{e_1 \oplus e_2 \Rightarrow e' \oplus e_2} \quad (6)$$

$$\frac{e_2 \Rightarrow e'}{e_1 \oplus e_2 \Rightarrow e_1 \oplus e'} \quad (7)$$

přepis na:

$$\frac{e_2 \Rightarrow e'}{n \oplus e_2 \Rightarrow n \oplus e'}$$

4. Dokažte nebo vyvraťte (dokažte negaci):

$$\begin{array}{l} \forall e \in Expr, t \in \{Num, Bool\} : (e : t) \implies \\ (e \in (Bool \cup Num) \vee \exists e' \in Expr : e \to e'). \end{array} \quad (8)$$

$$\begin{array}{rl} Expr ::= & Num \mid \\ & Bool \mid \\ & \triangle Expr \mid \\ & Expr \oplus Expr \mid \\ & \texttt{if } Expr \texttt{ then } Expr \texttt{ else } Expr \end{array} \quad (9)$$

=> stačí projít všechna pravidla, zda se přepisují a vracet, že jeden příklad s výsledkem je větší než jeden za středem. A že ten vzniklý jeden je buďto observ nebo nevzniklí konec

Konvence: $e, e', e'', \ldots \in Expr$, $b, b' \in Bool$, $n, n' \in Num$ a $t \in \{Num, Bool\}$.

$$\frac{}{\triangle n \to -n} \quad (10)$$

$$\frac{e \to e'}{\triangle e \to \triangle e'} \quad (11)$$

$$\frac{}{n \oplus n' \to n + n'} \quad (12)$$

$$\frac{e' \to e''}{e \oplus e' \to e \oplus e''} \quad (13)$$

$$\frac{e \to e''}{e \oplus e' \to e'' \oplus e'} \quad (14)$$

$$\frac{}{\texttt{if } true \texttt{ then } e \texttt{ else } e' \mapsto e} \quad (15)$$

$$\frac{}{\texttt{if } false \texttt{ then } e \texttt{ else } e' \mapsto e'} \quad (16)$$

$$\frac{e \to e'''}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' \to \texttt{if } e''' \texttt{ then } e' \texttt{ else } e''} \quad (17)$$

$$\frac{e' \to e'''}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' \to \texttt{if } e \texttt{ then } e''' \texttt{ else } e''} \quad (18)$$

$$\frac{e'' \to e'''}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' \to \texttt{if } e \texttt{ then } e' \texttt{ else } e'''} \quad (19)$$

$$\frac{}{n : Num} \quad (20)$$

$$\frac{}{b : Bool} \quad (21)$$

$$\frac{e : Num}{\triangle e : Num} \quad (22)$$

$$\frac{e : Num \quad e' : Num}{e \oplus e' : Num} \quad (23)$$

$$\frac{e : Bool \quad e' : t \quad e'' : t}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' : t} \quad (24)$$

5. Napište denotační sémantiku jazyka s níže uvedenou syntaxí (operátor $\ominus$ mu intuitivně reprezentuje mínus, operátor $\oplus$ plus). Jako sémantickou doménu použijte funkci, která k mapování jmen na hodnoty vrátí číslo ($(VarName \to Num) \to Num$). Nápověda: chcete po vás napsat funkci $Expr \to ((VarName \to Num) \to Num)$.

sémantická doména

(VarName → Num)

m (= expr)

navrátí typ

$$
\begin{aligned}
Expr ::= & \; Num \mid \\
& \; VarName \mid \\
& \; \ominus Expr \mid \\
& \; Expr \oplus Expr
\end{aligned}
\tag{25}
$$

$[\![n]\!] = \lambda m . n$

$[\![v]\!] = \lambda m . m(v)$

$[\![e + f]\!] = \lambda m . [\![+]\!]([\![e]\!]\,[\![f]\!])$

$[\![-e]\!] = \lambda m . [\![-]\!]([\![e]\!])$

obecný

[illegible]

$[\![-]\!] = \lambda e . \lambda m . -e(m)$

$[\![+]\!] = \lambda e . \lambda f . \lambda m . e(m) + f(m)$

$\lambda e_1 f$

ZLS

1) $F(w(x,y,z)) = \text{if } w(x) = y(z) \text{ then } w(x) \text{ else } F(w, w(z), y, y(x))$

~~$w(x)$ je objekt typu~~

$w(x)$ musí být jako $y(z)$

$y(x) \to z$

$w(z) \to x$

$w: A \to B$

$x: A$

$y: C \to D$ } then $\Rightarrow$ stejný typ

$z: C$

x je typu jako z

$w(z)$ musí být jako x

$y(x)$ musí být jako z

jeden typ A

2) $f: \forall A. (A) \to A$

$\forall A. (A \to A) \times A \times (A \to A) \times A \longrightarrow A$

3) $runInTransaction(a_L, s) = \lambda i.$ kolikátý (s_i) $\overset{RIT}{run}(a_L)(s)$

(počáteční stav, poč. stav)

$action(s_p, a_{ip}, s_p)$

FAIL $(\langle a_1 \dots a_n \rangle, s)$:

FAIL $(\langle\rangle, s) =$ ~~TRUE~~ FALSE

FAIL $(\langle a_1, a_n \rangle, s) =$ ~~FALSE~~ TRUE,

pokud $a_1(s) = \bot$

FAIL $(\langle a_1 \dots a_n \rangle, s) =$ FAIL $(\langle a_2, a_n \rangle, A_1(s))$

pokud $a_1(s) \neq \bot$

OK: vezmeme si exekuční pořadí (např: 0 akcí)

- $RIT(\langle\rangle) = \lambda s. s$

↳ tj. dokončený operace (úspěšný)

- jinak, akorát se stav do stavu

první ujetý

- $RIT(\langle a_1 \dots a_n \rangle) = \lambda s. s$

Pokud $A_1(s) = \bot$

$A_1(s) = \bot$

nebo

$\neq \bot$

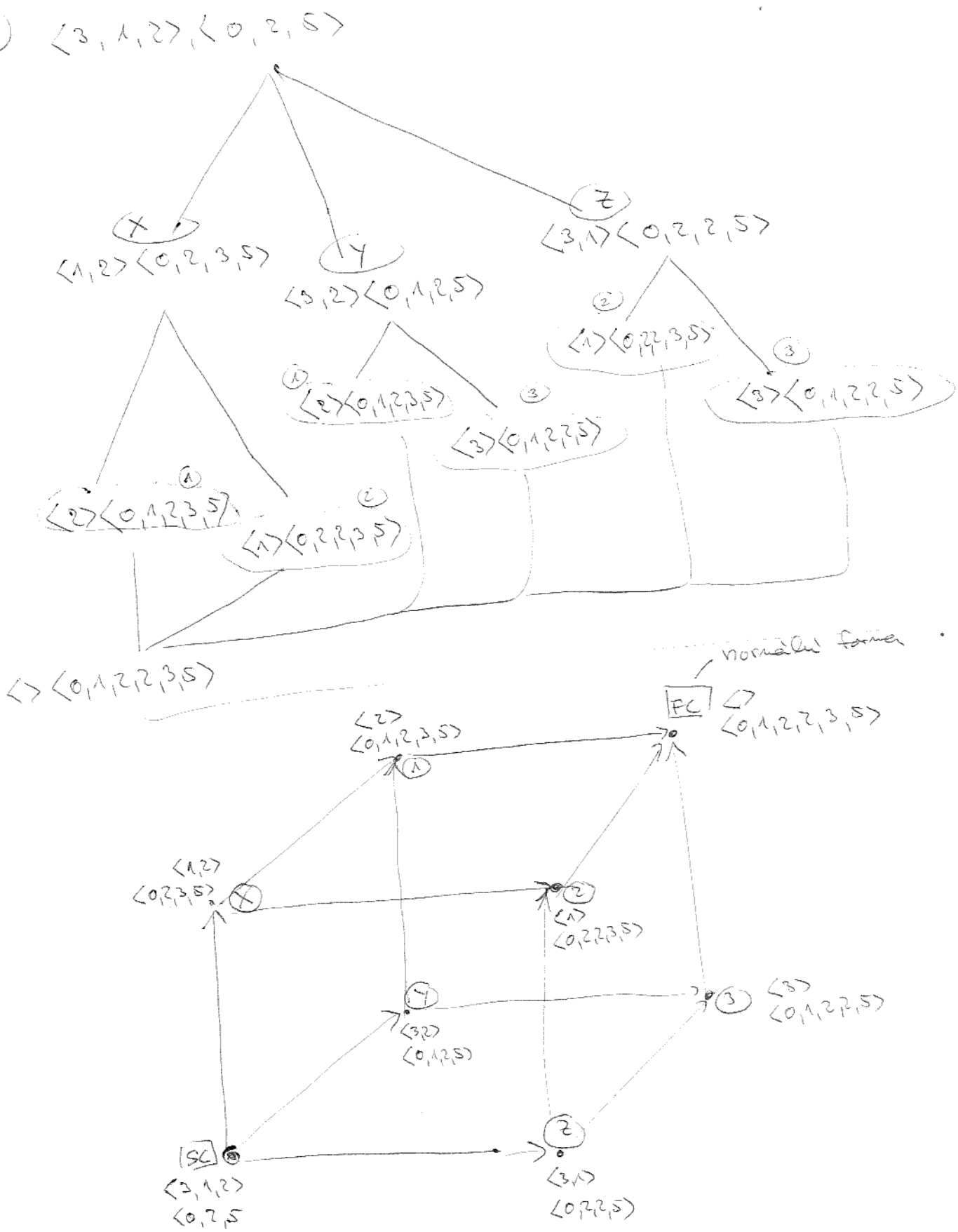
VÝSLEDEK:

- $RIT(\langle a_1 \dots a_n \rangle) = \lambda s. s$ Pokud FAIL $(\langle a_1 \dots a_n \rangle, s)$

Jinak $RIT(\langle a_1 \dots a_n \rangle) = \lambda s. RIT(\langle a_2, a_n \rangle)(a_1(s))$

+ FAIL definice

4) <3,1,2>,<0,2,5>


x
<1,2> <0,2,3,5>
y
<3,2> <0,1,2,5>
z
<3,1> <0,2,2,5>
<> <0,1,2,2,3,5>
normal form
FC
<> <0,1,2,2,3,5>
SC
<3,1,2>
<0,2,5>
<1,2>
<0,2,3,5>
<3,2>
<0,1,2,5>
<3,1>
<0,2,2,5>
<1>
<0,2,2,3,5>
<2>
<0,1,2,3,5>
<3>
<0,1,2,2,5>

ZK 5

Teorie programovacích jazyku. zkouška 31. 1. 2013

1. Napište co nejobecnější typ funkce f. Nápověda: pokud nedovedete určit přesný typ některé proměnné, použijte parametrický polymorfismus. Měli byste nějak reflektovat to, že některé proměnné se používají jako funkce a některé jako jejich parametry.

$$f(w, x, y, z) = \text{if } w(x) = y(z) \text{ then } w(x) \text{ else } f(w, w(z), y, y(x)) \qquad (1)$$

2. Akce je funkce, která v parametru dostane stav a buď proběhne v pořádku a vrátí nový stav nebo selže a vrátí $\perp$.

$$Action = State \to (State \cup \{\perp\}), \quad \perp \notin State \qquad (2)$$

Napište definici funkce $runInTransaction : Action^* \to (State \to State)$, která vrátí funkci, která buď zřetězí všechny akce a vrátí výsledný stav nebo, pokud některá z akcí v průběhu vykonávání selže, vrátí svůj počáteční stav. Nápověda: možná vám pomůže nadefinovat si pomocnou funkci.

3. Sémantika definovaná přepisovací relací $\to$ postupně bere instrukce ze zásobníku nalevo a manipuluje se zásobníkem hodnot napravo. Bohužel, pokud se programátor pokusí o operace typu součet dvou logických hodnot, negace neexistující hodnoty apod., sémantika se zasekne. Vaším úkolem je dodat do jazyka hodnotu *Error* a upravit relaci $\to$ tak, aby místo zaseknutí doběhla do konfigurace s nulovým počtem nezpracovaných instrukcí a na vrcholu zásobníku vrátila *Error*.

$$Instruction ::= Num \mid Bool \mid \text{inc} \mid \text{if} \qquad (3)$$

Množina hodnot $Value$ je definovaná jako $Value = Bool \cup Num$. Množina konfigurací je $Instruction^* \times Value^*$. Použité jmenné konvence: $n, n' \in Num$, $b, b' \in Bool$, $v_1, \dots \in Value$ a $i_1, \dots \in Instruction$.

$$\langle n, i_1, \dots, i_a \rangle, \langle v_1, \dots, v_b \rangle \to \langle i_1, \dots, i_a \rangle, \langle n, v_1, \dots, v_b \rangle \qquad (4)$$

$$\langle b, i_1, \dots, i_a \rangle, \langle v_1, \dots, v_b \rangle \to \langle i_1, \dots, i_a \rangle, \langle b, v_1, \dots, v_b \rangle \qquad (5)$$

$$\langle \text{inc}, i_1, \dots, i_a \rangle, \langle n, v_1, \dots, v_b \rangle \to \langle i_1, \dots, i_a \rangle, \langle n+1, v_1, \dots, v_b \rangle \qquad (6)$$

$$\langle \text{if}, i_1, \dots, i_a \rangle, \langle true, n, n', v_1, \dots, v_b \rangle \to \langle i_1, \dots, i_a \rangle, \langle n, v_1, \dots, v_b \rangle \qquad (7)$$

$$\langle \text{if}, i_1, \dots, i_a \rangle, \langle false, n, n', v_1, \dots, v_b \rangle \to \langle i_1, \dots, i_a \rangle, \langle n', v_1, \dots, v_b \rangle \qquad (8)$$

4. Nakreslete graf znázorňující všechny způsoby přepsání konfigurace $\langle 3, 1, 2 \rangle, \langle 0, 2, 5 \rangle$ na normální formu.

$$\langle a_1, \dots, a_n \rangle, \langle b_1, \dots, b_m \rangle \to \langle a_1, \dots, a_i, a_{i+1}, \dots, a_n \rangle, \langle b_1, \dots, b_j, a_i, b_{j+1}, \dots, b_m \rangle \quad \text{kde } i \in \{1, \dots, n\} \wedge b_j \le a_i \le b_{j+1} \qquad (9)$$

5. Svými slovy (ale přesně a srozumitelně) vysvětlete co konkrétně ve Featherweight Javě kontroluje typové pravidlo T-Method.

Answer

- Value = Bool ∪ Num ∪ Err

$\langle inc, i_1 \dots i_a \rangle \langle b, v_1 \dots v_b \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err, v_1 \dots v_b \rangle$

$\langle inc, i_1 \dots i_a \rangle \langle \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err \rangle$

$\langle inc, i_1 \dots i_a \rangle \langle err, v_1 \dots v_b \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err, v_1 \dots v_b \rangle$

$\langle if, i_1 \dots i_a \rangle \langle \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err \rangle$

$\langle if, i_1 \dots i_a \rangle \langle true, v_1 \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle \rangle$

$\langle if, i_1 \dots i_a \rangle \langle false, v_1 \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err \rangle$

$\langle if, i_1 \dots i_a \rangle \langle n, v_1, v_2, v_3 \dots v_b \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err, v_3 \dots v_b \rangle$

$\langle if, i_1 \dots i_a \rangle \langle err, v_1, v_2, v_3 \dots v_b \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle err, v_3 \dots v_b \rangle$

~~$\langle if, i_1 \dots i_a \rangle \langle false, v_1, e \rangle$~~

$\langle if, i_1 \dots i_a \rangle \langle true, v_1, err, v_2 \dots v_b \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle v_1, v_2 \dots v_b \rangle$

$\langle if, i_1 \dots i_a \rangle \langle false, err, v_1, v_2 \dots v_b \rangle \longrightarrow \langle i_1 \dots i_a \rangle \langle v_1, v_2 \dots v_b \rangle$

~~$\langle n \rangle \langle err \rangle / \langle err \rangle$~~

$\langle n \rangle \langle err, v_1 \dots v_b \rangle \longrightarrow \langle \rangle \langle err, n, v_1 \dots v_b \rangle$

$\langle b \rangle \langle err, v_1 \dots v_b \rangle \longrightarrow \langle \rangle \langle err, b, v_1 \dots v_b \rangle$

2K B

Teorie programovacích jazyků, zkouška 5. 2. 2013

Množina S je množina všech stavů nějaké datové struktury implementující množinu prvků z množiny A (typ datové struktury je irrelevantní). Nad množinami S a A je nadefinována konstanta $empty : S$, akce $add : S \times A \to S$, $remove : S \times A \to S$ a funkce $contains : S \times A \to Bool$. Doplňte rovnosti, které musí platit pro akce add a $remove$.

$$\forall a \in A : contains(empty, a) = false \quad (1)$$

$$\forall s \in S, a, a' \in A : contains(add(s, a), a') = \text{if } a = a' \text{ then true else } contains(s, a') \quad (2)$$

$$\forall s \in S, a, a' \in A : contains(remove(s, a), a') = \text{if } a = a' \text{ then false else } contains(s, a') \quad (3)$$

2. Featherweight Java podporuje vyhodnocování strategií call-by-name i call-by-value. Odstraňte několik pravidel (napište jejich tvrzení) jeho semantiky tak, aby podporovala pouze strategii call-by-name.

3. Napište libovolnou funkci f, která nějaký ze svých parametrů použije jako funkci a vrátí

$$f : \forall A <: Top, B <: Top, C <: Top, D <: Top, E <: Top. (A \to B \times C) \times D \to E \quad (4)$$

$f(x, y) = x$

4. Představte si Featherweight Javu rozšířenou o booleovské hodnoty $true$, $false$ a podmíněný výraz

$$e ::= x \mid e.f \mid e.m(\bar{e}) \mid new\ C(\bar{e}) \mid (C)\ e \mid true \mid false \mid \text{if } e \text{ then } e \text{ else } e \quad (6)$$

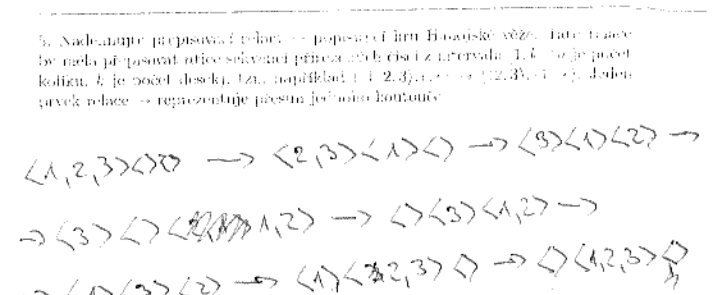
Doplňte typový systém FJ o jednoduchá pravidla odvozující jednoduchý typ $Bool$ a o sémantiky FJ přidejte jedno pravidlo popisující případnou hodnotu $true$.

5. Nadefinujte přepisovací relaci $\to$ popisující hru Hanojské věže. Tato hra může být hrána při použití různého počtu disků n a různého počtu kolíků k. n je počet disků, tzn. například $1 \le 2, 3, \ldots$ a k je počet kolíků, tzn. například $1 \le 2, 3, \ldots$. Jeden prvek relace $\to$ reprezentuje přesun jednoho kotouče.

$x : A \to (B \times C)$
$y : D$
$\Big\} x \to E$

$\lambda left, right.$
$\lambda left, \lambda D . \lambda S\ D(\lambda A(left(s)))$

$left = \lambda B, C$



1. Dokud $n > 1$, přesuň $n-1$ kotoučů na odkládací (3.) věž z (1.)
2. Přesuň největší kotouč z (1.) na cílovou (2.)
3. Dokud $n > 1$, přesuň $n-1$ z odkládací na cílovou

```
do(n, to, from, using) {
  if n == 0  return
  else:
    do(n-1, using, from, to)
    move(from, to)
    do(n-1, to, using, from)
```

Zk 1/1 definice **ForEach**

$ForEach(\langle a_1 \dots a_n \rangle, f) = \lambda s.\ loop(\langle a_1 \dots a_n \rangle, f, s)$

$loop(\langle\rangle, f, s) = s$ – na konci pracuje with žádný stav (prázdný vstup)

$loop(\langle a_1, a_2 \dots a_n \rangle, f, s) = loop(\langle a_2 \dots a_n \rangle, f, (f\ a_1)\ s)$

↳ nový stav, který vznikl tím, že f nad prvkem a_1

$loop: \left(A^* \times (A \times State \to State) \times State\right) \to State$

Zk 3/3 **doWhile** $\left((State \to State) \times (State \to Bool)\right) \to (State \to State)$

$doWhile(body, cond) = \lambda s.\ loopF(body, cond, s)$

$loopF(body, cond, s) = \lambda s.\ if\ (cond(body(s)))\ then\ doWhile(body, cond)(body(s))$
$else\ (body(s))$

Zk 4/4 **or** $(State \to Bool \times State) \times (State \to (Bool \times State)) \to (State \to (Bool \times State))$

left operand | right operand

$or = \lambda left, right.\ \lambda s.\ if\ (value(left(s)))\ then\ left(s)$ (ti vrátí TRUE a na pravé straně vůbec nezáleží ...)

$else\ right(state(left(s)))$

$Bool \times State \Rightarrow \lambda b, s.\ b$ ← zřetězen book

$Bool \times State \Rightarrow \lambda b, s.\ s$ ← zřetězen state

Teorie programovacích jazyků, zkouška 21. 12. 2012

1. Nadefinujte funkci *forEach* tak, aby vrátila akci, která postupně projde všechny prvky sekvence zadané v prvním parametru funkce *forEach* a na každém z nich vykoná akci zadanou v druhém parametru funkce *forEach*.

$$forEach : A^* \times (A \times State \to State) \to (State \to State)$$

akce — f — loop = ze stavu do stavu

- forEach $(\langle a_1 \dots a_n \rangle, f) = \lambda i.\ loop(\langle a_1 \dots a_n \rangle, f, i)$
  - $\langle a_1 \dots a_n \rangle$ = sekvence A^*
  - i = State, vstupní stav
- loop: $(A^* \times (A \times State \to State) \times State) \to State$
  - A^* → $\langle a_1 \dots a_n \rangle$
  - $(A \times State \to State)$ → f
  - $State$ → i
- pro prázdný řetězec vrací loop vstupní stav.
  - $loop(\langle\rangle, f, s) = s$ ✓
- pro neprázdný řetězec aplikuje funkci na první akci a_1, (získáno funkcí (state → state) pak to, co vrátí f) aplikuje na vstupní stav a získává znovu pomocí do loop se zbytkem akcí:
  - $loop(\langle a_1, a_2 \dots a_n \rangle, f, s) = loop(\langle a_2 \dots a_n \rangle, f, (f\ a_1)\ s)$ ✓

Magista ν λ calculu reducimi fei pro $\sum_{i=1}^{n} i$

$G = \lambda f . \lambda x$ if (isZero x) Then 0 else (f (Dec x) + x)

Vedi ce s Y substituiteren : YG

2. Dokažte následující tvrzení: pokud $(\vdash t : T) \wedge (t \Rightarrow t')$, pak $\vdash t' : T$.

$$t ::= \text{true} \mid \text{false} \mid \\ 0 \mid \text{succ } t \mid \\ \text{if } t \text{ then } t \text{ else } t \tag{1}$$

$$\frac{}{\text{if true then } t_1 \text{ else } t_2 \Rightarrow t_1} \tag{2}$$

$$\frac{}{\text{if false then } t_1 \text{ else } t_2 \Rightarrow t_2} \tag{3}$$

$$\frac{t_1 \Rightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \Rightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3} \tag{4}$$

$$\frac{t_1 \Rightarrow t_1'}{\text{succ } t_1 \Rightarrow \text{succ } t_1'} \tag{5}$$

$$\frac{}{\vdash \text{true} : Bool} \tag{6}$$

$$\frac{}{\vdash \text{false} : Bool} \tag{7}$$

$$\frac{}{\vdash 0 : Nat} \tag{8}$$

$$\frac{\vdash t : Nat}{\vdash \text{succ } t : Nat} \tag{9}$$

$$\frac{\vdash t_1 : Bool \qquad \vdash t_2 : T \qquad \vdash t_3 : T}{\vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \tag{10}$$

⇒ je třeba u každého pravidla dokázat, že pokud o něm dokážu k nějakému přepisu jazyka, že se nezmění jeho typ.

↳ tzn. že je nutné dokázat pro pravidla (2)-(5), ostatní jsou zřejmé / axiomy

(2) pokud leží výraz: „if true then t_1 else t_2" je typu T (podle pravidla 10), tak potom je totéž platí výraz „t_1" je typu T (pravidlo 10)

(3) totéž jako u (2), ale na pravé straně je výraz „t_2" – podle (10) také typ T

(4) podle pravidla (10) musí být „t_1" resp. „t_1'" na druhé straně typu Bool, proto se přepisuje: $t_1 \Rightarrow t_1'$ jako: * Bool => Bool, typ zůstává tedy zachován.

(5) podle pravidla (9) musí být succ t_1 typu Nat, totéž pro t_1' ; přepis je tedy z Nat => Nat *, typ zůstává zachován

3. Dokažte, že přepisovací relace $\Rightarrow$ definovaná v předcházejícím příkladě je silně normalizující (vždy terminuje).

=) Důkaz pomocí energie. Tedy budeme ~~přepisovací pravidla~~ na nějakém energii definici $\text{Exp}/t ::=$

$\text{energy}(\text{value}) = 0$ $\quad$ $\text{value} = \{\text{true}, \text{false}, 0\}$

$\text{energy}(\text{succ } t) = 1 + \text{energy}(t)$

$\text{energy}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) = 1 + \text{energy}(t_1) + \text{energy}(t_2) + \text{energy}(t_3)$

(Exp), ale zde $t ::= \ldots$)

- $\forall e, e' \in t : e \rightsquigarrow e' \Rightarrow \text{energy}(e) > \text{energy}(e')$
- $\forall e \in t : \text{energy}(e) \in \mathbb{N}$

=) tedy s každým přepisem se energie zmenší do doby, než bude co přepisovat

4. Sémantika S má vstupní funkci $\lambda z, e.(z, e)$, výstupní funkci $\lambda(z, z').z'$, množinu konfigurací $Z \times Expr$, množinu finálních konfigurací $Z \times Z$ a přepisovací relaci $\Rightarrow$. Napište ekvivalentní denotační sémantiku, jako sémantickou doménu použijte $Z \to Z$.

$$Expr ::= Num \mid \triangle Expr \mid Expr \odot Expr \mid \square \tag{11}$$

$$\frac{}{(z, \triangle z') \Rightarrow (z, -z')} \tag{12}$$

$$\frac{}{(z, z' \odot z'') \Rightarrow (z, z' + z'')} \tag{13}$$

$$\frac{(z, e) \Rightarrow (z, e')}{(z, \triangle e) \Rightarrow (z, \triangle e')} \tag{14}$$

$$\frac{(z, e) \Rightarrow (z, e')}{(z, e \odot e'') \Rightarrow (z, e' \odot e'')} \tag{15}$$

$$\frac{(z, e'') \Rightarrow (z, e')}{(z, e \odot e'') \Rightarrow (z, e \odot e')} \tag{16}$$

$$\frac{}{(z, \square) \Rightarrow (z, z)} \tag{17}$$

$[\![z']\!] = \lambda z . z'$
$[\![\square]\!] = \lambda z . z$
$[\![\triangle e]\!] = [\![\triangle]\!]([\![e]\!])$
$[\![e \odot e']\!] = [\![\odot]\!]([\![e]\!], [\![e']\!])$

} objekty sém.: $z \to z$ (tj. funkce z hodnoty pásku $\square$ do hodnoty pásku)

$[\![\triangle]\!] = \lambda e . \lambda z . -e(z)$
$[\![\odot]\!] = \lambda e, e' . \lambda z . e(z) + e'(z)$

} operace sém.: $(z \to z) \to (z \to z)$

př.: $[\![\triangle 1]\!] = [\![\triangle]\!][\![1]\!] = (\lambda e . \lambda z . -e(z))(\lambda z . 1) = \lambda z . -(\lambda z . 1)(z) = \lambda z . -1$

+ potřebné ustupní / postupní funkce: $\lambda z, e . [\![e]\!](z)$

---

zápis pomocí env: (pozdější zpracování):

$[\![n]\!] = \lambda env . n$
$[\![v]\!] = \lambda env . env(v)$
$[\![\triangle e]\!] = [\![\triangle]\!]([\![e]\!])$
$[\![e \odot e']\!] = [\![\odot]\!]([\![e]\!], [\![e']\!])$

$[\![\triangle]\!] = \lambda e . \lambda env . -e(env)$
$[\![\odot]\!] = \lambda e, e' . \lambda env . e(env) + e'(env)$

sémantické : env $\not\in$ VarName $\to$ Num
v $\not\in$ VarName
n $\not\in$ Num
e $\not\in$ Expr

1. Napište výraz lambda kalkulu implementující funkci f. Pro přehlednost používejte symbolická jména standardních kombinátorů (např. 0) a jejich definice rozepište vedle. Nápověda: nejprve zkonstruujte generátor funkce f.

$$f(1,x,y) = x(1,y(0)) \tag{1a}$$
$$f(n,x,y) = x(n,f(n-1,x,y)) \quad \text{pro } n > 1 \tag{1b}$$

$F = Y\ \lambda f.\lambda n.\lambda x.\lambda y.$ IF (ISZERO (DEC n))(x 1 (y 0))(x n (f (DEC n) x y))

↳ pravděpodobně:

Dec n-1 (pokud $n \neq 1$) $F(1,x,y) =$

kombinátory:

IF = $\lambda p.\lambda a.\lambda b.\ p\ a\ b$

ISZERO = $\lambda n.\ n\ (\lambda x.\text{FALSE})\ \text{TRUE}$

TRUE = $\lambda x.\lambda y.\ x$

FALSE = $\lambda x.\lambda y.\ y$

DEC n = předchůdce čísla n do λ:

0 = $\lambda s z.\ z$

1 = $\lambda s z.\ s(z)$

2 = $\lambda s z.\ s(s(z))$

⋮

$Y = \lambda g.(\lambda x.\ g\ (x\ x))\ (\lambda x.\ g\ (x\ x))$

SUCC = S = $\lambda n.\lambda f.\lambda x.\ f\ (n\ f\ x)$

5. Pravidlo 18 je pravidlo pro podtypování rekurzivních typů. Na příkladu vysvětlete, proč by pravidlo 19 nefungovalo. Svoje zdůvodnění opřete o vlastnosti programovacích jazyků jako je soundness, progress apod.

$$\frac{\Gamma \vdash \mu X.A \quad \Gamma \vdash \mu Y.B \quad \Gamma \cup \{Y <: Top, X <: Y\} \vdash A <: B}{\Gamma \vdash \mu X.A <: \mu Y.B} \tag{18}$$

$$\frac{\Gamma \vdash \mu X.A \quad \Gamma \vdash \mu Y.B \quad \Gamma \cup \{Y <: Top, X <: Top\} \vdash A <: B}{\Gamma \vdash \mu X.A <: \mu Y.B} \tag{19}$$

- Pravidlo (19) by nefungovalo - protipříklad:

$\{\} \vdash \mu X.X <: \mu Y.Y$ ⟶ $\{Y <: TOP, X <: TOP\} \vdash X <: Y$

(pokud se při ověřování dostane k tomu, což nepůjde)

- Nebo protipříklad v Javě: viz ZK1

5 / (E-1)

1. Napište co nejobecnější typ funkce f. Nápověda: typ proměnné a se dá odvodit z kontextu, přesné typy proměnných b, c a d neznáte, proto použijte parametrický polymorfismus. Měli byste nějak reflektovat to, že c je použito jako parametr funkce b a že výsledek volání funkce f je buď $b(c)$ nebo d.

$$f(a, b, c, d) = \text{if } a \text{ then } b(c) \text{ else } d \tag{1}$$

$f : \forall C, D \cdot (Bool \times (C \to D) \times C \times D) \to D$

Vysvětlení: $f(a, b, c, d)$

- argument „a" → obsahuje se jako logický výraz → bool
- argument „c, d" → o nich nic nevíme, takže jsou generické $(\forall C, D)$, ale umíme odvodit vztahy mezi nimi
  - * o třetí větvi je řeč: „d" → takže má mít stejný typ jako výstup celé funkce (D)
  - * o then větvi je řeč: „b(c)" → takže víme, že „b" je funkce, kterou přijímá typ C a vrací totéž, co celá funkce – tedy D ⇒
    ⇒ takže b je $(C \to D)$

⇒ sedíme a máme jasno

2. Rozšiřte definici funkce m (implementující rozpoznávání regulárních výrazů) o případ $m(r \times n, \langle a_1, \ldots, a_n \rangle, k)$ tak, aby výraz $r \times n$ byl, neformálně řečeno, syntaktická zkratka za

$$\underbrace{r \cdot r \cdot \ldots \cdot r}_{n}, \quad \text{kde } n \geq 0. \tag{2}$$

$$\begin{aligned} RegExp ::= {} & \epsilon \\ & A \\ & RegExp \cdot RegExp \\ & RegExp + RegExp \\ & RegExp \times N \end{aligned} \tag{3}$$

$$A^{\perp} = A^* \cup \{\perp\} \tag{4}$$

$$m : RegExp \times A^{\perp} \times (A^{\perp} \to A^{\perp}) \to A^{\perp} \tag{5}$$

$$m(r, \perp, k) = k(\perp) \tag{6}$$

$$m(\epsilon, \langle\rangle, k) = k(\langle\rangle) \tag{7}$$

$$m(\epsilon, \langle a_1, \ldots, a_n \rangle, k) = k(\langle a_1, \ldots, a_n \rangle) \tag{8}$$

$$m(a, \langle\rangle, k) = k(\perp) \tag{9}$$

$$m(a, \langle a_1, a_2, \ldots, a_n \rangle, k) = k(\langle a_2, \ldots, a_n \rangle) \quad \text{pokud } a = a_1 \tag{10}$$

$$m(a, \langle a_1, \ldots, a_n \rangle, k) = k(\perp) \quad \text{pokud } a \neq a_1 \tag{11}$$

$$m(r_1 \cdot r_2, \langle a_1, \ldots, a_n \rangle, k) = m(r_1, \langle a_1, \ldots, a_n \rangle, \lambda x.m(r_2, x, k)) \tag{12}$$

$$\begin{aligned} m(r_1 + r_2, \langle a_1, \ldots, a_n \rangle, k) = {} & m(r_1, \langle a_1, \ldots, a_n \rangle, \\ & \lambda x.\text{if } k(x) = \langle\rangle \text{ then } \langle\rangle \text{ else } m(r_2, \langle a_1, \ldots, a_n \rangle, k)) \end{aligned} \tag{13}$$

$$match : RegExp \times A^* \to Boolean \tag{14}$$

$$match(r, \langle a_1, \ldots, a_n \rangle) = true \quad \text{pokud } m(r, \langle a_1, \ldots, a_n \rangle, \lambda x.x) = \langle\rangle \tag{15}$$

$$match(r, \langle a_1, \ldots, a_n \rangle) = false \quad \text{jinak} \tag{16}$$

(podobně pro r^* v TPJ (4.6))

- nulý případ:

$$m(r \times 0, \langle a_1 \ldots a_n \rangle, k) = k(\langle a_1 \ldots a_n \rangle)$$

- ostatní možnosti n $(n > 0)$:

$$m(r \times n, \langle a_1 \ldots a_n \rangle, k) = m(r \cdot r \times n', \langle a_1 \ldots a_n \rangle, k) \quad (\text{kde } n' = n-1)$$

(nebo lépe takto:

$$m(r \cdot (r \times (n-1)), \langle a_1 \ldots a_n \rangle, k)$$

)

3. Nadefinujte funkci *doWhile* tak, aby vrátila akci, která reprezentuje do ... while cyklus. Pozor: v tomto cyklu se podmínka testuje až po provedení těla cyklu, každý do ... while cyklus proto proběhne alespoň jednou.

$$doWhile : (State \to State) \times (State \to Bool) \to (State \to State) \qquad (17)$$

$doWhile(body, condition) =$

doWhile (body, condition) = $\lambda s.$ IF (cond* (s)) THEN

(doWhile (body, cond)(body(s))) ELSE (body(s))

zdůvodnění: pokud není podmínka splněna, je jeden ohodnocení těla funkce nad stavem s (do ...) – body(s)

pokud je splněna, zavolám se znovu while nad modifikovaným stavem

* OPRAVA:

IF (cond (body(s))) → musím zkontrolovat podmínku po změně stavu,

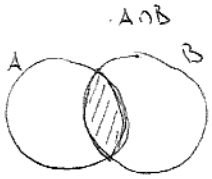
dále: pokud by v průběhu změnil T→F, tak se while provede 2×

4. Průnik typů A a B, značený $A \cap B$, je typ, jehož instance mají zároveň typ A i B. Formálně:

$$\frac{e : A \quad e : B}{e : A \cap B}. \tag{18}$$

Rozhodněte, zda platí $A <: A \cap B$ nebo $A \cap B <: A$ (případně oboje). Svoje tvrzení dobře zdůvodněte. Nápověda: zamyslete se nad subsumpcí.

- lze ilustrovat na Vennových diagramech:



① $A <: A \cap B$ neplatí → nebylo by každý bod z A (který ale nebyl v $A \cap B$ (není podtypem)

② $A \cap B <: A$ platí → pokud zvolím libovolný bod z $A \cap B$, je to také bod v A (je též podtypem)

- vyšší složitost:

aby existovalo $A \cap B$, musí existovat $A <: B$ nebo $B <: A$ (úplná hierarchie nad stejnou složitostí)

→ tedy pokud: $A = B$ => pokud platí obě tvrzení ($A \cap B = A \cap A = A$) [~~je~~ je sám sobě podtypem]

jinak: $A \neq B$ => pokud buď:

1) $A <: B$ pak musí $A \cap B <: A$ (jinak by neexistovalo e)

nebo:

2) $B <: A$ pak musí $A \cap B <: B$ a protože $B <: A$ tak také $A \cap B <: A$ (tranzitivita $<:$)

=> $A \cap B <: A$ PLATÍ ②

a pokud $A \neq B$ a zároveň $B <: A$ (z čehož plyne $A \cap B <: B$), pak nemůže platit $A <: A \cap B$ - antisymetrie $<:$

=> $A <: A \cap B$ NEPLATÍ ①

5. Napište definici konfluence a dokažte konfluenci relace $\Rightarrow$ definované níže.

$$\begin{aligned} Expr ::= {} & Num \mid \\ & \triangle Expr \mid \\ & Expr \odot Expr \end{aligned} \tag{19}$$

$$\frac{}{\triangle n \Rightarrow -n} \tag{20}$$

$$\frac{}{n \odot n' \Rightarrow n + n'} \tag{21}$$

$$\frac{e \Rightarrow e'}{\triangle e \Rightarrow \triangle e'} \tag{22}$$

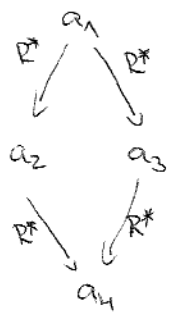
$$\frac{e_1 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e' \odot e_2} \tag{23}$$

$$\frac{e_2 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e_1 \odot e'} \tag{24}$$

Definice konfluence:

- Relace $R \subseteq A \times A$ je konfluentní právě když platí:

$$\forall a_1, a_2, a_3 \in A : a_1 R^* a_2 \wedge a_1 R^* a_3 \Rightarrow \exists a_4 \in A : a_2 R^* a_4 \wedge a_3 R^* a_4$$

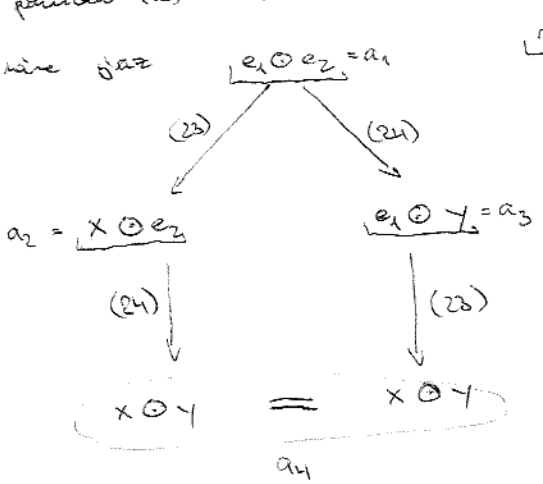


- konfluence implikuje single normal form (jsou NF / unikátní ~~NF~~ for term)

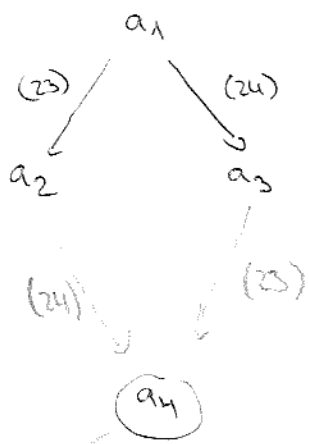
Důkaz konfluence:

- pro pravidla (23), (24) a (22) je to jasné, není třeba dokazovat = jsou deterministické, není možné rozhodnout se mezi dvěma

- pro pravidla (22) a (23):



nebo:



↳ pro a_4 hledáme předchozí pravidly přepsat a_2 a a_3 abychom dostali a_4 ⇒ jsou to pravidla:

a_2 – (24)

a_3 – (23)

1. Napište definici funkce *or*. Vrácená akce by měla svůj výsledek vyhodnocovat líně, tzn. vyhodnocovat druhý operand pouze pokud je to nutné. Pozor: druhý operand byste neměli vyhodnocovat nad počátečním stavem.

$$or : (State \to Bool \times State) \times (State \to Bool \times State) \to (State \to Bool \times State) \quad (1)$$

left right left or right

or = λleft, right . λs . IF (VALUE(left(s))) THEN left(s)
ELSE right(state(left(s)))

↳ hledající nad stavem po zhodnocení prvního operandu (levého)

VALUE = λb,s . b

STATE = λb,s . s
Bool × STATE

(pro lepší čitelnost:

or = λleft, right . λs . IF (λb,s . b (left(s))) THEN left(s)
ELSE right(λb,s . s (left(s)))

2. Napište tělo funkce $innerNodes : (\mu A. \diamond + (A \times A)) \rightarrow Number$, která vrátí počet vnitřních uzlů (tzn. ne listů) zadaného stromu. Nápověda: pravděpodobně budete potřebovat některé z operátorů $unfold$, $inLeft$, $inRight$, $asLeft$, $asRight$, $first$ a $second$.

(2.)

innerNodes = λtree. IF (ISNULL (unfold(tree))) THEN 0
ELSE IF (isLeaf(unfold(tree))) THEN 0
ELSE 1 + innerNodes(first(asNode(unfold(tree)))) +
+ innerNodes(second(asNode(unfold(tree))))

isLeaf = λtree. isNull(first(asNode(unfold(tree)))) and
isNull(second(asNode(unfold(tree))))

isNull = isLeft

asNode = asRight

4. Dokažte nebo vyvraťte (dokažte negaci):

→ stačí projít všechna pravidla (že se projdou a uvidět, že vždy před šipkou je jiný než za ní)

$$\forall e \in Expr, t \in \{Num, Bool\} : (e : t) \implies (e \in (Bool \cup Num) \vee \exists e' \in Expr : e \to e'). \quad (8)$$

} → když se přepíše na Bool/Num nebo jiný Expr

$$\begin{aligned} Expr ::= {} & Num \mid \\ & Bool \mid \\ & \triangle Expr \mid \\ & Expr \odot Expr \mid \\ & \texttt{if } Expr \texttt{ then } Expr \texttt{ else } Expr \end{aligned} \quad (9)$$

Konvence: $e, e', e'', \ldots \in Expr, b, b' \in Bool, n, n' \in Num$ a $t \in \{Num, Bool\}$.

$$\frac{}{\triangle n \to -n} \quad (10)$$

$$\frac{e \to e'}{\triangle e \to \triangle e'} \quad (11)$$

$$\frac{}{n \odot n' \to n + n'} \quad (12)$$

$$\frac{e' \to e''}{e \odot e' \to e \odot e''} \quad (13)$$

$$\frac{e \to e''}{e \odot e' \to e'' \odot e'} \quad (14)$$

$$\frac{}{\texttt{if } true \texttt{ then } e \texttt{ else } e' \to e} \quad (15)$$

$$\frac{}{\texttt{if } false \texttt{ then } e \texttt{ else } e' \to e'} \quad (16)$$

$$\frac{e \to e'''}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' \to \texttt{if } e''' \texttt{ then } e' \texttt{ else } e''} \quad (17)$$

$$\frac{e' \to e'''}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' \to \texttt{if } e \texttt{ then } e''' \texttt{ else } e''} \quad (18)$$

$$\frac{e'' \to e'''}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' \to \texttt{if } e \texttt{ then } e' \texttt{ else } e'''} \quad (19)$$

$$\frac{}{n : Num} \quad (20)$$

$$\frac{}{b : Bool} \quad (21)$$

$$\frac{e : Num}{\triangle e : Num} \quad (22)$$

$$\frac{e : Num \quad e' : Num}{e \odot e' : Num} \quad (23)$$

$$\frac{e : Bool \quad e' : t \quad e'' : t}{\texttt{if } e \texttt{ then } e' \texttt{ else } e'' : t} \quad (24)$$

4/ (24)

5. Napište denotační sémantiku jazyka s níže uvedenou syntaxí (operátor $\ominus$ intuitivně reprezentuje mínus, operátor $\oplus$ plus). Jako sémantickou doménu použijte funkci, která k mapování jmen na hodnoty vrátí číslo $((VarName \to Num) \to Num)$. Nápověda: chceme po vás napsat funkci $Expr \to ((VarName \to Num) \to Num)$.

návratový typ — sémantická doména

$$
\begin{aligned}
Expr ::= \; & Num \mid \\
& VarName \mid \\
& \ominus Expr \mid \\
& Expr \oplus Expr
\end{aligned}
\qquad (25)
$$

$(VarName \to Num) \to m$ … mapování

$[\![n]\!] = \lambda m.\, n$

$[\![v]\!] = \lambda m.\, m(v)$

$[\![e + f]\!] = \lambda m.\, [\![+]\!]([\![e]\!], [\![f]\!])$

$[\![-e]\!] = \lambda m.\, [\![-]\!]([\![e]\!])$

} objekty

$[\![-]\!] = \lambda e.\, \lambda m.\, -e(m)$

$[\![+]\!] = \lambda e, f.\, \lambda m.\, e(m) + f(m)$

$\lambda e.\, \lambda f$

} operace

5/ 24

3. Upravte relaci $\Rightarrow$ tak, aby sémantika zůstala stejná, ale vyhodnocování výrazů probíhalo striktně zleva doprava.

$$
\begin{aligned}
Expr ::= \; & Num \mid \\
& \triangle Expr \mid \\
& Expr \odot Expr
\end{aligned}
\qquad (2)
$$

Konvence: $e, e', e_1, e_2 \in Expr$ a $n, n' \in Num$.

$$\frac{}{\triangle n \Rightarrow -n} \qquad (3)$$

$$\frac{}{n \odot n' \Rightarrow n + n'} \qquad (4)$$

$$\frac{e \Rightarrow e'}{\triangle e \Rightarrow \triangle e'} \qquad (5)$$

$$\frac{e_1 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e' \odot e_2} \qquad (6)$$

$$\frac{e_2 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e_1 \odot e'} \qquad (7)$$

Jinak se dělá sémantika zhodnocení zleva

Musíš vymyslet na: $\frac{e_2 \Rightarrow e'}{n \odot e_2 \Rightarrow n \odot e'}$

3/ F4

2. Akce je funkce, která v parametru dostane stav a buď proběhne v pořádku a vrátí nový stav nebo selže a vrátí $\perp$.

$$Action = State \to (State \cup \{\perp\}), \quad \perp \notin State \tag{2}$$

Napište definici funkce $runInTransaction : Action^* \to (State \to State)$, která vrátí funkci, která buď zřetězí všechny akce a vrátí výsledný stav nebo, pokud některá z akcí v průběhu vykonávání selže, vrátí svůj počáteční stav. Nápověda: možná vám pomůže nadefinovat si pomocnou funkci.

runInTransaction = RIT : Action* → (State → State)

- $RIT(\langle a_1 \cdots a_n \rangle) = \lambda s.\, s$ pokud $Fail(\langle a_1 \cdots a_n \rangle, s)$

  $= \lambda s.\, RIT(\langle a_2 \cdots a_n \rangle)(a_1(s))$ jinak než FAIL

- FAIL $(\langle a_1 \cdots a_n \rangle, s)$:

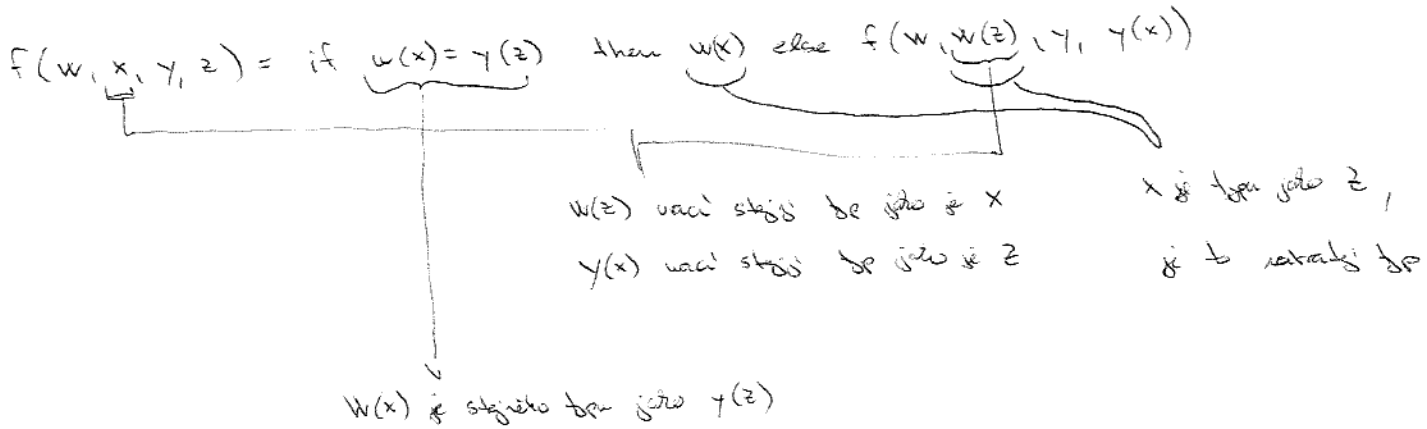
  $Fail(\langle\rangle, s) = FALSE$

  $Fail(\langle a_1 \cdots a_n \rangle, s) = TRUE$ pokud $a_1(s) = \perp$

  $Fail(\langle a_1 \cdots a_n \rangle, s) = Fail(\langle a_2 \cdots a_n \rangle, a_1(s))$ pokud $a_1(s) \neq \perp$

1. Napište co nejobecnější typ funkce f. Nápověda: pokud nedovedete určit přesný typ některé proměnné, použijte parametrický polymorfismus. Měli byste nějak reflektovat to, že některé proměnné se používají jako funkce a některé jako jejich parametry.

$$f(w,x,y,z) = \text{if } w(x) = y(z) \text{ then } w(x) \text{ else } f(w, w(z), y, y(x)) \qquad (1)$$



$w : A \to B$
$x : B$
$y : A \to B$
$z : B$
} → do x dosazuji w → $w : B \to B$, $x : B$
} → —"— =>

=> jsem generický typ:

$f : \forall A . ((A \to A) \times A \times (A \to A) \times A) \to A$

3. Sémantika definovaná přepisovací relací $\rightarrow$ postupně bere instrukce ze zásobníku nalevo a manipuluje se zásobníkem hodnot napravo. Bohužel, pokud se programátor pokusí o operace typu součet dvou logických hodnot, negace neexistující hodnoty apod., sémantika se zasekne. Vaším úkolem je dodat do jazyka hodnotu *Error* a upravit relaci $\rightarrow$ tak, aby místo zaseknutí doběhla do konfigurace s nulovým počtem nezpracovaných instrukcí a na vrcholu zásobníku vrátila *Error*.

$$Instruction ::= Num \mid Bool \mid \text{inc} \mid \text{if} \tag{3}$$

Množina hodnot $Value$ je definovaná jako $Value = Bool \cup Num$. Množina konfigurací je $Instruction^* \times Value^*$. Použité jmenné konvence: $n, n' \in Num$, $b, b' \in Bool$, $v_1, \cdots \in Value$ a $i_1, \cdots \in Instruction$.

$$\langle n, i_1, \ldots, i_a \rangle, \langle v_1, \ldots, v_b \rangle \rightarrow \langle i_1, \ldots, i_a \rangle, \langle n, v_1, \ldots, v_b \rangle \tag{4}$$

$$\langle b, i_1, \ldots, i_a \rangle, \langle v_1, \ldots, v_b \rangle \rightarrow \langle i_1, \ldots, i_a \rangle, \langle b, v_1, \ldots, v_b \rangle \tag{5}$$

$$\langle \text{inc}, i_1, \ldots, i_a \rangle, \langle n, v_1, \ldots, v_b \rangle \rightarrow \langle i_1, \ldots, i_a \rangle, \langle n+1, v_1, \ldots, v_b \rangle \tag{6}$$

$$\langle \text{if}, i_1, \ldots, i_a \rangle, \langle true, n, n', v_1, \ldots, v_b \rangle \rightarrow \langle i_1, \ldots, i_a \rangle, \langle n, v_1, \ldots, v_b \rangle \tag{7}$$

$$\langle \text{if}, i_1, \ldots, i_a \rangle, \langle false, n, n', v_1, \ldots, v_b \rangle \rightarrow \langle i_1, \ldots, i_a \rangle, \langle n', v_1, \ldots, v_b \rangle \tag{8}$$

obecně chyba při interpretaci:

$\langle inc, i_1 \cdots i_a \rangle, \langle b, v_1 \cdots v_b \rangle \longrightarrow \langle i_1 \cdots i_a \rangle \langle v_1 \cdots v_b, err \rangle$

$\langle inc, i_1 \cdots i_a \rangle, \langle \rangle \longrightarrow \langle i_1 \cdots i_a \rangle \langle err \rangle$ } zde zbytečné navíc: $\longrightarrow \langle \rangle \langle err \rangle$

$\langle inc, i_1 \cdots i_a \rangle, \langle err \rangle \longrightarrow \langle i_1 \cdots i_a \rangle \langle err \rangle$

- chyby při if:

$\langle if, i_1 \cdots i_a \rangle, \langle \rangle \longrightarrow \langle i_1 \cdots i_a \rangle \langle err \rangle$

$\langle if$

:

viz ZKS

NEBO LÉPE

$Instruction ::= Num \mid Bool \mid inc \mid if \mid Error$; $Value = Bool \cup Num \cup Error$

$e_1 \cdots e_n \in Error$

- správné pravidla:

$\langle inc, i_1 \cdots i_a \rangle \langle b, v_1 \cdots v_b \rangle \longrightarrow \langle i_1 \cdots i_a \rangle, e \rangle \langle (b)^*, v_1 \cdots v_b \rangle$

$\langle if, i_1 \cdots i_a \rangle \langle n, v_1 \cdots v_b \rangle \longrightarrow \langle i_1 \cdots i_a, e \rangle \langle (n)^*, v_1 \cdots v_b \rangle$

* jak se má pravidlo objeví interpret?

- zbylá pravidla:

$\langle inc, i_1 \cdots i_a \rangle \langle \rangle \longrightarrow \langle i_1 \cdots i_a \rangle, e \rangle \langle \rangle$

$\langle if, i_1 \cdots i_a \rangle \langle v_1 \cdots v_b \rangle \longrightarrow \langle i_1 \cdots i_a, e \rangle \langle v_1 \cdots v_b \rangle$ kde $b < 3$ *

- zbytečné instrukce Error:

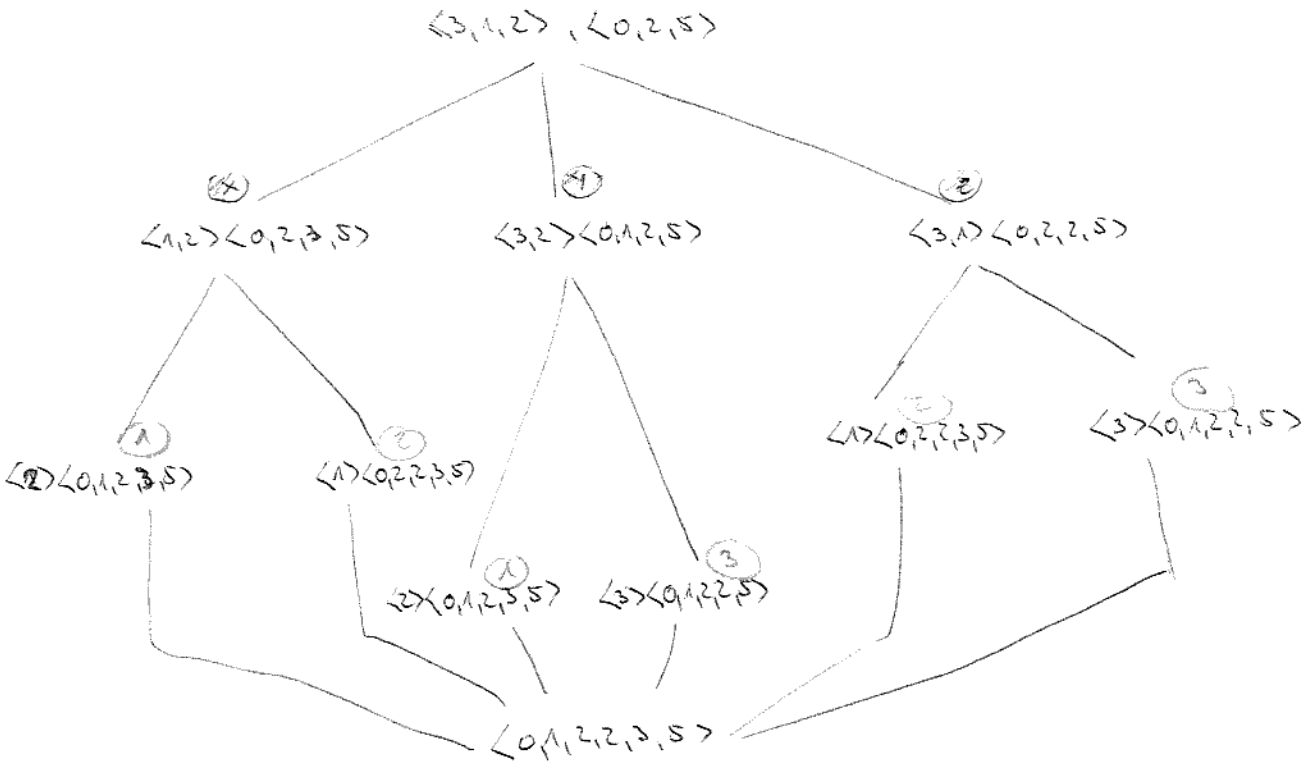
$\langle e, i_1 \cdots i_a \rangle \langle v_1 \cdots v_b \rangle \longrightarrow \langle \rangle \langle e', v_1 \cdots v_b \rangle$

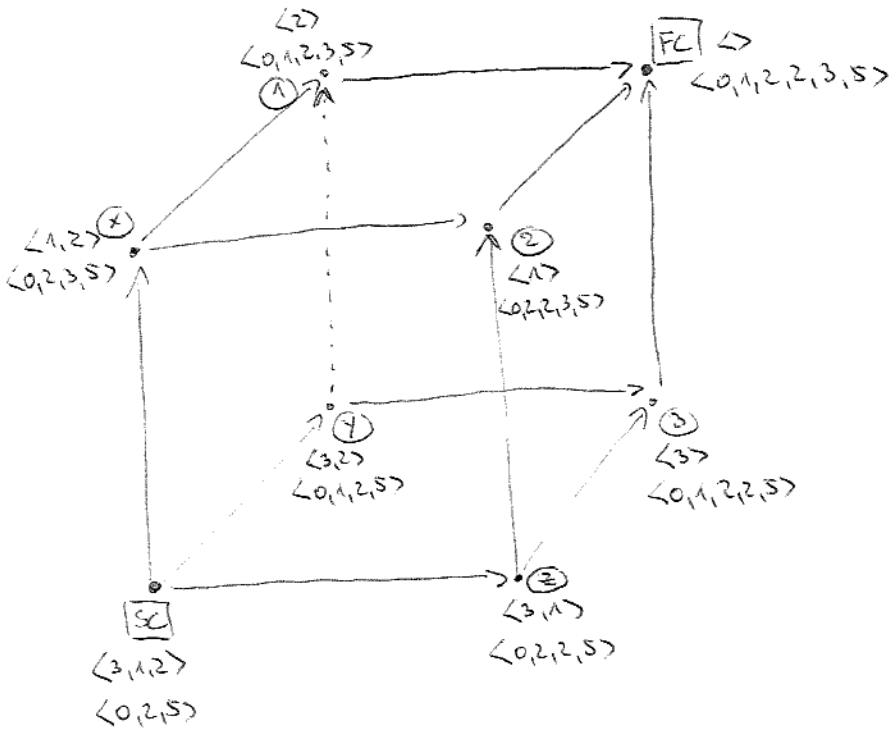
* if (true) by mohlo být $b < 2$, záleží jak se chová interpret

4. Nakreslete graf znázorňující všechny způsoby přepsání konfigurace $\langle 3, 1, 2\rangle, \langle 0, 2, 5\rangle$ na normální formu.

↳ bez dvou vedle přepsání

$$\langle a_1, \ldots, a_n\rangle, \langle b_1, \ldots, b_m\rangle \to \langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\rangle, \langle b_1, \ldots, b_j, a_i, b_{j+1}, \ldots, b_m\rangle \quad (9)$$
$$\text{kde } i \in \{1, \ldots, n\} \wedge b_j \leq a_i \leq b_{j+1}$$





1. Množina S je množina všech stavů nějaké datové struktury implementující množinu prvků z množiny A (typ datové struktury je irelevantní). Nad množinami S a A je nadefinována konstanta $empty : S$, akce $add : S \times A \to S$, $remove : S \times A \to S$ a funkce $contains : S \times A \to Bool$. Doplňte tvrzení, která musí platit pro akce add a $remove$.

$$\forall a \in A : contains(empty, a) = false \tag{1}$$

$$\forall s \in S, a \in A, a' \in A : contains(add(s, a), a') = \tag{2}$$

$$\forall s \in S, a \in A, a' \in A : contains(remove(s, a), a') = \tag{3}$$

---

(2) ③

(2) = if (a==a') then True else False

uiè ZkG

(3) = if (a==a') then False

else if (remove (s,a) == empty) then False

else True

X

---

spíše:

(2) = if (a==a') TRUE else contains (s,a')

potřeji o s nic nevim a to a' tam už klidně může být

(3) = if (contains(s,a') && a==a') FALSE

else if (contains (s,a') && a!=a') TRUE

else FALSE

3. Napište tělo funkce f tak, aby šlo korektně otypovat. Svoje řešení vysvětlete.

$$f : \forall A <: Top.\forall E <: Top.\forall B <: Top.\forall D <: A.\forall C <: E.(A \to (B \times C)) \times D \to E \quad (4)$$

$$f(x, y) = \quad (5)$$

1. Napište co nejobecnější typ funkce f.

$$f(a,b,c) = g(c), \quad \text{kde } g = a(b) \tag{1}$$

$f(a,b,c) = g(c)$ kde $g = a(b)$

$= a(b)(c)$

$f: \forall B,C,D. ((C \to D) \times C \times C) \to B$

LETOŠNÍ PŘÍKLAD:

$f(a,b,c,d) = g(c,d)$ kde $g = a(b)$

$= (a(b))(c,d)$

↳ tj. a přijme b a vrátí fci (z) která přijímá (c,d) a vrací „něco" ($= E$)

v mathematice: $z := \text{Function}[\{c,d\}, e]$, $a := \text{Function}[\{b\}, z]$ $\Big\} \to (a[b])[c,d]$

⇒ víme jen něco o a, o b,c,d nevíme nic → libovolné typy: $\forall B,C,D$

- o funkci z víme, že: přijímá (C,D) a vrací E ⇒ tj.: $z: (C \times D) \to E$
- o funkci a víme, že: přijímá typ B (proč b) a vrací funkci z tj.:

$a: B \to z \Rightarrow B \to ((C \times D) \to E)$

⇒ $f: \underbrace{\forall B,C,D,E}_{\forall B.\forall C.\forall D.\forall E} . \Big(\big(B \to ((C \times D) \to E)\big) \times B \times C \times D \Big) \to E$

kde B,C,D,E mohou být libovolné typy

5. Nadefinujte přepisovací relaci $\rightarrow$ popisující hru Hanojské věže. Tato relace by měla přepisovat $n$tice sekvencí přirozených čísel z intervalu $[1, k]$ (n je počet kolíků, k je počet desek), tzn. například $(\langle 1, 2, 3\rangle, \epsilon, \epsilon) \rightarrow (\langle 2, 3\rangle, \langle 1\rangle, \epsilon)$. Jeden prvek relace $\rightarrow$ reprezentuje přesun jednoho koutouče.

2. Co nejvíc zredukujte výraz $(\lambda x.\lambda x.x)((\lambda x.x\ x)(\lambda x.x\ x))(\lambda x.x)$ pomocí i) normální strategie vyhodnocování a ii) aplikativní strategie vyhodnocování. Dejte si pozor na správné ozávorkování podvýrazů.

1) Normální = tj. nejvíc vlevo: (tj. Call by Name)

$$(\lambda x.\lambda x.x)((\lambda x.x\ x)(\lambda x.x\ x))(\lambda x.x) =$$

↓ (x)

(x se přepíše na λx)

$$= (\lambda x.x)(\lambda x.x) = \lambda x.x$$

2) aplikativní (= Call by Value)

→ zacyklí se na vyhodnocování $((\lambda x.x\ x)(\lambda x.x\ x))$

Ω combinator

Lepší příklad: $(\lambda x.\lambda y.x)(\lambda x.x)((\lambda x.x\ x)(\lambda x.x\ x))$

1) Normální (CBN):

$$(\lambda x.\lambda y.x)(\lambda x.x)(\Omega) = (\lambda y.\lambda x.x)(\Omega) = \lambda x.x$$

2) Aplikativní (CBV):

$$(\lambda x.\lambda y.x)(\lambda x.x)\underbrace{((\lambda x.x\ x)(\lambda x.x\ x))}_{\text{Redex}} =$$

$$= (\lambda x.\lambda y.x)(\lambda x.x)((\lambda x.x\ x)(\lambda x.x\ x))$$

↖ ZDE se to cyklí

3. Napište definici funkce $f : (A \to B) \to (A^* \to B^*)$, která ke vstupní funkci g vrátí funkci, která na každou hodnotu ze svého vstupního seznamu zavolá funkci g a takto získané hodnoty opět vrátí jako seznam.

$f(g) =$ (2)

$g: A \to B$

$f: (A^* \to B^*)$

axi: $f(g) = \lambda i.\ inF(i, g, \{\})$

$inF(\{\}, g, r) = r$

$inF(\{a_1 \cdots a_n\}, g, \{r_1 \cdots r_n\}) = inF(\{a_2 \cdots a_n\}, g, \{r_1 \cdots r_n, g(a_1)\})$

4. Množina S je množina všech stavů nějaké datové struktury implementující kolekci (množinu s opakováním) prvků z množiny A (typ této datové struktury je irelevantní). Nad množinami S a A je nadefinována konstanta $empty : S$, akce $insert : S \times A \to S$, $remove : S \times A \to S$ a funkce $count : S \times A \to Number$. Operace *insert* přidává zadaný prvek do kolekce, *remove* odebírá jeden výskyt zadaného prvku z kolekce (pokud odebíraný prvek v kolekci není, nedělá nic), funkce *count* vrací počet výskytů zadaného prvku v kolekci. Doplňte následující tvrzení pro operace *insert* a *remove*.

$$\forall a \in A : count(empty, a) = 0 \tag{3}$$

$$\forall s \in S, a \in A, a' \in A : count(insert(s, a), a') = \tag{4}$$

$$\forall s \in S, a \in A, a' \in A : count(remove(s, a), a') = \tag{5}$$

(4) if $(a == a')$ then $count(s, a')$ // ukládám prvek, který už tam je → → dojde k přepsání, počet se nezmění

else ~~count~~ $count(s, a') + 1$

↳ je povolené opakování prvků v množině A.

(5) $= count(s, a') + 1$

(5) if $(remove(s, a) == s)$ then $count(s, a')$ → protože ~~nic~~ tam, žádné není nic neuloží

else $count(s, a') - 1$

---

možná (4) if $(a == a')$ $count(s, a') + 1$ else $count(s, a')$

5. Hra života (Game of Life) je definována jako dvoustavový dvourozměrný celulární automat s přechodovou funkcí implementující následující pravidla:

1. Každá živá buňka s méně než dvěma živými sousedy v následující generaci zemře.
2. Každá živá buňka se dvěma nebo třemi živými sousedy v následující generaci zůstává žít.
3. Každá živá buňka s více než třemi živými sousedy v následující generaci zemře.
4. Každá mrtvá buňka s právě třemi živými sousedy v následující generaci oživne.

Uvažujte nekonečnou čtvercovou mřížku a Moorovo okolí (sousední buňky se dotýkají hranou/ortogonálně nebo vrcholem/diagonálně). Konfigurace hry (automatu) c je reprezentována funkcí $c : \mathbb{Z} \times \mathbb{Z} \to Q$; $c(i, j)$ je stav buňky (i, j) v konfiguraci c. $Q = \{0, 1\}$, kde 0 odpovídá mrtvé buňce, 1 odpovídá živé buňce. Čas předpokládáme diskrétní a c^t reprezentuje konfiguraci v čase t.

Definujte:

$$c^{t+1}(i, j) = \begin{cases}$$