

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Jakub Bujak

Nr albumu: 370737

Logika separacji dla języka programowania Jafun

**Praca magisterska
na kierunku INFORMATYKA**

Praca wykonana pod kierunkiem
dr hab. Aleksego Schuberta, prof. UW

Wrzesień 2020

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora pracy

Streszczenie

W pracy zdefiniowano logikę separacji dla języka Jafun, przedstawiono jej formalizację w systemie Coq i udowodniono jej poprawność względem semantyki języka. Logika separacji pozwala na podział sterty na rozłączne fragmenty. Upraszcza to wnioskowanie o programach, pozwalając na dowodzenie własności podwyrażeń na prostszych fragmentach sterty.

Słowa kluczowe

Logika separacji, Jafun, weryfikacja

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

Spis treści

Wprowadzenie	5
1. Podstawowe pojęcia i definicje	7
2. Jafun	9
2.1. Składnia i semantyka	9
2.2. Ewaluacja	11
3. Składnia i semantyka	13
4. Reguły wnioskowania	15
5. Własności stert	19
5.1. Definicje	19
6. Własności ewaluacji	21
6.1. Łączenie ewaluacji	21
6.2. Ewaluacja przy rozszerzonym stosie i kontekście	21
6.3. Ewaluacja zależy tylko od zmiennych wolnych	21
6.3.1. Izomorfizmy	21
6.3.2. Zależność ewaluacji od zmiennych wolnych	23
7. Poprawność	27
8. Formalizacja w systemie Coq	29
9. Podsumowanie	31
Bibliografia	33

Wprowadzenie

Rozdział 1

Podstawowe pojęcia i definicje

Rozdział 2

Jafun

Jafun to zorientowany obiektowo język programowania podobny do Javy. Jego szczegółowy opis znajduje się w pracy [1]. Poniżej przytaczam te aspekty języka, które są istotne dla prezentowanej logiki.

2.1. Składnia i semantyka

Program w języku jafun jest listą definicji klas. Definicja klasy składa się z listy pól i listy metod. Metody mogą przyjmować dowolną liczbę argumentów i rzucać dowolną liczbę wyjątków, deklarowanych przez słowo kluczowe **throws**, podobnie jak w Javie.

Modyfikatory dostępu ϕ i μ nie mają znaczenia w prezentowanej logice, ale zostały uwzględnione w składni dla kompletności opisu.

$$\begin{aligned} \text{Prog} \ni \mathbf{C} &::= \mathbf{class} \ C_1 \ \mathbf{ext} \ C_2 \ \{\bar{\mathbf{F}} \ \bar{\mathbf{M}}\} \\ \text{Cld} \ni C &::= \langle \text{identifier} \rangle \quad (\text{class name}) \\ \mathbf{F} &::= \phi \ C \ x \\ \phi &::= \text{rep} \mid \emptyset \\ \text{Id} \ni x &::= \langle \text{identifier} \rangle \quad (\text{variable/field name}) \\ \text{arg} &::= \mu \ C \ x \quad \text{argn} ::= \emptyset \ C \ x \\ \text{Exc} &::= \mu \ C \quad \text{Excn} ::= \emptyset \ C \\ \mathbf{M} &::= \mu \ C \ \mu \ m(\bar{\text{arg}}) \ \mathbf{throws} \ \bar{\text{Exc}} \ \{E\} \mid \\ &\quad \emptyset \ C \ \emptyset \ m(\bar{\text{argn}}) \ \mathbf{throws} \ \bar{\text{Excn}} \ \{E\} \\ \text{AMod} \ni \mu &::= \text{rwr} \mid \text{rd} \mid \text{atm} \\ \text{Mld} \ni m &::= \langle \text{identifier} \rangle \quad (\text{method name}) \\ \text{Expr} \ni E &::= \mathbf{new} \ \mu \ C(\bar{v}) \mid \mathbf{let} \ C \ x = E_1 \ \mathbf{in} \ E_2 \mid \\ &\quad \mathbf{if} \ v_1 == v_2 \ \mathbf{then} \ E_3 \ \mathbf{else} \ E_4 \mid v.m(\bar{v}) \mid \\ &\quad \text{fieldref} = v \mid v \mid \text{fieldref} \mid \mathbf{throw} \ v \mid \\ &\quad \mathbf{try} \ \{E_1\} \ \mathbf{catch} \ (\mu \ C \ x) \ \{E_2\} \\ v &::= x \mid \mathbf{this} \mid \mathbf{null} \\ \text{fieldref} &::= v.x \\ A &::= C \mid \emptyset \\ \text{BCtxt} \ni \mathcal{C} &::= \llbracket \rrbracket_A \mid \mathbf{let} \ C \ x = \mathcal{C} \ \mathbf{in} \ E \mid \\ &\quad \mathbf{try} \ \{C\} \ \mathbf{catch} \ (\mu \ C \ x) \ \{E\} \end{aligned}$$

Rysunek 2.1: Składnia języka Jafun

Notacje pomocnicze dla deklaracji w $\overline{\mathbf{C}}$

Niech **class** C_1 **ext** C_2 $\{\overline{\mathbf{F}} \ \overline{\mathbf{M}}\}$ będzie deklaracją klasy w $\overline{\mathbf{C}}$. Niech $\phi \ C_3 \ x$ będzie deklaracją pola w $\overline{\mathbf{F}}$. Niech $\mu_r \ C_4 \ \mu_o \ m(\overline{\mathbf{arg}}) \ \mathbf{throws} \ \mathbf{Exc} \ \{E_1\}$ będzie deklaracją metody w $\overline{\mathbf{M}}$, gdzie $\overline{\mathbf{arg}} = \mu'_1 \ C'_1 \ x_1, \dots, \mu'_n \ C'_n \ x_n$, $\overline{\mathbf{Exc}} = \mu''_1 \ C''_1, \dots, \mu''_k \ C''_k$, and $\mu_r, \mu_o, \mu'_i, \mu''_j \in \mathbf{AMod}$ for all possible i, j . Ustalając $\overline{\mathbf{M}}_1 \cup \overline{\mathbf{M}}_2 = \overline{\mathbf{M}}_1 \cup \{\mathbf{M} \in \overline{\mathbf{M}}_2 \mid \text{name}(\mathbf{M}) \notin \overline{\mathbf{M}}_1\}$, możemy zdefiniować następujące pomocnicze notacje:

$C_1 \in \overline{\mathbf{C}}$	kiedy w deklaracja C_1 istnieje w $\overline{\mathbf{C}}$,
$\mathbf{Object} \in \overline{\mathbf{C}}, \mathbf{NPE} \in \overline{\mathbf{C}}$	
$\text{flds}(C_1) = \{x \in \text{Id} \mid \phi \ D_1 \ x \in \overline{\mathbf{F}}\} \cup \text{flds}(C_2)$	$\text{flds}(\mathbf{Object}) = \emptyset$
$\text{flds}(C_1) = \overline{\mathbf{F}}, \text{flds}(C_2)$	$\text{flds}(\mathbf{Object}) = \emptyset$
$\text{mthds}(C_1) = \overline{\mathbf{M}} \cup \text{mthds}(C_2)$	$\text{mthds}(\mathbf{Object}) = \emptyset$
$\text{ext}(C_1) = C_2$	$\text{ext}(\mathbf{Object}) = \emptyset$
$x \in C_1$	kiedy deklaracja x istnieje w $\overline{\mathbf{F}}$,
$\text{typeof}(C_1, x) = C_3$	dla $x \in C_1$
$m \in C_1$	kiedy deklaracja m istnieje w $\overline{\mathbf{M}}$,
$\text{body}(C_1, m) = E_1$,	dla $m \in C_1$,
$\text{pars}(C_1, m) = \overline{\mathbf{arg}}$	dla $m \in C_1$,
$\text{parNms}(C_1, m) = x_1, \dots, x_n$	dla $m \in C_1$,
$\text{class}(h, l)$	klasa obiektu znajdującego się pod lokacją $l \in \text{Loc}$ na stercie $h \in \text{Heap}$
$\text{class}(h, \mathbf{null}) = \perp$	dla każdej sterty h
$\text{alloc} : \text{Heap} \times \text{Prog} \times \text{Cld} \rightarrow \text{Loc} \times \text{Heap}$	funkcja alokacji
$\text{empty}_C(x) = \mathbf{null}$	dla $x \in \text{flds}(C)$

Rysunek 2.2: Notacje pomocnicze

Semantyka małych kroków języka Jafun jest zdefiniowana przez relację \rightarrow na rysunku 2.3, dla ustalonego programu . Relacja \rightarrow jest relacją binarną na parach (sterta, stos wywołań). W ogólności ma ona postać

$$\overline{\mathbf{C}}, \ h, \ C_1 \llbracket E_1 \rrbracket_{A_1} :: \dots :: C_n \llbracket E_n \rrbracket_{A_n} \rightarrow h', C'_1 \llbracket E_1 \rrbracket_{A'_1} :: \dots :: C'_m \llbracket E_m \rrbracket_{A'_m}.$$

Stos wywołań $C_1 \llbracket E_1 \rrbracket_{A_1} :: \dots :: C_n \llbracket E_n \rrbracket_{A_n}$, albo w skrócie $\overline{\mathbf{C}}$, to ciąg wyrażeń z rysunku 2.1, w którym aktualnie ewaluowane wyrażenie (redek) jest oznaczone specjalnym symbolem $\llbracket \ \rrbracket_A$. Indeks A opisuje, czy program wykonuje się w sposób normalny ($A = \emptyset$), czy był rzucony jakiś niezłapany jeszcze wyjątek ($A \in \overline{\mathbf{C}}$).

Dla wygody każda ramka stosu jest podzielona na kontekst $\mathcal{C}_i \in \mathbf{BCtxt}$ i redeks E_i . Kontekst \mathcal{C}_i opisuje wszystkie zagnieżdżone bloki **let** i **catch**, wewnątrz których znajduje się E_i .

- (newk) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{new } \mu C(l_1, \dots, l_k)]_\emptyset \rightarrow h'', \overline{C} :: \mathcal{C}[l_0]_\emptyset$
gdzie $\text{alloc}(h, \overline{C}, C) = (l_0, h')$, $\text{flds}(C) = x_1, \dots, x_k$,
 $o = \text{empty}_C\{x_1 \mapsto l_1, \dots, x_k \mapsto l_k\}$, $h'' = h'\{l_0 \mapsto o\}$
- (letin) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{let } C x = E_1 \text{ in } E_2]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[\text{let } C x = [E_1]_\emptyset \text{ in } E_2]$
(letgo) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{let } C x = [l]_\emptyset \text{ in } E] \rightarrow h, \overline{C} :: \mathcal{C}[E\{l/x\}]_\emptyset$
- (ifeq) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{if } l_0 == l_1 \text{ then } E_1 \text{ else } E_2]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[E_1]_\emptyset$ gdzie $l_0 = l_1$
(ifneq) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{if } l_0 == l_1 \text{ then } E_1 \text{ else } E_2]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[E_2]_\emptyset$ gdzie $l_0 \neq l_1$
- (mthdnpe) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{null.m}(\bar{l})]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[\text{npe}]_{\text{NPE}}$
(mthd) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{l.m}(\bar{l})]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[\text{l.m}(\bar{l})]_\emptyset :: [E]_\emptyset$
gdzie $\text{class}(h, l) = D$, $\text{body}(D, m) = E_0$, $E = E_0\{l/\text{this}, \bar{l}/\text{parNms}(D, m)\}$
- (mthdret) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{l.m}(\bar{l})]_\emptyset :: [l']_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[l']_\emptyset$
- (assignnpe) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{null.x} = l]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[\text{npe}]_{\text{NPE}}$
(assignev) $\overline{C}, h, \overline{C} :: \mathcal{C}[l_1.x = l]_\emptyset \rightarrow h', \overline{C} :: \mathcal{C}[l]_\emptyset$
gdzie $l_1 \neq \text{null}$, $o = h(l_1)\{x \mapsto l\}$, $h' = h\{l_1 \mapsto o\}$
- (varnpe) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{null.x}]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[\text{npe}]_{\text{NPE}}$
(var) $\overline{C}, h, \overline{C} :: \mathcal{C}[l.x]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[l']_\emptyset$ gdzie $l \neq \text{null}$, $l' = h(l)(x)$
- (thrownull) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{throw null}]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[\text{npe}]_{\text{NPE}}$
- (throw) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{throw } l]_\emptyset \rightarrow h, \overline{C} :: \mathcal{C}[l]_D$ gdzie $l \neq \text{null}$, $\text{class}(h, l) = D$
- (ctchin) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{try } \{E_1\} \text{ catch } (\mu C x) \{E_2\}]_\emptyset \rightarrow$
 $h, \overline{C} :: \mathcal{C}[\text{try } \{[E_1]_\emptyset\} \text{ catch } (\mu C x) \{E_2\}]$
- (ctchnrml) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{try } \{[l]_\emptyset\} \text{ catch } (\mu C x) \{E_2\}] \rightarrow h, \overline{C} :: \mathcal{C}[l]_\emptyset$
(ctchexok) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{try } \{[l]_{C'}\} \text{ catch } (\mu C x) \{E_2\}] \rightarrow h, \overline{C} :: \mathcal{C}[E'_2]_\emptyset$
gdzie $E'_2 = E_2\{l/x\}$, $C' \leq C$
- (letex) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{let } C x = [l]_{C'} \text{ in } E] \rightarrow h, \overline{C} :: \mathcal{C}[l]_{C'}$ gdzie $C' \neq \emptyset$
(methodex) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{l.m}(\bar{l})]_\emptyset :: [l']_{C'} \rightarrow h, \overline{C} :: \mathcal{C}[l']_{C'}$ gdzie $C' \neq \emptyset$
(ctchexnok) $\overline{C}, h, \overline{C} :: \mathcal{C}[\text{try } \{[l]_{C'}\} \text{ catch } (\mu C x) \{E_2\}] \rightarrow h, \overline{C} :: \mathcal{C}[l]_{C'}$
gdzie $C' \neq \emptyset$, $C' \not\leq C$

Rysunek 2.3: Semantyka języka Jafun

Ewaluacja programu zaczyna się od stanu $\overline{C}, h, [E']_\emptyset$ gdzie $h \in \text{Heap}$, $E' = E\{l_o/\text{this}\}$, $\text{class}(h, l_o) = C$, a $C' m() \text{ throws NPE } \{E\}$ jest metodą nieprzyjmującą argumentów w klasie C . Metoda m odpowiada funkcji `main` w zwykłej Javie. Dodatkowo zakładamy, że na stercie h , pod pewną lokacją `npe` istnieje obiekt klasy `NPE` (“null pointer exception”).

2.2. Ewaluacja

Częściową ewaluacją konfiguracji (h, \overline{C}) będziemy nazywać dowolny ciąg par $\text{confs} = (h_1, \overline{C}_1), \dots, (h_n, \overline{C}_n)$, taki że $h_1 = h$, $\overline{C}_1 = \overline{C}$ oraz $(h_i, \overline{C}_i) \rightarrow (h_{i+1}, \overline{C}_{i+1})$ dla $1 \leq i < n$. Częściowe ewaluacje będziemy oznaczać jako $(h_1, \overline{C}_1) \xrightarrow{\text{confs}} (h_n, \overline{C}_n)$.

Ewaluacją wyrażenia e na stercie h będziemy nazywać taką ewaluację konfiguracji $(h, [e]_\emptyset)$,

że $\bar{\mathcal{C}}_n = \llbracket l \rrbracket_A$ dla pewnych l, A . Jeśli taka ewaluacja istnieje, będziemy to oznaczać jako $(h, e) \overset{conf}{\rightsquigarrow} (h_n, A, l)$ lub

Rozdział 3

Składnia i semantyka

Prezentowana logika separacji dla języka Jafun jest logiką z kwantyfikatorami egzystencjalnymi pierwszego rzędu, trójkami Hoare’a, operatorem \hookrightarrow , pozwalającym na opisywanie wartości sterty i operatorami separacji $*$ i \multimap .

Iris, na którym wzorowana jest niniejsza logika, jest afiniczną logiką separacyjną, to znaczy własność spełniania termu przez stertę jest domknięta ze względu na rozszerzanie sterty. W celu zachowania zarówno afiniczności, jak i poprawności względem semantyki języka, prezentowana logika nie zawiera kwantyfikatora ogólnego, a kwantyfikator egzystencjalny jest ograniczony do termów najwyższego poziomu (Rysunek 3.1).

Używane będzie także oznaczenie $v_1 \neq v_2$ jako skrót dla $v_1 = v_2 \Rightarrow \text{False}$.

$$\begin{aligned} \mathbf{P} &::= \exists x : C. \mathbf{P} \mid \mathbf{P} \wedge \mathbf{P} \mid \mathbf{P} \vee \mathbf{P} \mid P \\ P &::= \text{True} \mid \text{False} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid v = v \mid \\ &\quad v \hookrightarrow x = v \mid \{P\}E\{x.P\}_A \mid P * P \mid P \multimap P \\ v &::= x \mid \text{null} \mid \text{this} \\ A &::= C \mid \phi \\ x &::= \langle \text{identifier} \rangle \text{ (variable/field name)} \\ C &::= \langle \text{identifier} \rangle \text{ (class name)} \\ e &::= \langle \text{Jafun expression} \rangle \end{aligned}$$

Rysunek 3.1: Składnia logiki

Środowisko to funkcja częściowa przypisująca identyfikatorom lokacje na stercie lub `null`. Semantyka logiki (Rysunek 3.2) jest standardowa dla kwantyfikatora i operatorów logicznych. Dla uproszczenia zapisu notacja $\llbracket \cdot \rrbracket$ została użyta do opisu semantyki obu poziomów termów (\mathbf{P} i P). To, do którego poziomu się odnosi, wynika z kontekstu.

Sterta spełnia trójkę Hoare’a $\{P\}E\{x.Q\}_A$, jeśli dla każdej sterty spełniającej P , wyrażenie E zostanie obliczone bez błędu, zwróci wyjątek typu A (czyli być może żaden), a wynikowa sterta będzie spełniała Q , w którym za x podstawiony zostanie wynik obliczenia.

Sterta spełnia term $P * Q$, jeśli można ją podzielić na dwa rozłączne fragmenty, z których jeden spełnia P , a drugi Q . Operator \multimap to pewnego rodzaju odwrotność operatora $*$ – sterta spełnia $P \multimap Q$, jeśli po połączeniu jej z dowolną rozłączną stertą spełniającą P , otrzymana sterta spełnia Q .

$$\begin{aligned}
\llbracket \text{True} \rrbracket &\triangleq \top \\
\llbracket \text{False} \rrbracket &\triangleq \perp \\
\llbracket \exists x : C.P \rrbracket_{h,env} &\triangleq \exists l : \text{Loc} . \text{class}(h, l) = C \wedge \llbracket P \rrbracket_{h,env[x \mapsto l]} \\
\llbracket P \wedge Q \rrbracket_{h,env} &\triangleq \llbracket P \rrbracket_{h,env} \wedge \llbracket Q \rrbracket_{h,env} \\
\llbracket P \vee Q \rrbracket_{h,env} &\triangleq \llbracket P \rrbracket_{h,env} \vee \llbracket Q \rrbracket_{h,env} \\
\llbracket P \Rightarrow Q \rrbracket_{h,env} &\triangleq \llbracket P \rrbracket_{h,env} \Rightarrow \llbracket Q \rrbracket_{h,env} \\
\llbracket x = y \rrbracket_{h,env} &\triangleq \text{env}(x) = \text{env}(y) \\
\llbracket x \hookrightarrow f = y \rrbracket_{h,env} &\triangleq h(\text{env}(x))(f) = \text{env}(y) \\
\llbracket \{P\}E\{x.Q\}_A \rrbracket_{h,env} &\triangleq \forall h : \text{Heap} . \llbracket P \rrbracket_{h,env} \Rightarrow \\
&\quad \exists h' : \text{Heap}, l : \text{Loc} . (h, E[/env]) \rightsquigarrow (h', A, l) \wedge \llbracket Q \rrbracket_{h',env[x \mapsto l]} \\
\llbracket P * Q \rrbracket_{h,env} &\triangleq \exists h_1, h_2 : \text{Heap} . h_1 \oplus h_2 = h \wedge \llbracket P \rrbracket_{h_1,env} \wedge \llbracket Q \rrbracket_{h_2,env} \\
\llbracket P \multimap Q \rrbracket_{h,env} &\triangleq \forall h' : \text{Heap} . \llbracket P \rrbracket_{h',env} \Rightarrow \llbracket Q \rrbracket_{h \oplus h',env}
\end{aligned}$$

Uwaga: $E[/math>/ env] oznacza wyrażenie powstałe przez podstawienie $env(x)$ w miejsce x dla każdej zmiennej wolnej x w E .$

Rysunek 3.2: Semantyka logiki

Rozdział 4

Reguły wnioskowania

Osądy w prezentowanej logice są postaci $\Gamma \mid P \vdash Q$, gdzie Γ to środowisko typów, przypisujące zmiennym odpowiadające im typy (czyli nazwy klas), a P i Q to termy logiki. Intuicyjnie, osąd $\Gamma \mid P \vdash Q$ oznacza że Q wynika z P , a więc że każda sterta spełniająca P spełnia też Q .

Dla poprawienia czytelności, jeśli Γ jest wspólne dla wszystkich osądów występujących w danej regule, to jest ono pomijane.

$$\begin{array}{c}
 \text{ASM} \frac{}{P \vdash P} \quad \text{TRANS} \frac{P \vdash Q \quad Q \vdash R}{P \vdash R} \quad \text{EQ-REFL} \frac{}{P \vdash v = v} \quad \text{EQ-SYM} \frac{P \vdash v = w}{P \vdash w = v} \\
 \\
 \perp\text{E} \frac{P \vdash \text{False}}{P \vdash Q} \quad \top\text{I} \frac{}{P \vdash \text{True}} \quad \wedge\text{I} \frac{R \vdash P \quad R \vdash Q}{R \vdash P \wedge Q} \quad \wedge\text{EL} \frac{R \vdash P \wedge Q}{R \vdash P} \\
 \\
 \wedge\text{ER} \frac{R \vdash P \wedge Q}{R \vdash Q} \quad \vee\text{IL} \frac{R \vdash P}{R \vdash P \vee Q} \quad \vee\text{IR} \frac{R \vdash Q}{R \vdash P \vee Q} \\
 \\
 \vee\text{E} \frac{S \vdash P \vee Q \quad P \vdash R \quad Q \vdash R}{S \vdash R} \quad \Rightarrow\text{I} \frac{R \wedge P \vdash Q}{R \vdash P \Rightarrow Q} \quad \Rightarrow\text{E} \frac{R \vdash P \Rightarrow Q \quad R \vdash P}{R \vdash Q} \\
 \\
 \exists\text{I} \frac{\Gamma, x : C \mid Q \vdash P[t/x]}{Q \vdash \exists x : C.P} \quad \exists\text{E} \frac{\Gamma \mid R \vdash \exists x : C.P \quad \Gamma, x : C \mid R \wedge P \vdash Q}{\Gamma \mid R \vdash Q}
 \end{array}$$

Rysunek 4.1: Reguły wnioskowania dla tradycyjnych operatorów logicznych

$$\begin{array}{c}
 \text{WEAK} \frac{}{P * Q \vdash P} \quad \text{SEP-ASSOC} \frac{}{P * (Q * R) \dashv\vdash (P * Q) * R} \quad \text{SEP-SYM} \frac{}{P * Q \vdash Q * P} \\
 \\
 *\text{I} \frac{P_1 \vdash Q_1 \quad P_2 \vdash Q_2}{P_1 * Q_1 \vdash P_2 * Q_2} \quad *\text{I} \frac{R * P \vdash Q}{R \vdash P * Q} \quad *\text{E} \frac{R_1 \vdash P * Q \quad R_2 \vdash P}{R_1 * R_2 \vdash Q}
 \end{array}$$

Rysunek 4.2: Reguły wnioskowania dla operatorów separacyjnych

Reguły strukturalne dla trójek Hoare'a

$$\begin{array}{c}
\text{HT-FRAME} \frac{S \vdash \{P\}E\{v.Q\}_A \quad S \text{ jest trwały}}{S \vdash \{P * R\}E\{v.Q * R\}_A} \quad \text{HT-RET} \frac{}{S \vdash \{\text{True}\}w\{v.v = w\}_\phi} \\
\\
\text{HT-CSQ} \frac{\Gamma | S \vdash P \Rightarrow P' \quad \Gamma | S \vdash \{P'\}E\{v.Q'\}_A \quad \Gamma, v : C | S \vdash Q' \Rightarrow Q \quad S \text{ jest trwały}}{S \vdash \{P\}E\{v.Q\}_A} \\
\\
\text{HT-DISJ} \frac{S \vdash \{P\}E\{v.Q\}_A \quad S \vdash \{Q\}E\{v.Q\}_A}{S \vdash \{P \vee Q\}E\{v.Q\}_A} \\
\\
\text{HT-PERS} \frac{S \wedge R \vdash \{Q\}E\{v.Q\}_A}{S \vdash \{Q \wedge R\}E\{v.Q\}_A} \text{jeśli } R \text{ trwały}
\end{array}$$

Reguły dla trójek Hoare'a opisujących konstrukcje języka

$$\begin{array}{c}
\text{HT-NEW-NULL} \frac{}{S \vdash \{\text{True}\}\mathbf{new} \ C(\bar{v})\{w.w \neq \mathbf{null}\}_\phi} \\
\\
\text{HT-NEW-FIELD} \frac{\text{flds}(C) = f_1, \dots, f_n}{S \vdash \{\text{True}\}\mathbf{new} \ C(v_1, \dots, v_n)\{w.w \hookrightarrow f_i = v_i\}_\phi} \\
\\
\text{HT-LET} \frac{\Gamma | S \vdash \{P\}E_1\{x.Q\}_\phi \quad \Gamma, x : C | S \vdash \{Q\}E_2\{w.R\}_A}{\Gamma | S \vdash \{P\}\mathbf{let} \ C \ x = E_1 \ \mathbf{in} \ E_2\{w.R\}_A} \text{jeśli } S \text{ trwały} \\
\\
\text{HT-LET-EX} \frac{S \vdash \{P\}E_1\{w.Q\}_A \quad A \neq \phi}{\Gamma | S \vdash \{P\}\mathbf{let} \ C \ x = E_1 \ \mathbf{in} \ E_2\{w.Q\}_A} \\
\\
\text{HT-FIELD-SET} \frac{}{S \vdash \{x \neq \mathbf{null}\}x.f = v\{ _ .x \hookrightarrow f = v \}_\phi} \\
\\
\text{HT-NULL-SET} \frac{}{S \vdash \{x = \mathbf{null}\}x.f = v\{w.w = \mathbf{npe}\}_{\text{NPE}}} \\
\\
\text{HT-FIELD-GET} \frac{}{S \vdash \{x \hookrightarrow f = v\}x.f\{w.w = v\}_\phi} \\
\\
\text{HT-NULL-GET} \frac{}{S \vdash \{x = \mathbf{null}\}x.f\{w.w = \mathbf{npe}\}_{\text{NPE}}}
\end{array}$$

Rysunek 4.3: Reguły wnioskowania dla trójek Hoare'a

$$\begin{array}{c}
\text{HT-IF} \frac{S \vdash \{P \wedge v_1 = v_2\} E_1 \{w.Q\}_A \quad S \vdash \{P \wedge v_1 \neq v_2\} E_2 \{w.Q\}_A}{S \vdash \{P\} \mathbf{if} \ v_1 = v_2 \ \mathbf{then} \ E_1 \ \mathbf{else} \ E_2 \{w.Q\}_A} \\
\\
\text{HT-INVOKE} \frac{\Gamma \vdash x : C \quad \{P'\} \cdot \{w.Q'\}_A \in \text{invariants}(C, m) \quad S \wedge \{P'\} x.m(\bar{v}) \{w.Q'\}_A \vdash \{P\} x.m(\bar{v}) \{w.Q\}_A}{S \vdash \{P\} x.m(\bar{v}) \{w.Q\}_A} \\
\\
\text{HT-NUL-VOKE} \frac{}{S \vdash \{x = \text{null}\} x.m(\bar{v}) \{w.w = \text{npe}\}_{\text{NPE}}} \\
\\
\text{HT-THROW} \frac{\Gamma \vdash x : C}{S \vdash \{x \neq \text{null}\} \mathbf{throw} \ x \{w.w = x\}_C} \\
\\
\text{HT-NUL-THROW} \frac{}{S \vdash \{x = \text{null}\} \mathbf{throw} \ x \{w.w = \text{npe}\}_{\text{NPE}}} \\
\\
\text{HT-CATCH-NORMAL} \frac{S \vdash \{P\} E_1 \{w.Q\}_\phi}{S \vdash \{P\} \mathbf{try} \ E_1 \ \mathbf{catch} \ (C \ x) \ E_2 \{w.Q\}_\phi} \\
\\
\text{HT-CATCH-EX} \frac{\Gamma | S \vdash \{P\} E_1 \{x.Q\}'_C \quad \Gamma, x : C' | S \vdash \{Q\} E_2 \{w.R\}_A \quad C' \leq C}{\Gamma | S \vdash \{P\} \mathbf{try} \ E_1 \ \mathbf{catch} \ (C \ x) \ E_2 \{w.R\}_A} \text{jeśli } S \text{ trwały} \\
\\
\text{HT-CATCH-PASS} \frac{S \vdash \{P\} E_1 \{w.Q\}'_C \quad C' \not\leq C}{S \vdash \{P\} \mathbf{try} \ E_1 \ \mathbf{catch} \ (C \ x) \ E_2 \{w.Q\}'_C}
\end{array}$$

Rysunek 4.4: Reguły wnioskowania dla trójek Hoare'a - c.d.

Rozdział 5

Własności stert

5.1. Definicje

Definicja 5.1.1 (Spójność sterty).

Niech h będzie stertą. Powiemy, że h jest spójna, jeśli każda lokacja będąca wartością pola w pewnym obiekcie na h również jest na h . To znaczy, dla każdego x, l_1, l_2 , jeśli $h(l_1)(x) = l_2$, to $l_2 \in \text{Dom}(h)$. \square

Definicja 5.1.2 (Suma rozłączna stert).

Niech h, h_1, h_2 będą stertami. Powiemy, że h jest sumą rozłączną stert h_1 i h_2 (zapisywane $h = h_1 \oplus h_2$), jeśli

1. Dla każdej lokacji l , $l \in h$ wtedy i tylko wtedy gdy $l \in h_1$ lub $l \in h_2$
2. Nie istnieje lokacja l , taka że $l \in h_1$ i $l \in h_2$

\square

Rozdział 6

Własności ewaluacji

Pokażę teraz twierdzenia o własności ewaluacji, które będą później użyte do udowodnienia poprawności reguł dla trójek Hoare’a.

6.1. Łączenie ewaluacji

Podatwowym twierdzeniem, pozwalającym mówić o ewaluacji złożonych wyrażeń, jest twierdzenie o łączeniu ewaluacji.

Twierdzenie 1 (O łączeniu ewaluacji). *Niech $(h, \bar{C}), (h', \bar{C}'), (h'', \bar{C}'')$ będą konfiguracjami, a $conf_s$ i $conf_{s'}$ – ciągami konfiguracji, takimi że $(h, \bar{C}) \xrightarrow{conf_s} (h', \bar{C}')$ i $(h', \bar{C}') \xrightarrow{conf_{s'}} (h'', \bar{C}'')$. Wtedy $(h, \bar{C}) \xrightarrow{conf_s ++ conf_{s'}} (h'', \bar{C}'')$.*

Dowód. TODO

□

6.2. Ewaluacja przy rozszerzonym stosie i kontekście

6.3. Ewaluacja zależy tylko od zmiennych wolnych

Twierdzenie o zależności ewaluacji od zmiennych wolnych jest kluczowe w dowodzie poprawności dla reguł WEAK i HT-FRAME. Mówi ono, że jeśli dwie sterty zgadzają się na lokacjach odpowiadających zmiennym wolnym w pewnym wyrażeniu E , to ewaluacje wyrażenia E na tych dwóch stertach będą w pewnym sensie równoważne.

Równoważność ta nie będzie niestety trywialna, bo nowo zaalokowane lokacje na obu sterach mogą się różnić. Zgodnie z semantyką języka, lokacja zwracana przez operator **new** to (maximum z lokacji na sterze) + 1. Stąd, ponieważ nie zakładamy niczego o lokacjach innych niż te odpowiadające zmiennym wolnym, wartość zwracana przez operator **new** może się różnić pomiędzy stertami. Nowo zaalokowane lokacje mogą następnie zostać zapisane w polach obiektów znajdujących się pod lokacjami odpowiadającymi zmiennym wolnym, co oznacza że nawet te obiekty, początkowo równe na obu stertach, mogą zacząć się różnić w czasie ewaluacji.

6.3.1. Izomorfizmy

Żeby obejść ten problem, zdefiniujemy *izomorfizm stert* jako bijekcję między lokacjami na tych stertach, zachowującą null, npe i kompozycję.

Definicja 6.3.1 (izomorfizm stert).

Niech $h_1, h_2 : \text{Heap}$. Funkcję $f : \text{Dom}(h_1) \cup \{\text{null}\} \rightarrow \text{Dom}(h_2) \cup \{\text{null}\}$ nazwiemy izomorfizmem między tymi stertami, jeśli:

1. f jest bijekcją
2. $f(\text{null}) = \text{null}$
3. $f(\text{npe}) = \text{npe}$
4. f zachowuje kompozycję, to znaczy dla dowolnych lokacji l_1, l_2 i pola x zachodzi

$$h_1(l_1) \hookrightarrow x = l_2 \iff h_2(f(l_2)) \hookrightarrow x = f(l_2)$$

Jeśli taka funkcja f istnieje, powiemy że sterty h_1 i h_2 są izomorficzne. □

Ostatecznie będziemy chcieli pokazać, że jeśli wyrażenie E nie zawiera zmiennych wolnych, a sterty h_1, h_2 są równe na wszystkich lokacjach występujących w E , to ewaluacje E na stertach h_1 i h_2 są równoważne z dokładnością do izomorfizmu.

To oznacza, że potrzebujemy mówić o izomorfizmach ewaluacji (czyli ciągów par (sterta, stos wywołań)), a zatem należy zdefiniować także izomorfizmy między stosami wywołań. Służy temu kolejnych kilka definicji.

Definicja 6.3.2 (izomorfizm wyrażeń).

Intuicyjnie, dwa wyrażenia są izomorficzne, jeśli różnią się tylko lokacjami w nich występującymi i istnieje izomorfizm stert, mapujący lokacje z pierwszego z wyrażeń na odpowiadające im lokacje w drugim. Formalnie zdefiniujemy ten izomorfizm przez indukcję po budowie wyrażeń.

Niech f będzie izomorfizmem między dwiema stertami i niech E_1, E_2 będą wyrażeniami Jafun. Powiemy, że f jest izomorfizmem między tymi wyrażeniami, jeśli

1. E_1 i E_2 są wyrażeniami tego samego rodzaju (np. oba są wyrażeniami **new**)
2. f jest izomorfizmem między odpowiadającymi sobie podwyrażeniami E_1 i E_2
3. Dla każdego **this** występującego w E_1 , na odpowiadającej mu pozycji w E_2 też jest **this**
4. Dla każdego identyfikatora występującego w E_1 , na odpowiadającej mu pozycji w E_2 jest taki sam identyfikator
5. Dla każdej lokacji l w E_1 , na odpowiadającej mu pozycji w E_2 jest $f(l)$

□

Definicja 6.3.3 (izomorfizm kontekstów).

Podobnie jak wyżej, definiujemy izomorfizm kontekstów przez indukcję po budowie kontekstu.

Niech f będzie izomorfizmem między dwiema stertami i niech C_1, C_2 będą kontekstami ewaluacji Jafun. Powiemy, że f jest izomorfizmem między tymi kontekstami, jeśli

- $C_1 = C_2 = \llbracket \rrbracket_A$

lub

- $C_1 = \text{let } C \ x = C'_1 \text{ in } E_1, \quad C_2 = \text{let } C \ x = C'_2 \text{ in } E_2,$
a f jest izomorfizmem między C'_1 i C'_2 oraz między E_1 i E_2
- lub
- $C_1 = \text{try } \{C'_1\} \text{ catch } (\mu \ C \ x) \ \{E_1\}, \quad C_2 = \text{try } \{C'_2\} \text{ catch } (\mu \ C \ x) \ \{E_2\},$
a f jest izomorfizmem między C'_1 i C'_2 oraz między E_1 i E_2

□

Definicja 6.3.4 (izomorfizm stosów wywołań).

f jest izomorfizmem między dwoma stosami wywołań, jeśli są one równej długości i f jest izomorfizmem między każdą parą odpowiadających sobie kontekstów i redeksów z tych stosów.

□

6.3.2. Zależność ewaluacji od zmiennych wolnych

Sformułuję teraz twierdzenie o zależności ewaluacji od zmiennych wolnych, a następnie udowodnię kilka lematów pomocnych w jego dowodzie.

Twierdzenie 2 (O zależności ewaluacji od zmiennych wolnych).

Niech h będzie spójną stertą, env środowiskiem, a E wyrażeniem Jafun, w którym nie występują konkretne lokacje, a wszystkie zmienne wolne są przez env mapowane na lokacje w h . Niech $h_1, h_2, h_1^{rest}, h_2^{rest}$ będą stertami takimi, że $h_1 = h \oplus h_1^{rest}$ i $h_2 = h \oplus h_2^{rest}$. Wreszcie, niech $conf_{s1}, h_{1,n}, A, l_1$ będą takie, że $(h_1, E[/env]) \xrightarrow{conf_{s1}} (h_{1,n}, A, l_1)$.

Wtedy istnieją $h_{1,n}^{base}, h_{2,n}^{base}, conf_{s2}, h_{2,n}, l_2, f$, takie że

1. $h_{1,n} = h_{1,n}^{base} \oplus h_1^{rest}$
2. $h_{2,n} = h_{2,n}^{base} \oplus h_2^{rest}$
3. f jest izomorfizmem między $h_{1,n}^{base}$ i $h_{2,n}^{base}$
4. f jest identycznością na lokacjach w env
5. $f(l_1) = l_2$
6. $(h_2, E[/env]) \xrightarrow{conf_{s2}} (h_{2,n}, A, l_2)$.

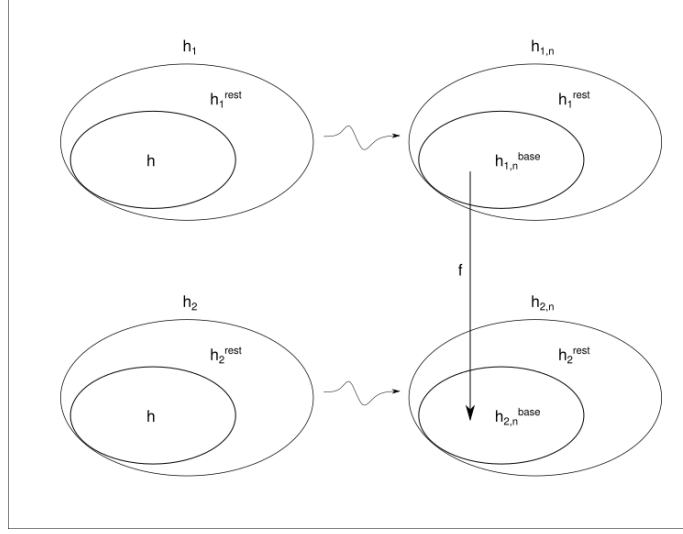
Dowód. Na końcu sekcji.

□

Sterty h_1 i h_2 w powyższym twierdzeniu to dwie sterety, o których wiemy, że zgadzają się na wszystkich lokacjach odpowiadających zmiennym wolnym w E (wszystkie one zawierają się w podstercie h). O ich pozostałych fragmentach (odpowiednio h_1^{rest} i h_2^{rest}) nie zakładamy nic. Twierdzenie mówi, że jeśli mamy ewaluację wyrażenia E w środowisku env na stercie h_1 , to istnieje też jego analogiczna ewaluacja na sterzie h_2 , taka że sterety docelowe oraz wyniki ewaluacji są izomorficzne, a fragmenty stert nie mające związku ze zmiennymi wolnymi pozostają niezmiennione (Rysunek 6.1).

W praktyce oznacza to, że jedynie fragmenty stert odpowiadające zmiennym wolnym w wyrażeniu mają znaczenie dla ewaluacji tego wyrażenia.

Dowód twierdzenia będzie przebiegał przez indukcję po długości ewaluacji $conf_{s1}$. W tym celu jednak musimy sformułować następujący lemat, będący krokiem indukcyjnym w uogólnieniu powyższego twierdzenia na częściowe ewaluacje.



Rysunek 6.1: Wizualizacja twierdzenia o zależności ewaluacji od zmiennych wolnych

Lemat 1 (O zależności redukcji od zmiennych wolnych).

Niech $\bar{\mathcal{C}}_1$ będzie dowolnym stosiem wywołań, a $h_1, h_1^{base}, h_1^{rest}$ będą stertami, takimi że h_1^{base} jest spójna, $h_1 = h_1^{base} \oplus h_1^{rest}$, a wszystkie lokacje występujące w wyrażeniach na stosie $\bar{\mathcal{C}}_1$ znajdują się na sterzie h_1^{base} . Niech teraz $h_{1,n}, \bar{\mathcal{C}}_{1,n}$ będą takie, że $(h_1, \bar{\mathcal{C}}_1) \rightarrow (h_{1,n}, \bar{\mathcal{C}}_{1,n})$.

Weźmy teraz dowolną spójną stertę h_2^{base} , stos wywołań $\bar{\mathcal{C}}_2$ oraz izomorfizm f , takie że f jest izomorfizmem między h_1^{base} i h_2^{base} oraz między $\bar{\mathcal{C}}_1$ i $\bar{\mathcal{C}}_2$, zdefiniowanym tylko na lokacjach znajdujących się w h_1^{base} . Jeśli wszystkie lokacje występujące w wyrażeniach na stosie $\bar{\mathcal{C}}_2$ znajdują się na sterze h_2^{base} , to dla dowolnych stert h_2, h_2^{rest} , takich że $h_2 = h_2^{base} \oplus h_2^{rest}$ istnieją sterty $h_{1,n}^{base}, h_{2,n}^{base}, h_{2,n}$, stos $\bar{\mathcal{C}}_{2,n}$ oraz izomorfizm f' , takie że

1. f' rozszerza f
2. f' jest zdefiniowane tylko na lokacjach znajdujących się w $h_{1,n}^{base}$
3. f' jest izomorfizmem między $h_{1,n}^{base}$ i $h_{2,n}^{base}$ oraz między $\bar{\mathcal{C}}_{1,n}$ i $\bar{\mathcal{C}}_{2,n}$
4. $h_{1,n} = h_{1,n}^{base} \oplus h_1^{rest}$
5. $h_{2,n} = h_{2,n}^{base} \oplus h_2^{rest}$
6. wszystkie lokacje występujące w wyrażeniach na stosie $\bar{\mathcal{C}}_{1,n}$ znajdują się na sterze $h_{1,n}^{base}$
7. wszystkie lokacje występujące w wyrażeniach na stosie $\bar{\mathcal{C}}_{2,n}$ znajdują się na sterze $h_{2,n}^{base}$
8. $(h_2, \bar{\mathcal{C}}_2) \rightarrow (h_{2,n}, \bar{\mathcal{C}}_{2,n})$.

Dowód. Rozpatrzmy wszystkie możliwe postaci izomorficznych stosów $\bar{\mathcal{C}}_1, \bar{\mathcal{C}}_2$, takich że istnieje redukcja $(h_1, \bar{\mathcal{C}}_1) \rightarrow (h_{1,n}, \bar{\mathcal{C}}_{1,n})$.

$$\bullet \quad \bar{\mathcal{C}}_1 = \bar{\mathcal{C}}'_1 :: \mathcal{C}_1[\text{new } \mu C(l_1, \dots, l_k)]_\emptyset, \quad \bar{\mathcal{C}}_2 = \bar{\mathcal{C}}'_2 :: \mathcal{C}_2[\text{new } \mu C(l'_1, \dots, l'_k)]_\emptyset$$

Redukcja $\bar{\mathcal{C}}_1$ jest postaci $(h_1, \bar{\mathcal{C}}'_1 :: \mathcal{C}_1[\text{new } \mu C(l_1, \dots, l_k)]_\emptyset) \rightarrow (h_{1,n}, \bar{\mathcal{C}}'_1 :: \mathcal{C}_1[l_0]_\emptyset)$,

gdzie $\text{alloc}(h_1, \bar{\mathcal{C}}'_1, C) = (l_0, h)$, $\text{flds}(C) = x_1, \dots, x_k$, $o = \text{empty}_C\{x_1 \mapsto l_1, \dots, x_k \mapsto l_k\}$, $h_{1,n} =$

$h\{l_0 \mapsto o\}$. Nowa sterta $h_{1,n}$ jest zatem po prostu stertą h_1 z dodatkowym nowym obiektem o pod nową lokacją l_0 . Weźmy zatem $h_{1,n}^{base} = h_1^{base}\{l_0 \mapsto o\}$. Spełnia ona warunki 4 i 6, ponieważ jedyną nową lokacją na stosie jest l_0 , które trafiło właśnie do $h_{1,n}^{base}$.

Weźmy teraz $(l'_0, h') = \text{alloc}(h_2, \bar{C}'_2, C)$, $o' = \text{empty}_C\{x_1 \mapsto l'_1, \dots, x_k \mapsto l'_k\}$, $h_{2,n} = h'\{l'_0 \mapsto o'\}$. Mamy wtedy $(h_2, \bar{C}_2) \rightarrow (h_{2,n}, \bar{C}'_2 :: C_2\llbracket l'_0 \rrbracket_\emptyset)$, a zatem biorąc takie $h_{2,n}$ i $\bar{C}_{2,n} = \bar{C}'_2 :: C_2\llbracket l'_0 \rrbracket_\emptyset$ mamy spełnione też warunki 7 i 8.

Jest to jedyny przypadek, w którym na sterwie pojawia się nowa lokacja, a zatem należy zmodyfikować izomorfizm f . Niech więc $h_{2,n}^{base} = h_2^{base}\{l'_0 \mapsto o'\}$ i $f' = f\{l_0 \mapsto l'_0\}$. Warunki 1 i 5 są w oczywisty sposób spełnione. Warunek 2 jest spełniony, ponieważ z założenia f było zdefiniowane tylko na lokacjach z h_1^{base} , a l_0 trafiło do $h_{1,n}^{base}$. Wreszcie, warunek 3 jest spełniony, ponieważ z założenia o izomorfizmie \bar{C}_1 i \bar{C}_2 , dla $j = 1, \dots, k$ mamy $f'(l_j) = l'_j$, a z definicji f' , $f'(l_0) = l'_0$.

- $\bar{C}_1 = \bar{C}'_1 :: C_1\llbracket \text{let } C \ x = e_1 \text{ in } e_2 \rrbracket_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2\llbracket \text{let } C \ x = e'_1 \text{ in } e'_2 \rrbracket_\emptyset$
Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1\llbracket \text{let } C \ x = e_1 \text{ in } e_2 \rrbracket_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1\llbracket e_1 \rrbracket_\emptyset \text{ in } e_2)$, a zatem jedyne co się w niej dzieje to dodanie e_2 do kontekstu i zmiana redeksu na e_1 .
Możemy więc wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2\llbracket \text{let } C \ x = \llbracket e'_1 \rrbracket_\emptyset \text{ in } e'_2 \rrbracket_\emptyset$, a sterty i izomorfizm f pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1\llbracket \text{let } C \ x = \llbracket l \rrbracket_\emptyset \text{ in } e \rrbracket_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2\llbracket \text{let } C \ x = \llbracket l' \rrbracket_\emptyset \text{ in } e' \rrbracket_\emptyset$
Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1\llbracket \text{let } C \ x = \llbracket l \rrbracket_\emptyset \text{ in } e \rrbracket_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1\llbracket e\{l/x\} \rrbracket_\emptyset)$. Podobnie jak wyżej, możemy pozostawić sterty oraz izomorfizm bez zmian i przyjąć $\bar{C}_{2,n} = (h_2, \bar{C}'_2 :: C_2\llbracket e'\{l'/x\} \rrbracket_\emptyset)$.
Wyrażenia $e\{l/x\}$ i $e'\{l'/x\}$ są izomorficzne, ponieważ z założenia f jest izomorfizmem między e i e' oraz $f(l) = l'$.
- $\bar{C}_1 = \bar{C}'_1 :: C_1\llbracket \text{if } l_0 == l_1 \text{ then } e_1 \text{ else } e_2 \rrbracket_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2\llbracket \text{if } l'_0 == l'_1 \text{ then } e'_1 \text{ else } e'_2 \rrbracket_\emptyset$
Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1\llbracket \text{if } l_0 == l_1 \text{ then } e_1 \text{ else } e_2 \rrbracket_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1\llbracket e_i \rrbracket_\emptyset)$, gdzie $i \in \{1, 2\}$ w zależności od tego czy $l_0 = l_1$.
Ponieważ $l'_0 = f(l_0)$ i $l'_1 = f(l_1)$, więc $l'_0 = l'_1$ wtedy i tylko wtedy gdy $l_0 = l_1$. Redukcja \bar{C}_2 zatem wybierze tę samą gałąź, co redukcja \bar{C}_1 . Możemy więc wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2\llbracket e'_i \rrbracket_\emptyset$, a sterty i izomorfizm f pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1\llbracket l.m(\bar{l}) \rrbracket_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2\llbracket l'.m(\bar{l}') \rrbracket_\emptyset$, $l \neq \text{null}$
Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1\llbracket l.m(\bar{l}) \rrbracket_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1\llbracket l.m(\bar{l}) \rrbracket_\emptyset :: \llbracket e \rrbracket_\emptyset)$, gdzie e jest ciałem metody m z wartościami \bar{l} podstawionymi w miejsce argumentów.
Niech teraz e' będzie ciałem metody m z wartościami \bar{l}' podstawionymi w miejsce argumentów. Ponieważ listy argumentów \bar{l} i \bar{l}' są izomorficzne, więc wyrażenia e i e' są izomorficzne.
Możemy więc wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2\llbracket l'.m(\bar{l}') \rrbracket_\emptyset :: \llbracket e' \rrbracket_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1\llbracket l_1.m(\bar{l}) \rrbracket_\emptyset :: \llbracket l_2 \rrbracket_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2\llbracket l'_1.m(\bar{l}') \rrbracket_\emptyset :: \llbracket l'_2 \rrbracket_\emptyset$
Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1\llbracket l_1.m(\bar{l}) \rrbracket_\emptyset :: \llbracket l_2 \rrbracket_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1\llbracket l_2 \rrbracket_\emptyset)$. Następuje więc jedynie zdjęcie wywołania metody ze stosu wywołań. Możemy zatem wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2\llbracket l'_2 \rrbracket_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.

- $\bar{C}_1 = \bar{C}'_1 :: C_1[l_1.x = l]_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2[l'_1.x = l']_\emptyset$, $l_1 \neq \mathbf{null}$
 Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1[l_1.x = l]_\emptyset) \rightarrow (h_{1,n}, \bar{C}'_1 :: C_1[l]_\emptyset)$, gdzie $o = h_1(l_1)\{x \mapsto l\}$, $h_{1,n} = h_1\{l_1 \mapsto o\}$
 Ponieważ izomorfizm jest różnowartościowy i zachowuje **null**, więc również $l'_1 \neq \mathbf{null}$.
 Weźmy zatem $o' = h_2(l'_1)\{x \mapsto l'\}$, $h_{2,n} = h_2\{l'_1 \mapsto o'\}$ i $h_{2,n}^{base} = h_2^{base}\{l'_1 \mapsto o'\}$.
 Możemy teraz wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[l']_\emptyset$ i pozostawić izomorfizm f bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1[l_1.x]_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2[l'_1.x]_\emptyset$, $l \neq \mathbf{null}$
 Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1[l_1.x]_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C[l_2]_\emptyset)$, gdzie $l_2 = h_1(l_1)(x)$.
 Ponieważ sterty h_1 i h_2 są izomorficzne, więc na sterzie h_2 istnieje obiekt pod lokacją l'_1 zawierający pole x i, co więcej, jeśli $l'_2 = h_2(l'_1)(x)$, to $f(l_2) = l'_2$.
 Zatem możemy wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C[l'_2]_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1[\mathbf{throw} l]_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2[\mathbf{throw} l']_\emptyset$, $l \neq \mathbf{null}$
 Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1[\mathbf{throw} l]_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1[l]_D)$, gdzie $\text{class}(h_1, l) = D$, zatem jedynym efektem jest zmiana trybu wykonania na wyjątkowy z wartością wyjątku l i typem D . Możemy więc po prostu wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[l']_D$, a sterty i izomorfizm pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1[\mathbf{try} \{e_1\} \mathbf{catch} (\mu C x) \{e_2\}]_\emptyset$, $\bar{C}_2 = \bar{C}'_2 :: C_2[\mathbf{try} \{e'_1\} \mathbf{catch} (\mu C x) \{e'_2\}]_\emptyset$
 Redukcja \bar{C}_1 jest postaci $(h_1, \bar{C}'_1 :: C_1[\mathbf{try} \{e_1\} \mathbf{catch} (\mu C x) \{e_2\}]_\emptyset) \rightarrow (h_1, \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket e_1 \rrbracket_\emptyset\} \mathbf{catch} (\mu C x) \{e_2\}])$, a zatem przebiega podobnie, jak analogiczna redukcja dla **let** – do kontekstu dodawane jest wyrażenie e_2 , a redeks zostaje zmieniony na e_1 . Podobnie jak dla **let**, możemy wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[\mathbf{try} \{\llbracket e'_1 \rrbracket_\emptyset\} \mathbf{catch} (\mu C x) \{e'_2\}]$, a sterty i izomorfizm pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket l \rrbracket_\emptyset\} \mathbf{catch} (\mu C x) \{e_2\}]$, $\bar{C}_2 = \bar{C}'_2 :: C_2[\mathbf{try} \{\llbracket l' \rrbracket_\emptyset\} \mathbf{catch} (\mu C x) \{e'_2\}]$
 Przy normalnym wykonaniu wyrażenia wewnątrz bloku **try**, jest on po prostu usuwany z kontekstu. Redukcja \bar{C}_1 jest wtedy postaci $(h_1, \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket l \rrbracket_\emptyset\} \mathbf{catch} (\mu C x) \{e_2\}]) \rightarrow (h_1, \bar{C}'_1 :: C_1[l]_\emptyset)$.
 Wystarczy zatem wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[l']_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket l \rrbracket_{C'}\} \mathbf{catch} (\mu C x) \{e_2\}]$, $\bar{C}_2 = \bar{C}'_2 :: C_2[\mathbf{try} \{\llbracket l' \rrbracket_{C'}\} \mathbf{catch} (\mu C x) \{e'_2\}]$, gdzie $C' \leq C$
 Redukcja w tym przypadku oznacza złapanie rzuconego wcześniej wyjątku i obsłużenie go przez e_2 . Jest ona postaci $(h_1, \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket l \rrbracket_{C'}\} \mathbf{catch} (\mu C x) \{e_2\}]) \rightarrow (h_1, \bar{C}'_1 :: C_1[e]_\emptyset)$, gdzie $e = e_2\{l/x\}$.
 Niech $e' = e'_2\{l'/x\}$. Wtedy wystarczy wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[e']_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.
- $\bar{C}_1 = \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket l \rrbracket_{C'}\} \mathbf{catch} (\mu C x) \{e_2\}]$, $\bar{C}_2 = \bar{C}'_2 :: C_2[\mathbf{try} \{\llbracket l' \rrbracket_{C'}\} \mathbf{catch} (\mu C x) \{e'_2\}]$, gdzie $C' \neq \emptyset, C' \not\leq C$
 W przypadku niezłapania wyjątku, kontekst **try** jest usuwany, a wyjątek przekazywany dalej. Redukcja jest postaci $(h_1, \bar{C}'_1 :: C_1[\mathbf{try} \{\llbracket l \rrbracket_{C'}\} \mathbf{catch} (\mu C x) \{e_2\}]) \rightarrow (h_1, \bar{C}'_1 :: C_1[l]_{C'})$. Bierzemy zatem $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[l']_{C'}$, a sterty i izomorfizm pozostawiamy bez zmian.

- $\bar{C}_1 = \bar{C}'_1 :: C_1[\mathbf{let} \ C \ x = \llbracket l \rrbracket_{C'} \ \mathbf{in} \ e], \quad \bar{C}_2 = \bar{C}'_2 :: C_2[\mathbf{let} \ C \ x = \llbracket l' \rrbracket_{C'} \ \mathbf{in} \ e'], \quad C' \neq \emptyset$

Przypadek wyjątku w czasie ewaluacji wyrażenia wewnętrznego **let** jest analogiczny jak poprzedni. Możemy wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[\llbracket l' \rrbracket_{C'}]$, a sterty i izomorfizm zostawić bez zmian.

- $\bar{C}_1 = \bar{C}'_1 :: C_1[\llbracket l_1.m(\bar{l}) \rrbracket_\emptyset :: \llbracket l_2 \rrbracket_C], \quad \bar{C}_2 = \bar{C}'_2 :: C_2[\llbracket l'_1.m(\bar{l}') \rrbracket_\emptyset :: \llbracket l'_2 \rrbracket_C], \quad C \neq \emptyset$

Podobnie w przypadku wyjątku podczas wykonywania metody. Wywołanie jest zdejmowane ze stosu, a wyjątek przekazywany wyżej. Redukcja jest postaci $(h_1, \bar{C}'_1 :: C_1[\llbracket l_1.m(\bar{l}) \rrbracket_\emptyset :: \llbracket l_2 \rrbracket_C]) \rightarrow (h_1, \bar{C}'_1 :: \llbracket l_2 \rrbracket_C)$.

Bierzemy $\bar{C}_{2,n} = (h_1, \bar{C}'_2 :: \llbracket l'_2 \rrbracket_C)$, a sterty i izomorfizm pozostawiamy bez zmian.

- Redukcja \bar{C}_1 powoduje odwołanie do lokacji **null**

Jest tak, gdy $\bar{C}_1 = \bar{C}'_1 :: C_1[\llbracket e \rrbracket_\emptyset]$, gdzie e jest postaci **null.x**, **null.x** = l , **null.m**(\bar{l}) lub **throw null**.

Redukcja \bar{C}_1 jest wtedy postaci $(h_1, \bar{C}'_1 :: C_1[\llbracket e \rrbracket_\emptyset]) \rightarrow (h_1, \bar{C}'_1 :: C_1[\llbracket \mathbf{npe} \rrbracket_{\mathbf{NPE}}])$.

Wówczas $\bar{C}_2 = \bar{C}'_2 :: C_2[\llbracket e' \rrbracket_\emptyset]$, gdzie e' jest izomorficzne z e , a ponieważ izomorfizm zachowuje **null** więc redukcja e' również powoduje odwołanie do **null**.

Ponieważ izomorfizm zachowuje także **npe**, możemy wziąć $\bar{C}_{2,n} = \bar{C}'_2 :: C_2[\llbracket \mathbf{npe} \rrbracket_{\mathbf{NPE}}]$, a sterty i izomorfizm pozostawić bez zmian.

□

Rozdział 7

Poprawność

Rozdział 8

Formalizacja w systemie Coq

Rozdział 9

Podsumowanie

Bibliografia

- [1] J. Chrzęszcz and A. Schubert. Function definitions for compound values in object-oriented languages. In *Proc. of the 19th International Symposium on Principles and Practice of Declarative Programming*, PPDP '17, pp. 61–72. ACM, 2017.
- [2] J. Chrzęszcz and A. Schubert. Formalisation of a frame stack semantics for a Java-like language. 2018