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 udowoniono 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

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.

$$
\begin{array}{rcl}
\mathsf{Prog} \ni \mathbf{C} & ::= & \mathbf{class}\ C_1\ \mathbf{ext}\ C_2\ \{\overline{\mathbf{F}}\ \overline{\mathbf{M}}\} \\
\mathsf{CId} \ni C & ::= & \langle \mathit{identifier} \rangle \quad \mathit{(class\ name)} \\
\mathbf{F} & ::= & \phi\ C\ x \\
\phi & ::= & \mathtt{rep} \mid \emptyset \\
\mathsf{Id} \ni x & ::= & \langle \mathit{identifier} \rangle \quad \mathit{(variable/field\ name)} \\
\mathsf{arg} & ::= & \mu\ C\ x \qquad \mathsf{argn}\ ::= \emptyset\ C\ x \\
\mathsf{Exc} & ::= & \mu\ C \qquad\quad \mathsf{Excn} ::= \emptyset\ C \\
\mathbf{M} & ::= & \mu\ C\ \mu\ m(\overline{\mathsf{arg}})\ \mathbf{throws}\ \overline{\mathsf{Exc}}\ \{E\} \mid \\
 & & \emptyset\ C\ \emptyset\ m(\overline{\mathsf{argn}})\ \mathbf{throws}\ \overline{\mathsf{Excn}}\ \{E\} \\
\mathsf{AMod} \ni \mu & ::= & \mathtt{rwr} \mid \mathtt{rd} \mid \mathtt{atm} \\
\mathsf{MId} \ni m & ::= & \langle \mathit{identifier} \rangle \quad \mathit{(method\ name)} \\
\mathsf{Expr} \ni E & ::= & \mathbf{new}\ \mu\ C(\overline{\mathsf{v}}) \mid \mathbf{let}\ C\ x = E_1\ \mathbf{in}\ E_2 \mid \\
 & & \mathbf{if}\ \mathsf{v}_1 == \mathsf{v}_2\ \mathbf{then}\ E_3\ \mathbf{else}\ E_4 \mid \mathsf{v}.m(\overline{\mathsf{v}}) \mid \\
 & & \mathsf{fieldref} = \mathsf{v} \mid \mathsf{v} \mid \mathsf{fieldref} \mid \mathbf{throw}\ \mathsf{v} \mid \\
 & & \mathbf{try}\ \{E_1\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E_2\} \\
\mathsf{v} & ::= & x \mid \mathbf{this} \mid \mathbf{null} \\
\mathsf{fieldref} & ::= & \mathsf{v}.x \\
A & ::= & C \mid \emptyset \\
\mathsf{BCtxt} \ni \mathcal{C} & ::= & [\![\ ]\!]_A \mid \mathbf{let}\ C\ x = \mathcal{C}\ \mathbf{in}\ E \mid \\
 & & \mathbf{try}\ \{\mathcal{C}\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E\}
\end{array}
$$

Rysunek 2.1: Składnia języka Jafun

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

$$\overline{\mathbf{C}},\quad h, \mathcal{C}_1[\![E_1]\!]_{A_1} :: \cdots :: \mathcal{C}_n[\![E_n]\!]_{A_n} \to h', \mathcal{C}'_1[\![E_1]\!]_{A'_1} :: \cdots :: \mathcal{C}'_m[\![E_m]\!]_{A'_m}.$$

Stos wywołań $\mathcal{C}_1[\![E_1]\!]_{A_1} :: \cdots :: \mathcal{C}_n[\![E_n]\!]_{A_n}$, albo w skrócie $\overline{\mathcal{C}}$, to ciąg wyrażeń z rysunku 2.2, w którym aktualnie ewaluowane wyrażenie (redeks) jest oznaczone specjalnym symbolem $[\![\ ]\!]_A$. 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 $C_i \in \mathsf{BCtxt}$ i redeks E_i. Kontekst C_i opisuje wszystkie zagnieżdżone bloki **let** i **catch**, wewnątrz którch znajduje się E_i.

Ewaluacja programu zaczyna się od stanu $\overline{\mathbf{C}}, h, [\![E']\!]_\emptyset$ gdzie $h \in \mathsf{Heap}$, $E' = E\{l_o/\mathbf{this}\}$, $\mathsf{class}(h, l_o) = C$, a $C'\ m()$ **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 object klasy `NPE` ("null pointer exception").

2.2. Ewaluacja

Ewaluacją konfiguracji (h, st) będziemy nazywać dowolny ciąg par $confs = (h_1, st_1), \ldots, (h_n, st_n)$, taki że $h_1 = h$, $st_1 = st$ oraz $(h_i, st_i) \to (h_{i+1}, st_{i+1})$ dla $1 \leq i < n$.

Ewaluacją wyrażenia e na stercie h będziemy nazywać taką ewaluację konfiguracji $(h, [\![e]\!]_\phi)$, że $st_n = [\![l]\!]_A$ dla pewnych l, A. Jeśli taka ewaluacja istnieje, będziemy to oznaczać jako $(h, e) \overset{confs}{\rightsquigarrow} (h_n, A, l)$

(newk) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{new}\ \mu\ C(l_1, \ldots, l_k)]\!]_\emptyset \rightarrow h'', \overline{\mathcal{C}} :: \mathcal{C}[\![l_0]\!]_\emptyset$
gdzie $\mathsf{alloc}(h, \overline{\mathcal{C}}, C) = (l_0, h'),\ \mathsf{flds}(C) = x_1, \ldots, x_k,$
$o = \mathsf{empty}_C\{x_1 \mapsto l_1, \ldots, x_k \mapsto l_k\},\ h'' = h'\{l_0 \mapsto o\}$

(letin) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{let}\ C\ x = E_1\ \mathbf{in}\ E_2]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = [\![E_1]\!]_\emptyset\ \mathbf{in}\ E_2]$
(letgo) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = [\![l]\!]_\emptyset\ \mathbf{in}\ E] \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![E\{l/x\}]\!]_\emptyset$

(ifeq) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{if}\ l_0 == l_1\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![E_1]\!]_\emptyset$ gdzie $l_0 = l_1$
(ifneq) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{if}\ l_0 == l_1\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![E_2]\!]_\emptyset$ gdzie $l_0 \neq l_1$

(mthdnpe) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{null}.m(\bar{l})]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathsf{npe}]\!]_{\mathrm{NPE}}$
(mthd) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![l.m(\bar{l})]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l.m(\bar{l})]\!]_\emptyset :: [\![E]\!]_\emptyset$
gdzie $\mathsf{class}(h, l) = D, \mathsf{body}(D, m) = E_0, E = E_0\{l/\mathbf{this}, \bar{l}/\mathsf{parNms}(D, m)\}$
(mthdret) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![l.m(\bar{l})]\!]_\emptyset :: [\![l']\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l']\!]_\emptyset$

(assignnpe) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{null}.x = l]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathsf{npe}]\!]_{\mathrm{NPE}}$
(assignev) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![l_1.x = l]\!]_\emptyset \rightarrow h', \overline{\mathcal{C}} :: \mathcal{C}[\![l]\!]_\emptyset$
gdzie $l_1 \neq \mathbf{null}, o = h(l_1)\{x \mapsto l\}, h' = h\{l_1 \mapsto o\}$

(varnpe) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{null}.x]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathsf{npe}]\!]_{\mathrm{NPE}}$
(var) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![l.x]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l']\!]_\emptyset$ gdzie $l \neq \mathbf{null}, l' = h(l)(x)$

(thrownull) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{throw\ null}]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathsf{npe}]\!]_{\mathrm{NPE}}$

(throw) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{throw}\ l]\!]_\emptyset \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l]\!]_D$ gdzie $l \neq \mathbf{null}, \mathsf{class}(h, l) = D$

(ctchin) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{try}\ \{E_1\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E_2\}]\!]_\emptyset \rightarrow$
$h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{try}\ \{[\![E_1]\!]_\emptyset\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E_2\}]$
(ctchnrml) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{try}\ \{[\![l]\!]_\emptyset\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E_2\}] \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l]\!]_\emptyset$
(ctchexok) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{try}\ \{[\![l]\!]_{C'}\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E_2\}] \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![E_2']\!]_\emptyset$
gdzie $E_2' = E_2\{l/x\}, C' \leq: C$

(letex) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = [\![l]\!]_{C'}\ \mathbf{in}\ E] \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l]\!]_{C'}$ gdzie $C' \neq \emptyset$
(methodex) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![l.m(\bar{l})]\!]_\emptyset :: [\![l']\!]_C \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l']\!]_C$ gdzie $C \neq \emptyset$
(ctchexnok) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{try}\ \{[\![l]\!]_{C'}\}\ \mathbf{catch}\ (\mu\ C\ x)\ \{E_2\}] \rightarrow h, \overline{\mathcal{C}} :: \mathcal{C}[\![l]\!]_{C'}$
gdzie $C' \neq \emptyset, C' \not\leq: C$

Rysunek 2.2: Semantyka języka Jafun

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 zawartości sterty i operatorami separacji $*$ i $-\!\!*$.

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 $v_1 = v_2 \Rightarrow \texttt{False}$.

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

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 $[\![\cdot]\!]$ 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 $-\!\!*$ to pewnego rodzaju odwrotność operatora $*$ – sterta spełnia $P -\!\!* Q$, jeśli po połączeniu jej z dowolną rozłączną stertą spełniającą P, otrzymana sterta spełnia Q.

$$
\begin{aligned}
[\![\texttt{True}]\!] &\triangleq \top \\
[\![\texttt{False}]\!] &\triangleq \bot \\
[\![\exists x : C.P]\!]_{h,env} &\triangleq \exists l : Loc \;.\; class(h,l) = C \wedge [\![P]\!]_{h,env[x \mapsto l]} \\
[\![P \wedge Q]\!]_{h,env} &\triangleq [\![P]\!]_{h,env} \wedge [\![Q]\!]_{h,env} \\
[\![P \vee Q]\!]_{h,env} &\triangleq [\![P]\!]_{h,env} \vee [\![Q]\!]_{h,env} \\
[\![P \Rightarrow Q]\!]_{h,env} &\triangleq [\![P]\!]_{h,env} \Rightarrow [\![Q]\!]_{h,env} \\
[\![x = y]\!]_{h,env} &\triangleq env(x) = env(y) \\
[\![x \hookrightarrow f = y]\!]_{h,env} &\triangleq h(env(x))(f) = env(y) \\
[\![\{P\}e\{x.Q\}_A]\!]_{h,env} &\triangleq \forall h : Heap \;.\; [\![P]\!]_{h,env} \Rightarrow \\
&\qquad \exists h' : Heap, l : Loc \;.\; (h, e[/env]) \leadsto (h', A, l) \wedge [\![Q]\!]_{h',env[x \mapsto l]} \\
[\![P * Q]\!]_{h,env} &\triangleq \exists h_1, h_2 : Heap \;.\; h_1 \oplus h_2 = h \wedge [\![P]\!]_{h_1,env} \wedge [\![Q]\!]_{h2,env} \\
[\![P \mathrel{-\!\!*} Q]\!]_{h,env} &\triangleq \forall h' : Heap \;.\; [\![P]\!]_{h',env} \Rightarrow [\![Q]\!]_{h \oplus h',env}
\end{aligned}
$$

Uwaga: $e[/env]$ 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 $\Gamma|P \vdash Q$, gdzie Γ to środowisko typów, przypisujące zmiennym odpowiadające im typy (czyli nazwy klas), a P i Q to termy logiki. Intuicyjnie, osąd $\Gamma|P \vdash Q$ 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.

$$\textsc{Asm}\ \frac{}{P \vdash P} \qquad \textsc{Trans}\ \frac{P \vdash Q \qquad Q \vdash R}{P \vdash R} \qquad \textsc{Eq-refl}\ \frac{}{P \vdash v = v} \qquad \textsc{Eq-sym}\ \frac{P \vdash v = w}{P \vdash w = v}$$

$$\bot\mathrm{E}\ \frac{P \vdash \texttt{False}}{P \vdash Q} \qquad \top\mathrm{I}\ \frac{}{P \vdash \texttt{True}} \qquad \wedge\mathrm{I}\ \frac{R \vdash P \qquad R \vdash Q}{R \vdash P \wedge Q} \qquad \wedge\mathrm{EL}\ \frac{R \vdash P \wedge Q}{R \vdash P}$$

$$\wedge\mathrm{ER}\ \frac{R \vdash P \wedge Q}{R \vdash Q} \qquad \vee\mathrm{IL}\ \frac{R \vdash P}{R \vdash P \vee Q} \qquad \vee\mathrm{IR}\ \frac{R \vdash Q}{R \vdash P \vee Q}$$

$$\vee\mathrm{E}\ \frac{S \vdash P \vee Q \qquad P \vdash R \qquad Q \vdash R}{S \vdash R} \qquad \Rightarrow\mathrm{I}\ \frac{R \wedge P \vdash Q}{R \vdash P \Rightarrow Q} \qquad \Rightarrow\mathrm{E}\ \frac{R \vdash P \Rightarrow Q \qquad R \vdash P}{R \vdash P \Rightarrow Q}$$

$$\exists\mathrm{I}\ \frac{\Gamma, x : C|Q \vdash P[t/x]}{Q \vdash \exists x : C.P} \qquad \exists\mathrm{E}\ \frac{\Gamma|R \vdash \exists x : C.P \qquad \Gamma, x : C|R \wedge P \vdash Q}{\Gamma|R \vdash Q}$$

Rysunek 4.1: Reguły wnioskowania dla tradycyjnych operatorów logicznych

$$\textsc{Weak}\ \frac{}{P * Q \vdash P} \qquad \textsc{Sep-assoc}\ \frac{}{P * (Q * R) \dashv\vdash (P * Q) * R} \qquad \textsc{Sep-sym}\ \frac{}{P * Q \vdash Q * P}$$

$$*\mathrm{I}\ \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * Q_1 \vdash P_2 * Q_2} \qquad -\!\!*\mathrm{I}\ \frac{R * P \vdash Q}{R \vdash P -\!\!* Q} \qquad -\!\!*\mathrm{E}\ \frac{R_1 \vdash P -\!\!* Q \qquad R_2 \vdash P}{R_1 * R_2 \vdash Q}$$

Rysunek 4.2: Reguły wnioskowania dla operatorów separacyjnych

Reguły strukturalne dla trójek Hoare'a

$$\textsc{Ht-frame}\ \frac{S \vdash \{P\}e\{v.Q\}_A \qquad S \text{ jest trwały}}{S \vdash \{P * R\}e\{v.Q * R\}_A} \qquad \textsc{Ht-ret}\ \frac{}{S \vdash \{\texttt{True}\}w\{v.v = w\}_\phi}$$

$$\textsc{Ht-csq}\ \frac{\Gamma|S \vdash P \Rightarrow P' \qquad \Gamma|S \vdash \{P'\}e\{v.Q'\}_A \qquad \Gamma, v : C|S \vdash Q' \Rightarrow Q \qquad S \text{ jest trwały}}{S \vdash \{P\}e\{v.Q\}_A}$$

$$\textsc{Ht-disj}\ \frac{S \vdash \{P\}e\{v.Q\}_A \qquad S \vdash \{Q\}e\{v.Q\}_A}{S \vdash \{P \vee Q\}e\{v.Q\}_A}$$

$$\textsc{Ht-pers}\ \frac{S \wedge R \vdash \{Q\}e\{v.Q\}_A}{S \vdash \{Q \wedge R\}e\{v.Q\}_A} \text{ jeśli R trwały}$$

Reguły dla trójek Hoare'a opisujących konstrukcje języka

$$\textsc{Ht-new-null}\ \frac{}{S \vdash \{\texttt{True}\}\mathbf{new}\ C(\overline{v})\{w.w \neq \mathbf{null}\}_\phi}$$

$$\textsc{Ht-new-field}\ \frac{\mathrm{flds}(C) = f_1, \ldots, f_n}{S \vdash \{\texttt{True}\}\mathbf{new}\ C(v_1, \ldots, v_n)\{w.w \hookrightarrow f_i = v_i\}_\phi}$$

$$\textsc{Ht-let}\ \frac{\Gamma|S \vdash \{P\}E_1\{x.Q\}_\phi \qquad \Gamma, x : C|S \vdash \{Q\}E_2\{w.R\}_A}{\Gamma|S \vdash \{P\}\mathbf{let}\ C\ x = E_1\ \mathbf{in}\ E_2\{w.R\}_A} \text{ jeśli S trwały}$$

$$\textsc{Ht-let-ex}\ \frac{S \vdash \{P\}E_1\{w.Q\}_A \qquad A \neq \phi}{\Gamma|S \vdash \{P\}\mathbf{let}\ C\ x = E_1\ \mathbf{in}\ E_2\{w.Q\}_A}$$

$$\textsc{Ht-field-set}\ \frac{}{S \vdash \{x \neq \texttt{null}\}x.f = v\{\_.x \hookrightarrow f = v\}_\phi}$$

$$\textsc{Ht-null-set}\ \frac{}{S \vdash \{x = \texttt{null}\}x.f = v\{w.w = \mathrm{npe}\}_{\texttt{NPE}}}$$

$$\textsc{Ht-field-get}\ \frac{}{S \vdash \{x \hookrightarrow f = v\}x.f\{w.w = v\}_\phi}$$

$$\textsc{Ht-null-get}\ \frac{}{S \vdash \{x = \texttt{null}\}x.f\{w.w = \mathrm{npe}\}_{\texttt{NPE}}}$$

Rysunek 4.3: Reguły wnioskowania dla trójek Hoare'a

$$\textsc{Ht-if}\ \frac{S \vdash \{P \wedge v_1 = v_2\}E_1\{w.Q\}_A \qquad S \vdash \{P \wedge v_1 \neq v_2\}E_2\{w.Q\}_A}{S \vdash \{P\}\textbf{if } v_1 = v_2 \textbf{ then } E_1 \textbf{ else } E_2\{w.Q\}_A}$$

$$\textsc{Ht-invoke}\ \frac{\Gamma \vdash x : C \qquad \begin{array}{c}\{P'\} \cdot \{w.Q'\}_A \in \texttt{invariants}(C, m) \\ S \wedge \{P'\}x.m(\overline{v})\{w.Q'\}_A \vdash \{P\}x.m(\overline{v})\{w.Q\}_A\end{array}}{S \vdash \{P\}x.m(\overline{v})\{w.Q\}_A}$$

$$\textsc{Ht-null-invoke}\ \frac{}{S \vdash \{x = \texttt{null}\}x.m(\overline{v})\{w.w = \texttt{npe}\}_{\texttt{NPE}}}$$

$$\textsc{Ht-throw}\ \frac{\Gamma \vdash x : C}{S \vdash \{x \neq \texttt{null}\}\textbf{throw } x\{w.w = x\}_C}$$

$$\textsc{Ht-null-throw}\ \frac{}{S \vdash \{x = \texttt{null}\}\textbf{throw } x\{w.w = \texttt{npe}\}_{\texttt{NPE}}}$$

$$\textsc{Ht-catch-normal}\ \frac{S \vdash \{P\}E_1\{w.Q\}_\phi}{S \vdash \{P\}\textbf{try } E_1 \textbf{ catch } (C\ x)\ E_2\{w.Q\}_\phi}$$

$$\textsc{Ht-catch-ex}\ \frac{\Gamma|S \vdash \{P\}E_1\{x.Q\}'_C \qquad \Gamma, x : C'|S \vdash \{Q\}E_2\{w.R\}_A \qquad C' \leq C}{\Gamma|S \vdash \{P\}\textbf{try } E_1 \textbf{ catch } (C\ x)\ E_2\{w.R\}_A} \text{ jeśli S trwały}$$

$$\textsc{Ht-catch-pass}\ \frac{S \vdash \{P\}E_1\{w.Q\}'_C \qquad C' \not\leq C}{S \vdash \{P\}\textbf{try } E_1 \textbf{ catch } (C\ x)\ E_2\{w.Q\}'_C}$$

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

Rozdział 5

Własności ewaluacji

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

5.1. Łączenie ewaluacji

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

Twierdzenie 1 (O łączeniu ewaluacji)**.** *Niech* $(h, st), (h', st'), (h'', st'')$ *będą konfiguracjami, a* $confs$ *i* $confs'$ *– ciągami konfiguracji, takimi że* $(h, st) \overset{confs}{\rightsquigarrow} (h', st')$ *i* $(h', st') \overset{confs'}{\rightsquigarrow} (h'', st'')$*. Wtedy* $(h, st) \overset{confs{+}{+}confs'}{\rightsquigarrow} (h'', st'')$*.*

Dowód. TODO □

5.2. Ewaluacja przy rozszerzonym stosie i kontekście

5.3. 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.

Równoważnośc 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 stercie) + 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.

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

Definicja 5.3.1 (izomorfizm stert)**.**
Niech $h_1, h_2 : \mathsf{Heap}$. Funkcję $f : \mathrm{Dom}(h_1) \cup \{\mathtt{null}\} \to \mathrm{Dom}(h_2) \cup \{\mathtt{null}\}$ nazwiemy izomorfizmem między tymi stertami, jeśli:

1. f jest bijekcją
2. $f(\texttt{null}) = \texttt{null}$
3. 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 konfiguracji (czyli ciągów par (sterta, stos wywołań)), a zatem należy zdefiniować także izomorfizmy między stosami wywołań. Służy temu kolejnych kilka definicji.

Rozdział 6

Poprawność

Rozdział 7

Formalizacja w systemie Coq

Rozdział 8

Podsumowanie

Bibliografia

[1] J. Chrząszcz 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. Chrząszcz and A.Schubert. Formalisation of a frame stack semantics for a Java-like language. 2018