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

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 $\mu_r\ C_4\ \mu_o\ m(\overline{\mathsf{arg}})$ **throws** $\overline{\mathsf{Exc}}$ $\{E_1\}$ będzie deklaracją metody w $\overline{\mathbf{M}}$, gdzie $\overline{\mathsf{arg}} = \mu'_1\ C'_1\ x_1, \ldots, \mu'_n\ C'_n\ x_n$, $\overline{\mathsf{Exc}} = \mu''_1\ C''_1, \ldots, \mu''_k\ C''_k$, and $\mu_r, \mu_o, \mu'_i, \mu''_j \in$ AMod for all possible i, j. Ustalając $\overline{\mathbf{M}}_1 \dot\cup \overline{\mathbf{M}}_2 = \overline{\mathbf{M}}_1 \cup \{\mathbf{M} \in \overline{\mathbf{M}}_2 \mid \mathsf{name}(\mathbf{M}) \notin \overline{\mathbf{M}}_1\}$, możemy zdefiniować następujące pomocnicze notacje:

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

Rysunek 2.2: Notacje pomocnicze

Semantyka małych kroków języka Jafun jest zdefiniowana przez relację $\to$ na rysunku 2.3, 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 2.1, 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 $\mathcal{C}_i \in \mathsf{BCtxt}$ i redeks E_i. Kontekst $\mathcal{C}_i$ opisuje wszystkie zagnieżdżone bloki **let** i **catch**, wewnątrz których znajduje się E_i.

(newk) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{new}\ \mu\ C(l_1, \ldots, l_k)\rrbracket_\emptyset \to h'', \overline{\mathcal{C}} :: \mathcal{C}\llbracket l_0 \rrbracket_\emptyset$
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) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{let}\ C\ x = E_1\ \mathbf{in}\ E_2 \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = \llbracket E_1 \rrbracket_\emptyset\ \mathbf{in}\ E_2]$
(letgo) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = \llbracket l \rrbracket_\emptyset\ \mathbf{in}\ E] \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket E\{l/x\} \rrbracket_\emptyset$

(ifeq) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{if}\ l_0 == l_1\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2 \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket E_1 \rrbracket_\emptyset$ gdzie $l_0 = l_1$
(ifneq) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{if}\ l_0 == l_1\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2 \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket E_2 \rrbracket_\emptyset$ gdzie $l_0 \neq l_1$

(mthdnpe) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{null}.m(\bar{l}) \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathsf{npe} \rrbracket_{\mathtt{NPE}}$
(mthd) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l.m(\bar{l}) \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l.m(\bar{l}) \rrbracket_\emptyset :: \llbracket E \rrbracket_\emptyset$
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) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l.m(\bar{l}) \rrbracket_\emptyset :: \llbracket l' \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l' \rrbracket_\emptyset$

(assignnpe) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{null}.x = l \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathsf{npe} \rrbracket_{\mathtt{NPE}}$
(assignev) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l_1.x = l \rrbracket_\emptyset \to h', \overline{\mathcal{C}} :: \mathcal{C}\llbracket l \rrbracket_\emptyset$
gdzie $l_1 \neq \mathbf{null}, o = h(l_1)\{x \mapsto l\}, h' = h\{l_1 \mapsto o\}$

(varnpe) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{null}.x \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathsf{npe} \rrbracket_{\mathtt{NPE}}$
(var) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l.x \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l' \rrbracket_\emptyset$ gdzie $l \neq \mathbf{null}, l' = h(l)(x)$

(thrownull) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{throw\ null} \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathsf{npe} \rrbracket_{\mathtt{NPE}}$

(throw) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket \mathbf{throw}\ l \rrbracket_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}\llbracket l \rrbracket_D$ gdzie $l \neq \mathbf{null}, \mathsf{class}(h, l) = D$

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

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

Rysunek 2.3: Semantyka języka Jafun

Ewaluacja programu zaczyna się od stanu $\overline{\mathbf{C}}, h, \llbracket E' \rrbracket_\emptyset$ 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 $\mathit{confs} = (h_1, st_1), \ldots, (h_n, st_n)$, 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 $(h, \llbracket e \rrbracket_\phi)$, że $st_n = \llbracket l \rrbracket_A$ dla pewnych l, A. Jeśli taka ewaluacja istnieje, będziemy to oznaczać jako

$(h, e) \overset{confs}{\rightsquigarrow} (h_n, A, l)$

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

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

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.

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

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

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

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

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

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

Reguły dla trójek Hoare'a opisujących konstrukcje języka

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

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

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

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

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

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

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

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

Rysunek 4.3: Reguły wnioskowania dla trójek Hoare'a

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

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

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

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

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

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

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

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

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

Rozdział 5

Własności 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 $l_2 \in \mathsf{Dom}(h)$. $\square$

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* $(h, st), (h', st'), (h'', st'')$ *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 □

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.

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.

6.3.1. Izomorfizmy

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

Definicja 6.3.1 (izomorfizm stert).
Niech h_1, h_2 : 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. $f(\mathtt{null}) = \mathtt{null}$
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 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 $\mathcal{C}_1, \mathcal{C}_2$ będą kontekstami ewaluacji Jafun. Powiemy, że f jest izomorfizmem między tymi kontekstami, jeśli

- $\mathcal{C}_1 = \mathcal{C}_2 = [\![\ ]\!]_A$

  lub

- $\mathcal{C}_1 = \textbf{let } C\ x = \mathcal{C}_1' \textbf{ in } E_1, \quad \mathcal{C}_2 = \textbf{let } C\ x = \mathcal{C}_2' \textbf{ in } E_2$,
  gdzie f jest izomorfizmem między $\mathcal{C}_1'$ i $\mathcal{C}_2'$ oraz między E_1 i E_2

lub

- $\mathcal{C}_1 = \textbf{try}\ \{\mathcal{C}'_1\}\ \textbf{catch}\ (\mu\ C\ x)\ \{E_1\}$, $\quad \mathcal{C}_2 = \textbf{try}\ \{\mathcal{C}'_2\}\ \textbf{catch}\ (\mu\ C\ x)\ \{E_2\}$,
  gdzie f jest izomorfizmem między $\mathcal{C}'_1$ i $\mathcal{C}'_2$ 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 $confs_1, h_{1,n}, A, l_1$ będą takie, że $(h_1, E[/env]) \overset{confs_1}{\rightsquigarrow} (h_{1,n}, A, l_1)$.

Wtedy istnieją $h_{1,n}^{base}, h_{2,n}^{base}, confs_2, h_{2,n}, l_2, f$, 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. $(h_2, E[/env]) \overset{confs_2}{\rightsquigarrow} (h_{2,n}, A, l_2)$.

□

Sterty h_1 i h_2 w powyższym twierdzeniu to dwie sterty, 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 stercie h_2, taka że sterty docelowe oraz wyniki ewaluacji są izomorficzne, a fragmenty stert nie mające związku ze zmiennymi wolnymi pozostają niezmienione.

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

Rozdział 7

Poprawność

Rozdział 8

Formalizacja w systemie Coq

Rozdział 9

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