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 dla tego języka 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, Iris, semantyka, weryfikacja programów

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

F. Theory of Computation
F.3. Logics and meanings of programs
F.3.1 Specifying and verifying and reasoning about programs

Spis treści

Wprowadzenie

Iris jest logiką separacyjną wyższego rzędu opisaną w pracach [3] i [4]. Logika separacyjna to logika będąca rozszerzeniem logiki Hoare'a o operatory umożliwiające opisywanie rozłącznych zasobów, takich jak sterty czy bardziej skomplikowane obiekty.

Iris pozwala na wnioskowanie o programach w języku $\lambda_{\text{ref,conc}}$, będącym rachunkiem lambda z dodatkowymi słowami kluczowymi pozwalającymi na dostęp do sterty i wykonanie równoległe, jednak z uwagi na jego ogólność, możliwe jest też stosowanie go do innych języków. Przykładem takiego zastosowania jest wykorzystanie Iris do udowodnienia poprawności systemu typów języka Rust (a dokładniej pewnego jego uproszczenia, λ_{Rust}) w pracy [5].

Iris jest, mimo jego zalet, skomplikowanym systemem, o wysokim poziomie ogólności i skomplikowanym modelu. W celu zachowania poprawności, model Iris budowany przy użyciu techniki znanej jako *step-indexing* ([6]). Nie rozpatruje się w tym podejściu zwykłych zbiorów i funkcji, ale bardziej skomplikowane struktury algebraiczne.

W niniejszej pracy przedstawiamy prostszą logikę separacyjną dla podobnego do Javy, imperatywnego i zorientowanego obiektowo języka Jafun. Przy jej definiowaniu przyjęliśmy jednak podejście odmienne od Iris, na którym się wzorowaliśmy. Wyszliśmy od istniejącej już semantyki języka Jafun (opisanej w [2]), zdefiniowaliśmy prostą semantykę dla logiki separacji, a następnie wzięliśmy taki jej wycinek, żeby zagwarantować jej poprawność względem semantyki języka.

Rozdział 1

Jafun

Jafun to zorientowany obiektowo język programowania podobny do Javy. Jego szczegółowy opis znajduje się w pracach [1] i [2]. Poniżej przytaczam te aspekty języka, które są istotne dla prezentowanej logiki.

1.1. 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.

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

Rysunek 1.1: 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 ϕ C_3 x będzie deklaracją pola w $\overline{\mathbf{F}}$. Niech μ_r C_4 μ_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 $\mu_r, \mu_o, \mu_i', \mu_j'' \in \mathsf{AMod}$ for all possible i, j. Ustalając $\overline{\mathbf{M}}_1 \cup \overline{\mathbf{M}}_2 = \overline{\mathbf{M}}_1 \cup \{\mathbf{M} \in \overline{\mathbf{M}}_2 \mid \mathsf{name}(\mathbf{M}) \notin \overline{\mathbf{M}}_1\}$, możemy zdefiniować następujące pomocnicze notacje:

$C_1 \in \overline{\mathbf{C}}$	kiedy w deklaracja C_1 istnieje w $\overline{\mathbf{C}}$,
$\mathtt{Object} \in \overline{\mathbf{C}}$, $\mathtt{NPE} \in \overline{\mathbf{C}}$	
$\mathsf{flds}(C_1) = \{x \in \mathsf{Id} \mid \phi\, D_1\, x \in \overline{\mathbf{F}}\} \cup \mathsf{flds}(C_2)$	$\mathsf{flds}(\mathsf{Object}) = \emptyset$
$\overline{\mathsf{flds}}(C_1) = \overline{\mathbf{F}}, \overline{\mathsf{flds}}(C_2)$	$\overline{\mathsf{flds}}(\mathsf{Object}) = \emptyset$
$\mathsf{mthds}(C_1) = \overline{\mathbf{M}} \cup \mathsf{mthds}(C_2)$	$\mathsf{mthds}(\mathsf{Object}) = \emptyset$
$\mathsf{ext}(C_1) = C_2$	$\mathsf{ext}(\mathsf{Object}) = \emptyset$
$x \in C_1$	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 1.2: Notacje pomocnicze

Semantyka małych kroków języka Jafun jest zdefiniowana przez relację $\to$ na rysunku 1.3, 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 1.1, 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) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{new}\ \mu\ C(l_1, \ldots, l_k)]\!]_\emptyset \to h'', \overline{\mathcal{C}} :: \mathcal{C}[\![l_0]\!]_\emptyset$
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) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\![\mathbf{let}\ C\ x = E_1\ \mathbf{in}\ E_2]\!]_\emptyset \to h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = [\![E_1]\!]_\emptyset\ \mathbf{in}\ E_2]$
(letgo) $\overline{\mathbf{C}}, h, \overline{\mathcal{C}} :: \mathcal{C}[\mathbf{let}\ C\ x = [\![l]\!]_\emptyset\ \mathbf{in}\ E] \to h, \overline{\mathcal{C}} :: \mathcal{C}[\![E\{l/x\}]\!]_\emptyset$

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

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

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

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

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

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

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

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

Rysunek 1.3: Semantyka języka Jafun

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 obiekt klasy NPE ("null pointer exception").

1.2. Ewaluacja

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

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

że $\overline{C}_n = [\![l]\!]_A$ dla pewnych l, A. Jeśli taka ewaluacja istnieje, będziemy to oznaczać jako $(h, e) \overset{confs}{\rightsquigarrow} (h_n, A, l)$ lub

Rozdział 2

Składnia i semantyka

2.1. Składnia

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 2.1).

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

Rysunek 2.1: Składnia logiki

2.2. Oznaczenia i notacje

Definicja 2.2.1 (Środowisko)**.**
Środowisko to funkcja częściowa przypisująca identyfikatorom zmiennych i wartości **this** lokacje na stercie lub `null`. □

Definicja 2.2.2 (Sterta)**.**
Sterta to funkcja częściowa przypisująca lokacjom obiekty języka Jafun. □

Definicja 2.2.3 (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 $l_2 \in \mathsf{Dom}(h)$. □

Definicja 2.2.4 (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 $l \neq \mathsf{npe}$ i $l \in h_1$ i $l \in h_2$

□

Osłabienie warunku rozłączności sterty poprzez dopuszczenie występowania lokacji npe jednocześnie w obu stertach wynika z dwóch czynników. Po pierwsze, semantyka języka Jafun wymaga, aby sterta, na której wykonywany jest program, zawierała tę lokację, zatem aby można było ewaluować pewne wyrażenia Jafun na obu rozłącznych częściach sterty, lokacja npe musi znajdować się na obu z nich. Po drugie, ponieważ klasa `NPE` nie zawiera żadnych pól, obiekt pod lokacją npe nie może być modyfikowany. Dzięki temu nie może dojść do konfliktu wynikającego z różnic między obiektami pod tą samą lokacją.

Semantykę termu P na stercie h w środowisku env będziemy oznaczać jako $[\![P]\!]_{h,env}$. Może ona przyjmować wartości $\top$ lub $\bot$, oznaczające odpowiednio prawdę i fałsz. W naturalny sposób powiemy, że sterta h spełnia term P w środowisku env wtedy i tylko wtedy, gdy $[\![P]\!]_{h,env} = \top$.

Dodatkowo powiemy, że term P jest *trwały* (ang. *persistent*), jeśli nie zawiera żadnych operatorów opisujących sterty – $\hookrightarrow$, $*$ ani $-\!\!*$. Może on natomiast zawierać porównania $=$, spójniki logiczne i trójki Hoare'a. Trwałe termy będą odgrywały istotną rolę w regułach wnioskowania dla naszej logiki, ponieważ, jak zobaczymy później, ich semantyka nie zależy od wyboru konkretnej sterty.

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

2.3. Semantyka

Semantyka logiki dla danej sterty i środowiska jest przedstawiona na rysunku 2.2. 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.

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

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

Rysunek 2.2: Semantyka logiki

Semantyka jest standardowa dla tradycyjnych spójników logicznych i wartości `True` oraz `False`.

Intuicyjnie, sterta spełnia term $\exists x : C.P$, jeśli istnieje na niej taka lokacja l, że obiekt znajdujący się pod nią jest typu C oraz spełniony jest term $P[l/x]$. Dla uproszczenia dalszego wnioskowania, w formalnym zapisie semantyki ten ostatni warunek został zastąpiony przez odpowiednią modyfikację środowiska.

W logice występują dwa różne sposoby porównania wartości. Pierwsza z nich, $x = y$ jest interpretowana jako porównanie lokacji, na które mapowane są w środowisku zmienne x i y (oczywiście zmienne te muszą być obecne w środowisku). Nie następuje przy tym odwołanie do sterty. Drugim sposobem jest porównanie $x \hookrightarrow f = y$. Ono z kolei jest interpretowane jako porównanie pola f obiektu znajdującego się pod lokacją odpowiadająca zmiennej x z lokacją odpowiadającą zmiennej y. W tym przypadku również x i y muszą być obecne w środowisku, a dodatkowo lokacja odpowiadająca zmiennej x nie może być równa **null**, musi być obecna na stercie oraz zawierać pole f.

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, jeśli $A = \emptyset$), a wynikowa sterta będzie spełniała Q, w którym za x podstawiony zostanie wynik obliczenia.

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

Rozdział 3

Reguły wnioskowania

Ustalmy pewien program $\overline{\mathbf{C}}$ języka Jafun i listę niezmienników `invariants`, zawierającą trójki postaci $(C, m, \{P\} \cdot \{w.Q\}_A)$, składające się z nazwy klasy z $\overline{\mathbf{C}}$, nazwy metody w tej klasie i trójki Hoare'a ze znakiem $\cdot$ zamiast wyrażenia. Każdy taki niezmiennik będzie naszym założeniem na temat zachowania metody m wywołanej na obiekcie klasy C (lub obiekcie dowolnej klasy będącej rozszerzeniem C). Intuicyjnie, jeśli $(C, m, \{P\} \cdot \{w.Q\}_A) \in$ `invariants`, to zakładamy, że każdy obiekt klasy C lub pochodnej spełnia trójkę Hoare'a $\{P\}E\{w.Q\}_A$, gdzie E to ciało metody m.

Osądy w prezentowanej logice są postaci $\overline{\mathbf{C}}, \texttt{invariants}, \Gamma | P \vdash Q$, gdzie Γ to środowisko typów, przypisujące zmiennym i **this** odpowiadające im typy (czyli nazwy klas), a P i Q to termy logiki. Intuicyjnie, osąd $\overline{\mathbf{C}}, \texttt{invariants}, \Gamma | P \vdash Q$ 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. Ponieważ $\overline{\mathbf{C}}$ i `invariants` są ustalone i wspólne dla osądów we wszystkich regułach, będą one również pomijane.

Dodatkowo zakładamy, że wszystkie zmienne wolne występujące w termach w danym osądzie muszą być obecne w środowisku typów dla tego osądu, tj. żeby osąd $\Gamma | P \vdash Q$ mógł pojawić się w regule, wszystkie zmienne wolne w termach P i Q muszą mieć przypisany pewien typ w Γ. Będziemy też wymagać, żeby każda klasa C pojawiająca się w osądach należała do programu $\overline{\mathbf{C}}$.

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

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

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

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

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

Rysunek 3.1: Reguły wnioskowania dla tradycyjnych operatorów logicznych

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

$$*\text{I}\ \frac{\Gamma_1 | P_1 \vdash Q_1 \qquad \Gamma_2 | P_2 \vdash Q_2 \qquad \Gamma_1 \cap \Gamma_2 = \emptyset}{\Gamma_1, \Gamma_2 | P_1 * P_2 \vdash Q_1 * Q_2}$$

$$-\!\!*\text{I}\ \frac{R * P \vdash Q}{R \vdash P -\!\!* Q} \qquad -\!\!*\text{E}\ \frac{\Gamma_1 | R_1 \vdash P -\!\!* Q \qquad \Gamma_2 | R_2 \vdash P \qquad \Gamma_1 \cap \Gamma_2 = \emptyset}{\Gamma_1, \Gamma_2 | R_1 * R_2 \vdash Q}$$

Rysunek 3.2: 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

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

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

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

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

$$\textsc{Ht-field-set}\ \frac{\Gamma \vdash x : C, \quad v : \text{typeof}(C, f)}{\Gamma|S \vdash \{x \neq \texttt{null}\}x.f = v\{\_.x \hookrightarrow f = v\}_\phi}$$

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

$$\textsc{Ht-field-get}\ \frac{\Gamma \vdash x : C, \quad v : \text{typeof}(C, f)}{S \vdash \{x \hookrightarrow f = v\}x.f\{w.w = v\}_\phi}$$

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

Rysunek 3.3: Reguły wnioskowania dla trójek Hoare'a

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

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

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

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

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

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

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

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

Rysunek 3.4: Reguły wnioskowania dla trójek Hoare'a - c.d.

Rozdział 4

Własności ewaluacji

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

4.1. Łączenie ewaluacji

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

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

Dowód. Przez indukcję po długości ciągu konfiguracji $confs'$.

- Jeśli $confs'$ jest ciągiem pustym, to znaczy że $confs ++ confs' = confs$, $h' = h''$ i $\overline{C}' = \overline{C}''$. A zatem, skoro $(h, \overline{C}) \overset{confs}{\rightsquigarrow} (h', \overline{C}')$, to również $(h, \overline{C}) \overset{confs ++ confs'}{\rightsquigarrow} (h'', \overline{C}'')$.
- Załóżmy, że $confs'$ jest niepuste, to znaczy $confs' = confs'' ++ [(h'_n, \overline{C}'_n)]$ Z założenia indukcyjnego wiemy, że istnieje wtedy ewaluacja $(h, \overline{C}) \overset{confs ++ confs''}{\rightsquigarrow} (h'_n, \overline{C}'_n)$. Ale, ponieważ $confs'$ było poprawną ewaluacją $(h', \overline{C}') \overset{confs'}{\rightsquigarrow} (h'', \overline{C}'')$, więc poprawną ewaluacją jest też $(h', \overline{C}') \overset{confs'' ++ [(h'_n, \overline{C}'_n)]}{\rightsquigarrow} (h'', \overline{C}'')$. Stąd, z definicji ewaluacji musi istnieć redukcja $(h'_n, \overline{C}'_n) \to (h'', \overline{C}'')$, a zatem ostatecznie poprawną ewaluacją jest też $(h, \overline{C}) \overset{confs ++ confs'}{\rightsquigarrow} (h'', \overline{C}'')$.

□

4.2. Ewaluacja w rozszerzonym kontekście

Reguły takie jak HT-LET pozwalają wnioskować o ewaluacji złożonych wyrażeń na podstawie założeń o ewaluacji ich podwyrażeń. Dowód ich poprawności wymaga jednak skonstruowania ewaluacji wyrażenia opakowanego w dodatkowy kontekst **let** lub **try** przy założeniu, że istnieje ewaluacja tego wyrażenia bez dodatkowego kontekstu. W tej sekcji sformułuję i udowodnię twierdzenie pozwalające mówić o takich ewaluacjach.

Lemat 1 (O redukcji w rozszerzonym kontekście)**.**
Niech $h, h', \overline{\mathcal{C}}, \overline{\mathcal{C}}', \mathcal{C}, \mathcal{C}', E_1, E_1', A, A'$ *będą takie, że* $(h, \mathcal{C}[\![E_1]\!]_A :: \overline{\mathcal{C}}) \to (h', \mathcal{C}'[\![E_1']\!]_{A'} :: \overline{\mathcal{C}}')$. *Wtedy dla dowolnych* μ, C, x, E_2 *istnieją również następujące redukcje:*

- $(h, [\textbf{let } C\ x = \mathcal{C}[\![E_1]\!]_A \textbf{ in } E_2] :: \overline{\mathcal{C}}) \to (h', [\textbf{let } C\ x = \mathcal{C}[\![E_1']\!]_{A'} \textbf{ in } E_2] :: \overline{\mathcal{C}}')$
- $(h, [\textbf{try } \{\mathcal{C}[\![E_1]\!]_A\} \textbf{ catch } (\mu\ C\ x)\ \{E_2\}] :: \overline{\mathcal{C}}) \to$
  $(h, [\textbf{try } \{\mathcal{C}'[\![E_1']\!]_{A'}\} \textbf{ catch } (\mu\ C\ x)\ \{E_2\}] :: \overline{\mathcal{C}}')$

Dowód. W pierwszej kolejności, rozpatrzmy dwa przypadki: jeśli stos $\overline{\mathcal{C}}$ jest niepusty, redukowany jest pewien redeks z $\overline{\mathcal{C}}$, a zatem ramka $\mathcal{C}[\![E_1]\!]_A$ pozostaje bez zmian. Stąd, także w przypadku rozszerzonego kontekstu redukowany będzie redeks ze stosu $\overline{\mathcal{C}}$, a rozszerzona ramka $[\textbf{let } C\ x = \mathcal{C}[\![E_1]\!]_A \textbf{ in } E_2]$ pozostanie bez zmian.

Załóżmy więc, że stos $\overline{\mathcal{C}}$ jest pusty, a E_1 jest aktualnym redeksem. Zauważmy, że zdecydowana większość przypadków redukcji w ogóle nie rusza kontekstu, w którym znajduje się E_1, a tylko modyfikuje samo wyrażenie, dodaje nowy kontekst lub wrzuca nową ramkę na stos. We wszystkich tych przypadkach, redukcja jest niezależna od zewnętrznych kontekstów, a więc będzie przebiegała tak samo przy rozszerzonym kontekście.

Wyjątkiem są redukcje letgo i letex, powodujące zdjęcie kontekstu **let** oraz redukcje ctchnrml, ctchexok, ctchexnok, powodujące zdjęcie kontekstu **try**. Jeśli jednak redukcja $(h, \mathcal{C}[\![E_1]\!]_A :: \overline{\mathcal{C}}) \to (h', \mathcal{C}'[\![E_1']\!]_{A'} :: \overline{\mathcal{C}}')$ jest jedną z nich, to zdejmowany kontekst musi być najgłębszym kontekstem w $\mathcal{C}$.

Istnieją zatem dokładnie te same redukcje konfiguracji $(h, [\textbf{let } C\ x = \mathcal{C}[\![E_1]\!]_A \textbf{ in } E_2] :: \overline{\mathcal{C}})$ oraz $(h, [\textbf{try } \{\mathcal{C}[\![E_1]\!]_A\} \textbf{ catch } (\mu\ C\ x)\ \{E_2\}] :: \overline{\mathcal{C}})$, w których również zdejmowany jest jeden z kontekstów w $\mathcal{C}$, a zewnętrzny kontekst **let** lub **try** pozostaje niezmieniony. □

Twierdzenie 2 (O ewaluacji w rozszerzonym kontekście)**.**
Niech $h, h', \overline{\mathcal{C}}, \overline{\mathcal{C}}', \mathcal{C}, \mathcal{C}', E_1, E_1', A, A', confs$ *będą takie, że* $confs$ *jest poprawną ewaluacją* $(h, \mathcal{C}[\![E_1]\!]_A :: \overline{\mathcal{C}}) \overset{confs}{\rightsquigarrow} (h', \mathcal{C}'[\![E_1']\!]_{A'} :: \overline{\mathcal{C}}')$. *Wtedy dla dowolnych* μ, C, x, E_2 *istnieją ciągi konfiguracji* $confs', confs''$, *takie że zachodzą również następujące ewaluacje:*

- $(h, [\textbf{let } C\ x = \mathcal{C}[\![E_1]\!]_A \textbf{ in } E_2] :: \overline{\mathcal{C}}) \overset{confs'}{\rightsquigarrow} (h', [\textbf{let } C\ x = \mathcal{C}[\![E_1']\!]_{A'} \textbf{ in } E_2] :: \overline{\mathcal{C}}')$
- $(h, [\textbf{try } \{\mathcal{C}[\![E_1]\!]_A\} \textbf{ catch } (\mu\ C\ x)\ \{E_2\}] :: \overline{\mathcal{C}}) \overset{confs''}{\rightsquigarrow}$
  $(h, [\textbf{try } \{\mathcal{C}'[\![E_1']\!]_{A'}\} \textbf{ catch } (\mu\ C\ x)\ \{E_2\}] :: \overline{\mathcal{C}}')$

Dowód. Przez indukcję po długości $confs$.

- Jeśli $confs$ jest ciągiem pustym, to konfiguracja początkowa i końcowa muszą być sobie równe. Zatem w obu przypadkach rozszerzonego kontekstu otrzymujemy równe konfiguracje początkowe i końcowe, opakowane dodatkowo w kolejny kontekst. Biorąc więc ciągi puste za $confs'$ i $confs''$, dostajemy w obu przypadkach poprawne konfiguracje.
- Jeśli $confs$ jest niepuste i $confs = conf :: confs_{suf}$, to z lematu 1 dostajemy istnienie odpowiednich redukcji dla pierwszego kroku ewaluacji w rozszerzonych kontekstach, a następnie z założenia indukcyjnego dostajemy pozostałą część ewaluacji. Składając je razem, dostajemy oczekiwaną pełną ewaluację.

□

4.3. 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 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.

4.3.1. Izomorfizmy

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

Definicja 4.3.1 (izomorfizm stert)**.**
Niech $h_1, h_2 : \mathsf{Heap}$. Funkcję $f : \mathrm{Dom}(h_1) \cup \{\texttt{null}\} \to \mathrm{Dom}(h_2) \cup \{\texttt{null}\}$ nazwiemy izomorfizmem między tymi stertami, jeśli:

1. f jest bijekcją
2. $f(\texttt{null}) = \texttt{null}$
3. $f(\texttt{npe}) = \texttt{npe}$
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 4.3.2 (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 4.3.3 (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

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

  lub

- $\mathcal{C}_1 = \textbf{try}\ \{\mathcal{C}_1'\}\ \textbf{catch}\ (\mu\ C\ x)\ \{E_1\}, \quad \mathcal{C}_2 = \textbf{try}\ \{\mathcal{C}_2'\}\ \textbf{catch}\ (\mu\ C\ x)\ \{E_2\},$
  a f jest izomorfizmem między $\mathcal{C}_1'$ i $\mathcal{C}_2'$ oraz między E_1 i E_2

□

Definicja 4.3.4 (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.

□

4.3.2. Zależność ewaluacji od zmiennych wolnych

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

Twierdzenie 3 (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

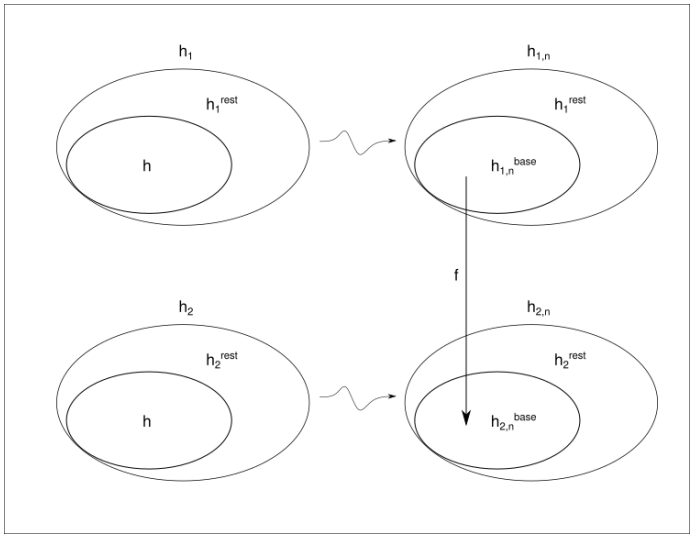
1. $h_{1,n} = h_{1,n}^{base} \oplus h_1^{rest}$
2. $h_{2,n} = h_{2,n}^{base} \oplus h_2^{rest}$
3. *f jest izomorfizmem między $h_{1,n}^{base}$ i $h_{2,n}^{base}$*
4. *f jest identycznością na lokacjach w env*

5. $f(l_1) = l_2$

6. $(h_2, E[/env]) \overset{confs_2}{\rightsquigarrow} (h_{2,n}, A, l_2)$.

Dowód. Na końcu sekcji. □

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 (Rysunek 4.1).



Rysunek 4.1: Wizualizacja twierdzenia o zależności ewaluacji od zmiennych wolnych

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 $confs_1$. W tym celu jednak musimy sformułować następujący lemat, będący krokiem indukcyjnym w uogólnieniu powyższego twierdzenia na częściowe ewaluacje.

Lemat 2 (O zależności redukcji od zmiennych wolnych)**.**
Niech $\overline{\mathcal{C}}_1$ będzie dowolnym stosem 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 $\overline{\mathcal{C}}_1$ znajdują się na stercie h_1^{base}. Niech teraz $h_{1,n}, \overline{\mathcal{C}}_{1,n}$ będą takie, że $(h_1, \overline{\mathcal{C}}_1) \to (h_{1,n}, \overline{\mathcal{C}}_{1,n})$.

Weźmy teraz dowolną spójną stertę h_2^{base}, stos wywołań $\overline{\mathcal{C}}_2$ oraz izomorfizm f, takie że f jest izomorfizmem między h_1^{base} i h_2^{base} oraz między $\overline{\mathcal{C}}_1$ i $\overline{\mathcal{C}}_2$, zdefiniowanym tylko na lokacjach znajdujących się w h_1^{base}. Jeśli wszystkie lokacje występujące w wyrażeniach na stosie $\overline{\mathcal{C}}_2$ znajdują się na stercie 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 $\overline{\mathcal{C}}_{2,n}$ 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 $h^{base}_{1,n}$ i $h^{base}_{2,n}$ oraz między $\overline{\mathcal{C}}_{1,n}$ i $\overline{\mathcal{C}}_{2,n}$*

4. *$h_{1,n} = h^{base}_{1,n} \oplus h^{rest}_1$*

5. *$h_{2,n} = h^{base}_{2,n} \oplus h^{rest}_2$*

6. *wszystkie lokacje występujące w wyrażeniach na stosie $\overline{\mathcal{C}}_{1,n}$ znajdują się na stercie $h^{base}_{1,n}$*

7. *wszystkie lokacje występujące w wyrażeniach na stosie $\overline{\mathcal{C}}_{2,n}$ znajdują się na stercie $h^{base}_{2,n}$*

8. *$(h_2, \overline{\mathcal{C}}_2) \to (h_{2,n}, \overline{\mathcal{C}}_{2,n})$.*

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

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathbf{new}\ \mu\ C(l_1, \ldots, l_k)]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![\mathbf{new}\ \mu\ C(l'_1, \ldots, l'_k)]\!]_\emptyset$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathbf{new}\ \mu\ C(l_1, \ldots, l_k)]\!]_\emptyset) \to (h_{1,n}, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_0]\!]_\emptyset)$, gdzie $\mathsf{alloc}(h_1, \overline{\mathcal{C}}'_1, C) = (l_0, h)$, $\mathsf{flds}(C) = x_1, \ldots, x_k$, $o = \mathsf{empty}_C\{x_1 \mapsto l_1, \ldots, x_k \mapsto l_k\}$, $h_{1,n} = h\{l_0 \mapsto o\}$. Nowa sterta $h_{1,n}$ jest zatem po prostu stertą h_1 z dodatkowym nowym obiektem o pod nową lokacją l_0. Weźmy zatem $h^{base}_{1,n} = h^{base}_1\{l_0 \mapsto o\}$. Spełnia ona warunki 4 i 6, ponieważ jedyną nową lokacją na stosie jest l_0, które trafiło właśnie do $h^{base}_{1,n}$.

  Weźmy teraz $(l'_0, h') = \mathsf{alloc}(h_2, \overline{\mathcal{C}}'_2, C)$, $o' = \mathsf{empty}_C\{x_1 \mapsto l'_1, \ldots, x_k \mapsto l'_k\}$, $h_{2,n} = h'\{l'_0 \mapsto o'\}$. Mamy wtedy $(h_2, \overline{\mathcal{C}}_2) \to (h_{2,n}, \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_0]\!]_\emptyset))$, a zatem biorąc takie $h_{2,n}$ i $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_0]\!]_\emptyset$ mamy spełnione też warunki 7 i 8.

  Jest to jedyny przypadek, w którym na stercie pojawia się nowa lokacja, a zatem należy zmodyfikować izomorfizm f. Niech więc $h^{base}_{2,n} = h^{base}_2\{l'_0 \mapsto o'\}$ 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^{base}_1, a l_0 trafiło do $h^{base}_{1,n}$. Wreszcie, warunek 3 jest spełniony, ponieważ z założenia o izomorfizmie $\overline{\mathcal{C}}_1$ i $\overline{\mathcal{C}}_2$, dla $j = 1, \ldots, k$ mamy $f'(l_j) = l'_j$, a z definicji f', $f'(l_0) = l'_0$.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathbf{let}\ C\ x = e_1\ \mathbf{in}\ e_2]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![\mathbf{let}\ C\ x = e'_1\ \mathbf{in}\ e'_2]\!]_\emptyset$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathbf{let}\ C\ x = e_1\ \mathbf{in}\ e_2]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\mathbf{let}\ C\ x = [\![e_1]\!]_\emptyset\ \mathbf{in}\ e_2])$, a zatem jedyne co się w niej dzieje to dodanie e_2 do kontekstu i zmiana redeksu na e_1.

  Możemy więc wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\mathbf{let}\ C\ x = [\![e'_1]\!]_\emptyset\ \mathbf{in}\ e'_2])$, a sterty i izomorfizm f pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\mathbf{let}\ C\ x = [\![l]\!]_\emptyset\ \mathbf{in}\ e], \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\mathbf{let}\ C\ x = [\![l']\!]_\emptyset\ \mathbf{in}\ e']$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\mathbf{let}\ C\ x = [\![l]\!]_\emptyset\ \mathbf{in}\ e]) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![e\{l/x\}]\!]_\emptyset)$. Podobnie jak wyżej, możemy pozostawić sterty oraz izomorfizm bez zmian i przyjąć $\overline{\mathcal{C}}_{2,n} = (h_2, \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![e'\{l'/x\}]\!]_\emptyset)$.

  Wyrażenia $e\{l/x\}$ i $e'\{l'/x\}$ są izomorficzne, ponieważ z założenia f jest izomorfizmem między e i e' oraz $f(l) = l'$.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathbf{if}\ l_0 == l_1\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![\mathbf{if}\ l'_0 == l'_1\ \mathbf{then}\ e'_1\ \mathbf{else}\ e'_2]\!]_\emptyset$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathbf{if}\ l_0 == l_1\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![e_i]\!]_\emptyset)$, 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 $\overline{\mathcal{C}}_2$ zatem wybierze tę samą gałąź, co redukcja $\overline{\mathcal{C}}_1$. Możemy więc wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![e'_i]\!]_\emptyset$, a sterty i izomorfizm f pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l.m(\overline{l})]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'.m(\overline{l'})]\!]_\emptyset, \quad l \neq \textbf{null}$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}_1 :: \mathcal{C}_1[\![l.m(\overline{l})]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}_1 :: \mathcal{C}_1[\![l.m(\overline{l})]\!]_\emptyset :: [\![e]\!]_\emptyset)$, gdzie e jest ciałem metody m z wartościami $\overline{l}$ podstawionymi w miejsce argumentów.

  Niech teraz e' będzie ciałem metody m z wartościami $\overline{l'}$ podstawionymi w miejsce argumentów. Ponieważ listy argumentów $\overline{l}$ i $\overline{l'}$ są izomorficzne, więc wyrażenia e i e' są izomorficzne.

  Możemy więc wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}_2 :: \mathcal{C}_2[\![l'.m(\overline{l'})]\!]_\emptyset :: [\![e']\!]_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.m(\overline{l})]\!]_\emptyset :: [\![l_2]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_1.m(\overline{l'})]\!]_\emptyset :: [\![l'_2]\!]_\emptyset$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.m(\overline{l})]\!]_\emptyset :: [\![l_2]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_2]\!]_\emptyset)$. Następuje więc jedynie zdjęcie wywołania metody ze stosu wywołań. Możemy zatem wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_2]\!]_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.x = l]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_1.x = l']\!]_\emptyset, \quad l_1 \neq \textbf{null}$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.x = l]\!]_\emptyset) \to (h_{1,n},\ \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l]\!]_\emptyset)$, 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 $l'_1 \neq \textbf{null}$. Weźmy zatem $o' = h_2(l'_1)\{x \mapsto l'\}$, $h_{2,n} = h_2\{l'_1 \mapsto o'\}$ i $h^{base}_{2,n} = h^{base}_2\{l'_1 \mapsto o'\}$.

  Możemy teraz wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l']\!]_\emptyset$ i pozostawić izomorfizm f bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.x]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_1.x]\!]_\emptyset, \quad l \neq \textbf{null}$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.x]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}[\![l_2]\!]_\emptyset)$, gdzie $l_2 = h_1(l_1)(x)$.

  Ponieważ sterty h_1 i h_2 są izomorficzne, więc na stercie 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 $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}[\![l'_2]\!]_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\textbf{throw } l]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![\textbf{throw } l']\!]_\emptyset, \quad l \neq \textbf{null}$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\textbf{throw } l]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l]\!]_D)$, gdzie $\mathsf{class}(h_1, l) = D$, zatem jedynym efektem jest zmiana trybu wykonania na wyjątkowy z wartością wyjątku l i typem D. Możemy więc po prostu wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l']\!]_D$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\textbf{try } \{e_1\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}]\!]_\emptyset, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![\textbf{try } \{e'_1\} \textbf{ catch } (\mu\ C\ x)\ \{e'_2\}]\!]_\emptyset$

  Redukcja $\overline{\mathcal{C}}_1$ jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\textbf{try } \{e_1\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![e_1]\!]_\emptyset\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}])$, 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 $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\textbf{try } \{[\![e'_1]\!]_\emptyset\} \textbf{ catch } (\mu\ C\ x)\ \{e'_2\}])$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![l]\!]_\emptyset\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}], \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\textbf{try } \{[\![l']\!]_\emptyset\} \textbf{ catch } (\mu\ C\ x)\ \{e'_2\}]$

Przy normalnym wykonaniu wyrażenia wewnątrz blocku **try**, jest on po prostu usuwany z kontekstu. Redukcja $\overline{\mathcal{C}}_1$ jest wtedy postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![l]\!]_\emptyset\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}]) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l]\!]_\emptyset)$.

Wystarczy zatem wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l']\!]_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![l]\!]_{C'}\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}], \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\textbf{try } \{[\![l']\!]_{C'}\} \textbf{ catch } (\mu\ C\ x)\ \{e'_2\}]$, gdzie $C' \leq: C$

  Redukcja w tym przypadku oznacza złapanie rzuconego wcześniej wyjątku i obsłużenie go przez e_2. Jest ona postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![l]\!]_{C'}\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}]) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![e]\!]_\emptyset)$, gdzie $e = e_2\{l/x\}$.

  Niech $e' = e'_2\{l'/x\}$. Wtedy wystarczy wziąć $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![e']\!]_\emptyset$, a sterty i izomorfizm pozostawić bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![l]\!]_{C'}\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}], \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\textbf{try } \{[\![l']\!]_{C'}\} \textbf{ catch } (\mu\ C\ x)\ \{e'_2\}]$, gdzie $C' \neq \emptyset, C' \not\leq: C$

  W przypadku niezłapania wyjątku, kontekst **try** jest usuwany, a wyjątek przekazywany dalej. Redukcja jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{try } \{[\![l]\!]_{C'}\} \textbf{ catch } (\mu\ C\ x)\ \{e_2\}]) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l]\!]_{C'})$. Bierzemy zatem $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l']\!]_{C'}$, a sterty i izomorfizm pozostawiamy bez zmian.

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\textbf{let } C\ x = [\![l]\!]_{C'} \textbf{ in } e], \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\textbf{let } C\ x = [\![l']\!]_{C'} \textbf{ in } e'], \quad C' \neq \emptyset$

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

- $\overline{\mathcal{C}}_1 = \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.m(\bar{l})]\!]_\emptyset :: [\![l_2]\!]_C, \quad \overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![l'_1.m(\bar{l'})]\!]_\emptyset :: [\![l'_2]\!]_C, \quad C \neq \emptyset$

  Podobnie w przypadku wyjątku podczas wykonywania metody. Wywołanie jest zdejmowane ze stosu, a wyjątek przekazywany wyżej. Redukcja jest postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![l_1.m(\bar{l})]\!]_\emptyset :: [\![l_2]\!]_C) \to (h_1, \overline{\mathcal{C}}'_1 :: [\![l_2]\!]_C)$.

  Bierzemy $\overline{\mathcal{C}}_{2,n} = (h_1, \overline{\mathcal{C}}'_2 :: [\![l'_2]\!]_C)$, a sterty i izomorfizm pozostawiamy bez zmian.

- Redukcja $\overline{\mathcal{C}}_1$ powoduje odwołanie do lokacji **null**

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

  Redukcja $\overline{\mathcal{C}}_1$ jest wtedy postaci $(h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![e]\!]_\emptyset) \to (h_1, \overline{\mathcal{C}}'_1 :: \mathcal{C}_1[\![\mathsf{npe}]\!]_{\mathtt{NPE}})$.

  Wówczas $\overline{\mathcal{C}}_2 = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![e']\!]_\emptyset$, 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 $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}'_2 :: \mathcal{C}_2[\![\mathsf{npe}]\!]_{\mathtt{NPE}}$, a sterty i izomorfizm pozostawić bez zmian.

□

Korzystając z tego lematu w kroku indukcyjnym, udowodnimy teraz następujący lemat, mówiący o ewaluacjach dowolnej długości.

Lemat 3 (O zależności częściowej ewaluacji od zmiennych wolnych)**.**
Niech $\overline{\mathcal{C}}_1$ będzie dowolnym stosem 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

$\overline{\mathcal{C}}_1$ *znajdują się na stercie* h_1^{base}*. Niech teraz* $h_{1,n}, \overline{\mathcal{C}}_{1,n}, confs_1$ *będą takie, że* $(h_1, \overline{\mathcal{C}}_1) \overset{confs_1}{\rightsquigarrow} (h_{1,n}, \overline{\mathcal{C}}_{1,n})$.

Weźmy teraz dowolną spójną stertę h_2^{base}*, stos wywołań* $\overline{\mathcal{C}}_2$ *oraz izomorfizm* f*, takie że* f *jest izomorfizmem między* h_1^{base} *i* h_2^{base} *oraz między* $\overline{\mathcal{C}}_1$ *i* $\overline{\mathcal{C}}_2$*, zdefiniowanym tylko na lokacjach znajdujących się w* h_1^{base}*. Jeśli wszystkie lokacje występujące w wyrażeniach na stosie* $\overline{\mathcal{C}}_2$ *znajdują się na stercie* 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* $\overline{\mathcal{C}}_{2,n}$*, izomorfizm* f' *oraz ciąg konfiguracji* $confs_2$*, 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* $\overline{\mathcal{C}}_{1,n}$ *i* $\overline{\mathcal{C}}_{2,n}$
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* $\overline{\mathcal{C}}_{1,n}$ *znajdują się na stercie* $h_{1,n}^{base}$
7. *wszystkie lokacje występujące w wyrażeniach na stosie* $\overline{\mathcal{C}}_{2,n}$ *znajdują się na stercie* $h_{2,n}^{base}$
8. $(h_2, \overline{\mathcal{C}}_2) \overset{confs_2}{\rightsquigarrow} (h_{2,n}, \overline{\mathcal{C}}_{2,n})$.

Dowód. Przez indukcję po długości ciągu konfiguracji $confs_1$.

- Jeśli $confs_1 = [\,]$ jest pustym ciągiem, to znaczy że $h_{1,n} = h_1$ oraz $\overline{\mathcal{C}}_{1,n} = \overline{\mathcal{C}}_1$. Możemy wtedy wziąć $h_{1,n}^{base} = h_1^{base}$, $h_{2,n}^{base} = h_2^{base}$, $h_{2,n} = h_2$, $\overline{\mathcal{C}}_{2,n} = \overline{\mathcal{C}}_2$, $f' = f$ oraz $confs_2 = [\,]$.
- Jeśli $confs_1 = confs'_1 :: (h_{1,n-1}, \overline{\mathcal{C}}_{1,n-1})$, to Z założenia indukcyjnego będziemy mieli sterty $h_{1,n-1}^{base}, h_{2,n-1}^{base}, h_{2,n-1}$, stos $\overline{\mathcal{C}}_{2,n-1}$, izomorfizm f' oraz ciąg konfiguracji $confs'_2$, takie że spełnione są warunki z treści zadania, w szczególności $(h_2, \overline{\mathcal{C}}_2) \overset{confs'_2}{\rightsquigarrow} (h_{2,n-1}, \overline{\mathcal{C}}_{2,n-1})$. Z lematu 2 dostaniemy sterty $h_{1,n}^{base}, h_{2,n}^{base}, h_{2,n}$, stos $\overline{\mathcal{C}}_{2,n}$ oraz izomorfizm f'', takie że spełnione są warunki 1-7 oraz $(h_{2,n-1}, \overline{\mathcal{C}}_{2,n-1}) \to (h_{2,n}, \overline{\mathcal{C}}_{2,n})$. Składając tę redukcję z ewaluacją z założenia indukcyjnego, dostaniemy, że w istocie $(h_2, \overline{\mathcal{C}}_2) \overset{confs_2}{\rightsquigarrow} (h_{2,n}, \overline{\mathcal{C}}_{2,n})$.

□

Możemy wreszcie udowodnić twierdzenie 3

Dowód. (twierdzenia o zależności ewaluacji od zmiennych wolnych) Niech h, env, E, h_1, h_2, h_1^{rest}, h_2^{rest}, $confs_1, h_{1,n}, A, l_1$ będą jak w treści twierdzenia. Pokażę, że istnieją istnieją $h_{1,n}^{base}, h_{2,n}^{base}, confs_2, h_{2,n}, l_2, f$, spełniające warunki z treści.

W tym celu zastosujemy lemat 3 dla $h_1^{base} = h_2^{base} = h$, ewaluacji $(h_1, [\![E[/env]]\!]_\emptyset) \overset{confs_1}{\rightsquigarrow} (h_{1,n}, [\![l_1]\!]_A)$ i $\overline{\mathcal{C}}_2 = [\![E[/env]]\!]_\emptyset$. Żeby jednak móc go zastosować, musimy pokazać że wszystkie lokacje z wyrażenia $E[/env]$ występują na stercie h. Wynika to jednak wprost z założeń, że w wyrażeniu E nie ma konkretnych lokacji, a wszystkie zmienne wolne mapowane są przez env na lokacje z h. Stąd, po podstawieniu za zmienne wolne lokacji z env, założenie jest spełnione.

Ostatnim problemem jest wybranie odpowiedniego początkowego f. Wystarczy jednak, żeby f była identycznością na **null**, npe i lokacjach występujących w h lub $E[/env]$. Ponieważ h jest spójna, więc tak zdefiniowana f jest w oczywisty sposób automorfizmem na h oraz na $E[/env]$. □

Podczas wnioskowania o izomorficznych stertach przydatny będzie jeszcze następujący lemat, mówiący że ewaluacje izomorficznych stert są izomorficzne.

Lemat 4 (O ewaluacji izomorficznych stert)**.**
Niech h_1, h_2 będą izomorficznymi stertami, E_1 i E_2 izomorficznymi wyrażeniami, a f izomorfizem między nimi. Wtedy dla dowolnych h_1', l_1, A, jeśli $(h_1, E_1) \rightsquigarrow (h_1', l_1, A)$, to istnieją h_2', l_2, f', takie że

1. *f' rozszerza f*
2. *f' jest izomorfizmem między h_1' i h_2'*
3. *$f(l_1) = l_2$*
4. *$(h_2, E_2) \rightsquigarrow (h_2', l_2, A)$*

Dowód. Dowód przebiega przez indukcję po długości ewaluacji. W kroku indukcyjnym należy zapewnić zachowanie wszystkich własności z tezy lematy, co wygląda analogicznie jak w dowodzie lematu 2 (O zależności redukcji od zmiennych wolnych). □

Rozdział 5

Równoważność stert i środowisk względem semantyki

W tym rozdziale przedstawimy kilka twierdzeń dotyczących równoważności pewnych stert i środowisk przy pewnych założeniach.

Definicja 5.0.1 (Równoważność stert i środowisk)**.**
Powiemy, że sterty h_1 i h_2 oraz środowiska env_1 i env_2 są równoważne dla termu P, jeśli zachodzi $[\![P]\!]_{h_1,env_1} = [\![P]\!]_{h_2,env_2}$.

Jeśli w danym zastosowaniu $h_1 = h_2$ lub $env_1 = env_2$, to będziemy je pomijać, mówiąc po prostu o równoważności stert lub środowisk. □

Dowód wszystkich przedstawianych równoważności przebiega w ten sam sposób – przez indukcję po budowie termu P. Najpierw jednak zauważmy, że przypadki w których P jest równe `True` lub `False`, albo jest koniunkcją, alternatywą lub implikacją są prawdziwe niezależnie od konkretnych warunków na h_1, h_2, env_1 i env_2, i równoważnoćć w tych przypadkach wynika wprost z założeń indukcyjnych dla ich podwyrażeń.

Lemat 5 (Równoważność izomorficznych stert i środowisk)**.**
Jeśli f jest izomorfizmem pomiędzy stertami h_1 i h_2, oraz między środowiskami env_1 i env_2, to te sterty oraz środowiska są równoważne dla dowolnego termu.

Dowód. Zgodnie z zastrzeżeniem powyżej, wystarczy pokazać tę równoważność dla równości, trójek Hoare'a i operatorów separacyjnych.

- „$v_1 = v_2$" i „$v_1 \hookrightarrow f = v_2$" – równoważność wynika wprost z semantyki tych operatorów i zachowania kompozycji przez izomorfizm.
- „$*$" i „$-\!\!*$" – równoważność wynika wprost z założeń indukcyjnych dla podwyrażeń
- trójki Hoare'a – równoważność wynika z lematu 4 (O ewaluacji izomorficznych stert) i założeń indukcyjnych dla warunku początkowego i końcowego trójki Hoare'a: równoważność warunków początkowych w obu stertach mamy od razu. Ponieważ env_1 i env_2 są izomorficzne, więc również $E[/env_1]$ i $E[/env_2]$, a zatem oba te wyrażenia ewaluują się do izomorficznych stert. A zatem możemy skorzystać też z założenia indukcyjnego dla warunków końcowych na tych nowych stertach.

□

Następujące twierdzenie jest użyteczne w dowodzie poprawności reguły dla wywołania funkcji. Mówi ona, że podstawienie lokacji za zmienną w wyrażeniu ma taki sam efekt, jak dodanie mapowania tej zmiennej na lokację w środowisku i pozostawienie wyrażenia bez zmian.

Lemat 6 (Równoważność podstawienia zmiennych i dodania ich do środowiska)**.**
Dla dowolnego wyrażenia P, środowiska env i sterty h zachodzi równość $[\![P]\!]_{h,env[x\mapsto l]} = [\![P[l/x]]\!]_{h,env}$

Dowód. Przez indukcję po budowie termu P. Równoważność wszystkich przypadków poza równościami i trójkami Hoare'a jest oczywista i wynika wprost z założeń indukcyjnych dla podwyrażeń.

- „$v_1 = v_2$" i $v_1 \hookrightarrow f = v_2$ – w tym przypadku należy rozpatrzeć jakiej postaci są wartości v_1 i v_2. Przypadki dla **this** i **null** oraz dla zmiennej innej niż x są proste, bo wartości te są identyczne w obu przypadkach. Jeśli któraś (lub obie) jest zmienną x, to semantyką tej wartość w przypadku $P[l/x]$ jest po prostu l, natomiast w przypadku P jest nią $env[x \mapsto l](x)$, a zatem również l.
- W przypadku trójki Hoare'a jest jeszcze prościej, wyrażenia ewaluowane są bowiem w obu trójkach identyczne. W pierwszym przypadku jest to $E[/env[x \mapsto l]]$, natomiast w drugim $E[l/x][/env]$. Przez prostą indukcję po budowie E można udowonić, że wyrażenia te są sobie równe.

□

Kolejne twierdzenie to przeniesienie twierdzenia 3 (O zależności ewaluacji od zmiennych wolnych) z ewaluacji na semantykę termów. Mówi ono, że dla semantyki termu istotne są tylko te elementy sterty i środowiska, które odpowiadają zmienny, wolnym w tym termie.

Lemat 7 (Równoważność rozszerzonej sterty i środowiska)**.**
Weźmy P, h_1, i env_1 takie, że wszystkie zmienne wolne w P są przez env_1 mapowane na h_1. Wtedy dla dowolnej sterty h_2 rozszerzającej h_1 i dowolnego środowiska env_2 rozszerzającego env_1, sterty h_1 i h_2 oraz środowiska env_1 i env_2 są równoważne dla P.

Dowód. Ponownie, jedynymi interesującymi przypadkami są równości, operatory separacyjne i trójki Hoare'a.

- „$v_1 = v_2$" i „$v_1 \hookrightarrow f = v_2$" – jeśli v_1, lub v_2 nie są zmiennymi, to semantyka tych wartości jest w oczywisty sposób taka sama. Jeśli natomiast któreś z nich jest zmienną, to jest to zmienna wolna, a zatem z założenia musi znaleźć się w env_1 i być mapowana na h_1. A ponieważ env_2 i h_2 rozszerzają odpowiednio env_1 i h_1, a zatem semantyka wartości dla env_2 i h_2 jest również taka sama, jak dla env_1 i h_1.
- „$*$" i „$-\!\!*$" – z założenia indukcyjnego możemy nadmiarową część sterty dołożyć do dowolnego z operandów bez zmiany semantyki. Zauważmy, że powstała w ten sposób sterta nie musi być spójna (na nadmiarowej części mogą znajdować się obiekty zawierające w polach lokacje z obu rozłącznych części), ale to nie przeszkadza, gdyż nigdy nie odwołamy się do obiektu pod żadną lokacją z tej nadmiarowej części.
- trójki Hoare'a – odpowiednia ewaluacja wynika z twierdzenia 3 (O zależności ewaluacji od zmiennych wolnych), a równoważność semantyki dla warunków końcowych trójek Hoare'a wynika z założenia indukcyjnego i lematu 5 (O równoważności izomorficznych stert i środowisk).

□

Szczególnym przypadkiem powyższego lematu jest następujący, mówiący że możemy bezkarnie dodawać świeże zmienne do środowiska. Dzięki słabszym założeniom jest on łatwiejszy w stosowaniu.

Lemat 8 (Równoważność środowiska po dodaniu świeżej zmiennej)**.**
Jeśli x nie jest zmienną wolną w P, a env zawiera wszystkie zmienne wolne z P i mapuje je na lokacje z h, to dla dowolnej lokacji l środowiska env i $env[l/x]$ są równoważne dla termu P.

Dowód. Teza wynika wprost z lematu 7(O równoważności rozszerzonej sterty i środowiska) po wzięciu $env_1 = env$, $env_2 = env[l/x]$ i $h_1 = h_2$. □

Rozdział 6

Poprawność

Głównym twierdzeniem, dowodzonym w ramach niniejszej pracy, jest poprawność prezentowanej logiki. Mówi ono, że jeśli możemy udowodnić pewne wynikanie $\Gamma|P \vdash Q$, to każda sterta spełniająca P będzie również spełniała Q. Dla ścisłego sformułowania tego twierdzenia potrzebujemy jednak kilku dodatkowych założeń o stertach i programach, o których chcemy wnioskować.

6.1. Konieczne założenia

Będziemy rozważać jedynie spójne sterty. Ponieważ język Jafun jest silnie typowany, więc potrzebne jest także założenie o odpowiednim otypowaniu obiektów na stercie. Powiemy więc, że środowisko *env* zgadza się ze środowiskiem typów Γ na stercie h, jeśli mają te same dziedziny i typ każdej zmiennej w Γ zgadza się z jej typem na stercie h. To znaczy, dla każdego x, jesli $\Gamma \vdash x : C$, to obiekt $h(env(x))$ jest typu C.

W twierdzeniu o poprawności będziemy wymagać, żeby rozważane środowisko było zgodne z Γ na stercie, o której będziemy wnioskować. W połączeniu z założeniem o tym, że wszystkie zmienne wolne pojawiające się w osądach muszą być w Γ, zapewni to, że wszystkie zmienne wolne są mapowane przez środowisko na pewną lokację istniejącą na stercie i, co więcej, obiekt znajduący się pod tą lokacją jest odpowiedniego typu.

Wreszcie, należy poczynić odpowiednie założenia o niezmiennikach. Lista niezmienników została wprowadzona w rozdziale 3 jako lista elementów postaci $(C, m, \{P\} \cdot \{w.Q\}_A)$. Każdy taki niezmiennik oznaczał założenie o własności ewaluacji metody m na obiektach klasy C i jej pochodnych. Żeby jednak takie założenie było poprawne, należy zadbać o to, żeby każdy taki niezmiennik dał się wyprowadzić korzystając z reguł dowodzenia i niezmienników będących wcześniej na liście. Pozwoli to udowodnić poprawność każdego z nich przez prostą indukcję po długości listy niezmienników.

Zdefiniujmy więc *udowodnioną listę niezminników* indukcyjnie, w następujący sposób. Powiemy, że pusta lista niezmienników jest udowodniona. Niepusta lista niezmienników $\texttt{invariants} :: (C, m, \{P\} \cdot \{w.Q\}_A)$ dla programu $\overline{\mathbf{C}}$ jest udowodniona wtedy i tylko wtedy, gdy udowodniona jest lista `invariants` oraz osąd $\overline{\mathbf{C}}, \texttt{invariants}, \Gamma|\texttt{True} \vdash \{P\}E\{w.Q\}_A$ jest wyprowadzalny dla każdego E będącego ciałem metody m w klasie C lub jej pochodnej i dla Γ będącej środowiskiem typów przypisującym nazwom argumentów metody m ich typy, a wartości **this** typ C.

Zauważmy, że warunek dodania niezmiennika do listy jest bardzo silny – należy udowodnić jego prawdziwość nie tylko dla metody w interesującej nas klasie, ale również dla każdego jej przeciążonego odpowiednika w klasach pochodnych. Z jednej strony ogranicza to zakres

wnioskowania o metodach, ale z drugiej wymusza pewną dyscyplinę podczas przeciążania metod. Definiując metodę z intencją jej przeciążenia należy bowiem zastanowić się, jakie własności powinna ona spełniać we wszystkich klasach pochodnych. Dzięki temu można zapewnić, że metoda ta we wszystkich klasach dziedziczących zachowuje się w podobny, dobrze określony sposób.

6.2. Twierdzenie o poprawności

Możemy teraz ściśle sformułować twierdzenie o poprawności.

Twierdzenie 4 (O poprawności logiki separacji dla języka Jafun)**.**
Niech $\overline{\mathbf{C}}$ *będzie programem języka Jafun i niech* `invariants` *będzie udowodnioną listą niezmienników. Niech* P, Q *będą dowolnymi termami logiki separacji, takimi że* $\Gamma | P \vdash Q$*. Wtedy dla każdej spójnej sterty* h *i środowiska* env*, takich że* env *zgadza się z* Γ *na stercie* h *mamy* $[\![P]\!]_{h,env} = [\![Q]\!]_{h,env}$*.* □

Dowód tego twierdzenia przebiega w pierwszej kolejności przez indukcję po długości listy niezmienników, a następnie przez indukcję po budowie drzewa dowodu dla $\Gamma | P \vdash Q$. Najpierw należy pokazać, poprawność przypadków bazowych, czyli udowodnić tezę twierdzenia dla wszystkich reguł niezawierających osądów w przesłankach. Następnie należy pokazać, że wszystkie pozostałe reguły zachowują poprawność. W tym celu będziemy zakładać, że wszystkie osądy z przesłanek są poprawne, a następnie, korzystając z tych założeń, będziemy dowodzić poprawność osądu, będącego wnioskiem reguły. Takie założenie jest możliwe dzięki temu, że drzewa dowodów przesłanek są właściwymi poddrzewami dowodu dla wniosku, a zatem założenie o poprawności przesłanek jest założeniem indukcyjnym.

Dowód tego twierdzenia w systemie Coq znajduje się w pliku `JaIrisSoundeness.v` jako dowód twierdzenia `JFIOuterSoundness`.

6.3. Poprawność reguł dla tradycyjnych operatorów logicznych

Dowody poprawności reguł przedstawionych na rysunku 3.1 są do siebie bardzo podobne i, oprócz kilku szczegółów technicznych, mało interesujące, jako że wprost wynikają z własności operatorów logicznych. Na przykład, poprawność reguły Asm jest oczywista – każda sterta spełniająca P spełnia P.

Dowód poprawności innych reguł jednak wymaga doprecyzowania. Poprawność reguły Trans wymaga pokazania, że zmienna v istnieje w środowisku env. Jest to prawdą, ponieważ, z faktu, że v jest zmienną wolną w osądzie, wynika, że v ma przypisany typ w środowisku typów Γ. Istnienie v w środowisku env wynika więc z założenia o zgodności env i Γ

Kolejną regułą, której poprawność jest nieoczywista, jest reguła $\exists$I, wprowadzająca kwantyfikator $\exists$. Należy tu pokazać, że, zakładając poprawność przesłanki, każda sterta spełniająca Q spełnia też $\exists x : C.P$. Kluczową rolę gra tutaj wartość v, którą podstawiamy za zmienną x. Należy rozpatrzeć przypadki, w których v jest równa **null**, **this** lub jest zmienną. Następnie możemy dodać do środowiska mapowanie x na odpowiednio **null**, env(**this**) lub $env(v)$. Powstałe środowisko jest wtedy zgodne ze środowiskiem $(\Gamma, x : C)$ na stercie h, a zatem można po prostu zastosować założenie indukcyjne.

Wreszcie, najbardziej skomplikowanym przypadkiem, spośród reguł dla klasycznych operatorów, jest poprawność reguły $\exists$E, eliminującej kwantyfikator $\exists$.

Kluczowym krokiem w dowodzie poprawności $\exists$E jest zauważenie, że dodanie świeżej, nie będącej zmienną wolną w termie, zmiennej do środowiska nie zmienia spełnialności tego

termu przez stertę. To znaczy jeśli $x \notin FV(P)$, to $[\![P]\!]_{h,env} = [\![P]\!]_{h,env[x \mapsto l]}$. Formalny dowód tego faktu wymaga indukcji po budowie termu P, ale intuicyjnie jest to prawda, ponieważ skoro x nie jest wolne w P, więc P nie ma możliwości odwoływania się do niego, a zatem podczas obliczania semantyki P w środowisku env nigdy nie odwołamy się do lokacji $env(x)$. Nie ma zatem znaczenia, czy x występuje w środowisku env i, jeśli tak, to jaka jest wartość $env(x)$.

Kiedy udowodnimy ten fakt, reszta dowodu przebiega już w prosty sposób: mamy stertę h oraz środowisko env zgadzające się na h z Γ, takie że h spełnia term R w środowisku env. Skoro tak, to z założenia indukcyjnego dla pierwszej przesłanki mamy, że h spełnia też w tym środowisku term $\exists x : C.P$, a zatem istnieje taka lokacja l, że $h(l)$ jest typu C i h spełnia P w środowisku $env[x \mapsto l]$.

Możemy więc teraz zastosować założenie indukcyjne dla drugiej przesłanki. Jak zauważyliśmy przed chwilą, $env[x \mapsto l]$ zgadza się z $(\Gamma, x : C)$ na stercie h oraz h spełnia P w tym środowisku. Pozostaje pokazać, że h spełnia R w tym środowisku. W tym momencie korzystamy z obserwacji o zachowaniu spełniania przy dodaniu świeżej zmiennej. Wiemy, że h spełnia R w środowisku env, oraz że x jest świeże w R (bo x nie jest w Γ, a wszystkie zmienne wolne z R są). h spełnia więc też R w środowisku $env[x \mapsto l]$, a zatem z założenia indukcyjnego dla drugiej przesłanki, h spełnia Q w środowisku $env[x \mapsto l]$.

Możemy teraz ponownie skorzystać z faktu o świeżych zmiennych, bo, podobnie jak w R, x jest świeże w Q. Ostatecznie dostajemy więc, że h spełnia Q w środowisku env, a zatem reguła $\exists$E jest poprawna.

6.4. Poprawność reguł dla operatorów separacji

Poprawność reguł Sep-assoc i Sep-sym wynika w prosty sposób z własności sterty, takich jak przemienność i łączność operatora $\oplus$ oraz faktu że suma spójnych stert jest spójna. Poprawność pozostałych reguł jest nieco bardziej skomplikowanym zagadnieniem.

6.4.1. Reguły eliminacji i wprowadzania operatorów separacji

Dowód poprawności eliminacji i wprowadzania operatorów separacji nie jest skomplikowany, ale kryje się w nim kilka subtelności związanych z manipulowaniem środowiskami, tak żeby było możliwe skorzystanie z nich używając założeń indukcyjnych. Pokażę je na przykładzie reguły $*$I, ale podobne rozumowanie jest też stosowane w pozostałych regułach.

Aby pokazać poprawność tej reguły, bierzemy dowolne spójne h spełniające $P_1 * P_2$ i środowisko env zgadzające się z (Γ_1, Γ_2) na h. Ponieważ h spełnia $P_1 * P_2$, więc istnieje podział h na rozłączne sterty h_1 i h_2, spełniające odpowiednio P_1 i P_2. Chcemy teraz skorzystać z założeń indukcyjnych, żeby otrzymać, że spełniają one też odpowiednio Q_1 i Q_2, co zakończy dowód.

Nie możemy jednak zrobić tego używając środowiska env, bo nie musi ono zgadzać się z Γ_1 ani Γ_2 – jest na to zbyt duże. O ile oba ze środowisk typów są niepuste, to env zawiera zarówno zmienne niebędące w Γ_1, jak i zmienne niebędące w Γ_2. Należy więc ostrożnie wziąć podzbiór środowiska env, zgodny z Γ_1 na h_1 i drugi, zgodny z Γ_2 na h_2 w taki sposób, żeby zachować spełnianie termów przez sterty h_1 i h_2 w tych okrojonych środowiskach.

Aby to osiągnąć, wystarczy wziąć env_1 i env_2 będące obcięciami środowiska env odpowiednio do zmiennych znajdujących się w Γ_1 i Γ_2. Wtedy oczywiście env_1 zgadza się z Γ_1 na h_1 i analogicznie dla env_2. Pozostaje więc pokazać, że h_1 spełnia P_1 w środowisku env_1. W tym celu możemy znów skorzystać z faktu, że dodawanie świeżych zmiennych do środowiska

nie wpływa na spełnianie termu przez stertę. Ponieważ wszystkie zmienne wolne w P_1 znajdują się w Γ_1, więc znajdują się też w env_1. Możemy więc otrzymać env przez dodawanie do env_1 kolejnych zmiennych, z których każda jest świeża w P_1, a zatem dodanie żadnej z nich nie zmienia spełniania P_1 przez h_1. Stąd mamy więc $[\![P_1]\!]_{h_1,env} = [\![P_1]\!]_{h_1,env_1}$ i podobnie $[\![Q_1]\!]_{h_1,env} = [\![Q_1]\!]_{h_1,env_1}$.

Korzystając więc z powyższych równości i z założenia indukcyjnego, dostajemy, że h_1 spełnia Q_1 w środowisku env. Przeprowadzenie analogicznego rozumowania dla h_2 kończy dowód.

6.4.2. Reguła osłabiania

Reguła osłabiania *-Weak jest jedną z dwóch reguł, obok reguły Ht-frame, dzięki którym operatory separacji są tak przydatne. Sprawia ona, że logika separacji jest afiniczna, to znaczy że jeśli sterta spełnia pewien term, to każda jej spójna nadsterta też go spełnia.

Istotnie, skoro mamy $P_1 * P_2 \vdash P_1$ dla dowolnych P_1, P_2, to po przyjęciu $P_2 = \texttt{True}$ możemy w pewnym sensie zapomnieć o niepasującej nam części sterty. Jeśli mamy więc stertę h_1 spełniającą term P_1, to dla dowolnej sterty h_2 mamy $h_1 \oplus h_2$ spełnia $P_1 * \texttt{True}$, a zatem, korzystając z reguły osłabiania, także $h_1 \oplus h_2$ spełnia P_1.

Jest to niewątpliwie silna własność logiki, jednak zachować jej poprawność przy zadanej semantyce języka, niezbędne były pewne kompromisy. Przede wszystkim, logika ta nie zawiera kwantyfikatora ogólnego $\forall$. Przy naszej definicji sterty kwantyfikator ten zaburzyłby poprawność reguły osłabiania. Bez dodatkowych założeń nie ma bowiem gwarancji, że $h_1 \oplus h_2$ spełnia $\forall x.P$, nawet jeśli samo h_1 spełnia $\forall x.P$. Podczas rozszerzania sterty moglibyśmy dodać nowe lokacje, niespełniające P, i w ten sposób zepsuć poprawność tej reguły.

Jednocześnie, niezbędne jest też zadbanie, żeby nie było możliwe symulowanie kwantyfikatora ogólnego przy pomocy innych konstrukcji logiki. Możliwe byłoby to na przykład przez zastąpienie termu $\forall x : C.P$ przez $\neg\exists x : C.\neg P$. Z tego powodu kwantyfikator $\exists$ został w logice ograniczony do termów najwyższego poziomy, tak że niemożliwe staje się jego zanegowanie.

Te dwa ograniczenia wystarczają do zapewnienia poprawności reguły osłabiania przy zachowaniu semantyki języka bez zmian.

Wciąż jednak, pomimo tych ograniczeń, dowód poprawności reguły osłabiania nie jest trywialny. Wynika ona wprost z pomocniczego lematu 8 [O równoważności środowiska po dodaniu świeżej zmiennej], mówiącego że jeśli pewne środowisko mapuje wszystkie zmienne wolne termu na lokacje na pewnej stercie, to możemy dowolnie rozszerzać zarówno środowisko, jak i stertę bez zmiany semantyki tego termu. To znaczy, jeśli $h_1 \subseteq h$, $env_1 \subseteq env$ i wszystkie zmienne wolne z termu P mapowane są przez env_1 na lokacje w h_1, to $[\![P]\!]_{h_1,env_1} = [\![P]\!]_{h,env}$.

Twierdzenie to, mimo że wygląda podobnie to przytaczanego wcześniej faktu o rozszerzaniu środowiska o świeże zmienne, jest od niego istotnie trudniejsze do udowodnienia, ponieważ tutaj chcemy zagwarantować także możliwość rozszerzania sterty. Powoduje to utrudnienia w dowodzie dla przypadku trójki Hoare'a. W pozostałych przypadkach dowód nadal nie jest trudny. Dzięki temu, że operator $*$ może występować jedynie w wyrażeniach wewnętrznych, nie musimy przejmować się kwantyfikatorem $\exists$.

W przypadku trójek Hoare'a jednak musimy skorzystać z udowodnionego wcześniej twierdzenia 3 (O zależności ewaluacji od zmiennych wolnych). Zapewnia nam ono, że dla dowolnej ewaluacji na stercie h_1 będziemy mieli odpowiednią ewaluację na stercie h i, co więcej, otrzymane na końcu sterty będą izomorficzne. Pozostaje więc pokazać, że dla semantyka termu dla izomorficznych stert jest taka sama. Jest to prawda tylko przy założeniu, że środowiska, w których rozpatrujemy semantykę, również są izomorficzne, przynajmniej na zmiennych

wolnych w tym termie. Na szczęście jednak twierdzenie 3 zapewnia nam, że otrzymany izomorfizm jest identycznością na lokacjach przypisanych w środowisku, a zatem wyjściowe środowiska *env* i env_1 spełniają to założenie.

6.5. Poprawność reguł dla trójek Hoare'a

Najliczniejszą grupę reguł wnioskowania stanowią, przedstawione na rysunkach 3.3 i 3.4, reguły dla trójek Hoare'a. Większość z nich jest bardzo prosta, a ich poprawność wynika wprost z semantyki języka. Do takich należą HT-NEW-NULL, gwarantująca że alokacja obiektu zawsze się powiedzie i zwróci niepustą lokację, czy HT-NULL-GET, mówiąca że próba odczytu pola obiektu pod lokacją **null** zakończy się rzuceniem wyjątku npe.

Z drugiej strony, dowody poprawności niektórych z nich są skomplikowane lub zawierają subtelne niuanse. O takich przykładach opowiem poniżej.

Jednym z motywów powtarzających się w wielu regułach dla trójek Hoare'a jest założenie o trwałości niektórych termów. Zazwyczaj oznacza to, że w dowodzie poprawności potrzebujemy założenia o spełnianiu takiego termu przez pewną stertę, przy założeniu, że spełnia go inna sterta. Ponieważ termy trwałe nie opisują w żaden sposób własności sterty, więc dowolne dwie sterty są równoważne, jeśli chodzi o spełnianie danego trwałego termu.

6.6. Reguły strukturalne

Dzięki regule HT-FRAME możemy w pełni docenić operator separacji $*$. Upraszcza ona dowodzenie faktów o trójkach Hoare'a na złożonych stertach, sprawiając, że możliwe jest dowodzenie własności, ograniczając się tylko do istotnych dla nich fragmentów sterty, a następnie rozszerzanie tych fragmentów to interesującej nas całości. Dzięki takiemu podejściu możliwe jest ograniczenie założeń w warunkach początkowych trójki Hoare'a do niezbędnego minimum.

Dowód poprawności reguły HT-FRAME przebiega podobnie do dowodu dla reguły osłabiania i również korzysta z twierdzenia 3 (O zależności ewaluacji od zmiennych wolnych).

Reguła HT-CSQ pozwala na wzmacnianie warunków początkowych i osłabianie warunków końcowych trójek Hoare'a. Niuansem jest w niej założenie o trwałości S. Jest ono używane w dowodzie poprawności, kiedy chcemy skorzystać z założenia indukcyjnego dla trzeciej przesłanki. Konkretnie, po ewaluacji wyrażenia E otrzymamy stertę, o której wiemy, że spełnia Q'. Założenie indukcyjne dla trzeciej przesłanki pozwoli nam stwierdzić, że o ile spełnia ona S, to spełnianie Q' implikuje spełnianie Q. Jednak, ponieważ nie wiemy nic szczególnego o wyrażeniu E, więc nie możemy bez dodatkowych założeń stwierdzić, że sterta po ewaluacji spełnia S. Skoro jednak S jest trwałe i sterta przed ewaluacją spełniała S, więc każda inna sterta, w szczególności ta po ewaluacji, spełnia S.

6.7. Reguły opisujące konstrukcje języka

Zbiór reguł opisujących konstrukcje języka najbardziej odróżnia się od reguł logiki Iris, opisującej język $\lambda_{\text{ref,conc}}$, będący nieotypowanym językiem funkcyjnym, opartym na rachunku lambda, z operatorami pozwalającymi na dostęp do sterty i równoległość.

Niniejsza logika jest pod tym względem istotnie różna od logiki Iris. Język jafun jest imperatywnym, silnie typowanym językiem obiektowym. Jego konstrukcje różnią się zatem od konstrukcji $\lambda_{\text{ref,conc}}$ zarówno pod względem składni, jak i semantyki.

6.7.1. Reguły dla wyrażeń prostych

Reguły HT-NEW-*, HT-FIELD-GET i HT-FIELD-SET w prosty sposób opisują działanie alokacji i dostępu do pól obiektów. Poprawność ich wszystkich wynika wprost z semantyki języka i logiki. Jedynym nieoczywistym krokiem w dowodzie jest zapewnienie, że odpowiednie zmienne istnieją w środowisku i są mapowane na obiekty odpowiedniego typu na stercie. Jest to jednak zapewnianie przez założenia, że wszystkie zmienne wolne w wyrażeniach należą do Γ, a środowisko *env* zgadza się z Γ na rozpatrywanej stercie.

Reguły HT-NULL-* i HT-THROW różnego rodzaju sytuacje w trakcie ewaluacji programu, których może zostać rzucony wyjątek. Należą do nich wszystkie próby dostępu do obiektu pod lokacją **null** oraz rzucenie wyjątku explicite przez słowo kluczowe **throw**. Znów, nie dzieje się tutaj nic specjalnego, a dowód poprawności wynika w prosty sposób z semantyki języka.

Najbardziej skomplikowaną spośród reguł dla wyrażeń prostych jest reguła HT-INVOKE. Opisuje ona wywołanie zachowanie ewaluacji podczas wywołania metody m na obiekcie klasy C, lub pochodnej i stanowi główną motywację dla wprowadzenia pojęcia listy niezmienników. W klasycznym podejściu, zastosowanym również w logice Iris, reguła dla wywołania funkcji wymaga udowodnienia oczekiwanych własności tylko dla wyrażenia będącego ciałem tej funkcji z odpowiednimi wartościami podstawionymi w miejsce argumentów. Jednak, z uwagi na to, że Jafun (podobnie jak Java) oferuje polimorfizm, nie można bez ewaluacji rozstrzygnąć, jakiego dokładnie typu jest dany obiekt, a zatem nie można też rozstrzygnąć, jak wygląda ciało metody o danej nazwie.

Założenie, mówiące że lista niezmienników jest udowodniona, daje nam pewność, że trójka Hoare'a $\{P'\} \cdot \{w.Q'\}_A$ rzeczywiście zachodzi dla każdej metody w obiektach dziedziczących z C. Zatem, na mocy założenia indukcyjnego dla przesłanki, zachodzi dla nich także trójka $\{P\}x.m(\overline{v})\{w.Q\}_A$.

6.7.2. Reguły dla wyrażeń złożonych

Podstawowymi twierdzeniami pozwalającymi nam udowodnić poprawność reguł dla wyrażeń złożonych są twierdzenia 1 (O łączeniu ewaluacji) oraz 2 (O ewaluacji w rozszerzonym kontekście). Przykładem zastosowania ich obu jest dowód poprawności reguły HT-LET, opisującej wykonanie wyrażenia **let** $C\ x = E_1$ **in** E_2 przy założeniu, że wyrażenie E_1 wykonuje się w sposób normalny, niewyjątkowy.

Z założenia indukcyjnego dla pierwszej przesłanki wynika, że wyrażenie E_1 ewaluuje się poprawnie do pewnej lokacji l zawierającej obiekt typu C,w taki sposób, że sterta po ewaluacji spełnia wyrażenie $Q[l/x]$.

Korzystając z redukcji letin, dostajemy $[\![\textbf{let}\ C\ x = E_1\ \textbf{in}\ E_2]\!]_\emptyset \to [\textbf{let}\ C\ x = [\![E_1]\!]_\emptyset\ \textbf{in}\ E_2]$. Możemy teraz zastosować twierdzenie 2 od otrzymanej z założenia indukcyjnego ewaluacji wyrażenia E_1, co w efekcie da nam kompletną ewaluację wyrażenia wewnętrznego **let**: $[\textbf{let}\ C\ x = [\![E_1]\!]_\emptyset\ \textbf{in}\ E_2] \rightsquigarrow [\textbf{let}\ C\ x = [\![l]\!]_\emptyset\ \textbf{in}\ E_2]$. Korzystając teraz z redukcji letgo, otrzymujemy $[\textbf{let}\ C\ x = [\![l]\!]_\emptyset\ \textbf{in}\ E_2] \to [\![E_2\{l/x\}]\!]$. Możemy wreszcie zastosować założenie indukcyjne dla drugiej przesłanki w środowisku rozszerzonym o $(x \mapsto l)$, co daje nam ewaluację $[\![E_2\{l/x\}]\!] \rightsquigarrow [\![l']\!]_A$, przy czym sterta po ewaluacji spełnia $R[l'/w]$.

Ostatecznie, możemy skorzystać z twierdzenia 1, aby połączyć wyżej pokazane ewaluacje i redukcje w jedną ewaluację $[\![\textbf{let}\ C\ x = E_1\ \textbf{in}\ E_2]\!]_\emptyset \rightsquigarrow [\![l']\!]_A$, dla której sterta końcowa spełnia $R[l'/w]$.

Założenie o trwałości termu S jest istotne w powyższym rozumowaniu w momencie, w którym stosujemy założenie indukcyjne dla drugiej przesłanki. Zauważmy, że sterta, dla

której chcemy je zastosować to sterta powstała po ewaluacji wyrażenia E_1, a zatem bez założenia o trwałości nie musiałaby ona spełniać S, co uniemożliwiłoby zastosowanie założenia indukcyjnego. Ponieważ jednak S jest trwałe i spełniane przez stertę początkową, jest więc również spełniane przez stertę po wykonaniu E_1.

Dowód poprawności reguły dla **let** jest najbardziej skomplikowanym z dowodów dla reguł złożonych. Zastosowane są w nim wszystkie techniki używane w dowodzie pozostałych przypadków. Bardzo podobne rozumowanie można więc przeprowadzić dla wszystkich pozostałych reguł.

Rozdział 7

Formalizacja w systemie Coq

W ramach niniejszej pracy prezentowana logika oraz dowód poprawności zostały sformalizowane w systemie Coq. Dowód poprawności logiki został zbudowany na podstawie Coq-owej formalizacji semantyki języka Jafun, która została przedstawiona w [2].

7.1. Reprezentacja poszczególnych obiektów

7.1.1. Obiekty Jafun

Reprezentacja wyrażeń Jafun, sterty oraz lokacji i obiektów na stercie pozostaje bez zmian w stosunku do formalizacji semantyki języka Jafun. W szczególności sterta reprezentowana jest jako skończony słownik przyporządkowujący liczbom naturalnym (interpretowanym jako lokacje) obiekty Jafun:

```
Module NatMap := FMapAVL.Make(Nat_as_OT).
Definition Heap : Type :=  NatMap.t Obj. (* Loc => Obj *)
```

Taka reprezentacja wymagała jednak udowodnienia dużej liczby pomocniczych lematów opisujących podstawowe własności sterty, takie jak łączność, przemienność i element neutralny operatora $\oplus$ sumy rozłącznej stert, czy przechodniość relacji bycia podstertą.

Największym jednak wyzwaniem było zdefiniowanie równości na stertach i jej własności. Ponieważ sterty zdefiniowane są jako `FMapAVL`, kolejność, w której dodawane są elementy, ma znaczenie. Oznacza to, że sterty `Heap.add(l1, o1, Heap.add(l2, o2, Heap.empty))` oraz `Heap.add(l2, o2, Heap.add(l1, o1, Heap.empty))` są interpretowane przez Coq jako dwie różne sterty, mimo że koncepcyjnie reprezentują dokładnie tę samą stertę $\{l_1 \mapsto o_1, l_2 \mapsto o_2\}$.

Wymusiło to zdefiniowanie własnej relacji równoważności stert, utożsamiającej sterty zawierające te same lokacje i przypisujące im równe obiekty.

```
Definition HeapEq (h1 h2 : Heap) := forall n, Heap.find n h1 = Heap.find n h2.
```

Oczywiście, mając taką definicję, również należało udowodnić wszystkie jej pożądane własności, takie jak symetria i zwrotność. Oprócz tego istotnym jest twierdzenie podobne do tych zaprezentowanych w rozdziale 5 – dowolne dwie równe sterty są równoważne względem semantyki dla każdego termu. Dowód tego twierdzenia przebiega przez prostą indukcję po budowie termu, a przypadki dla równości i trójek Hoare'a wynikają wprost z równości obiektów pod równymi lokacjami.

7.1.2. Termy logiki

Termy logiki reprezentowane są przez dwa typy indukcyjne, `JFIOuterTerm` i `JFITerm`, odpowiadające termom odpowiednio **P** i P. Dla uproszczenia dowodu, w definicji termu definiujemy wartości w inny sposób niż są one definiowane w składni Jafun. Wartości w Jafun zdefiniowane są jako

```
Inductive Loc : Set := | null | JFLoc (n:nat).
Inductive JFRef := | JFVar (x:JFXId) | JFThis.
Inductive JFVal := | JFVLoc (l:Loc) | JFSyn (x:JFRef).
```

Z zachowaniem takiej definicji wartości także dla termów logiki wiązałyby się dwa problemy. Po pierwsze, w logice nie dopuszczamy konkretnych lokacji poza **null**, zatem w dowodzie trzeba by zachować tę własność jako niezmiennik. Po drugie, **null**, **this** oraz nazwane zmienne traktowane są w logice jako równorzędne obiekty, więc dodatkowa hierarcha nie jest potrzebna. Z tych powodów, w składni logiki przyjęliśmy prostszą definicję wartości:

```
Inductive JFIVal : Type := | JFINull | JFIThis | JFIVar (var : string).
```

Ponieważ jednak w wyrażeniach Jafun (występujących w logice w trójkach Hoare'a) wartości nadal reprezentowane są przez `JFVal`, nieraz zdarza się, że ta sama wartość jest reprezentowana zarówno przez typy `JFVal`, jak i `JFIVal`. Do zapewnienia, że w takich sytuacjach wartości są ze sobą zgodne, służy pomocnicza definicja `JFIValToJFVal`, zdefiniowana w naturalny sposób jako

```
Definition JFIValToJFVal val :=
 match val with
 | JFINull => JFVLoc null
 | JFIThis => JFSyn JFThis
 | JFIVar var => JFSyn (JFVar var)
 end.
```

Jest ona wielokrotnie używana w formalizacji reguł wnioskowania dla trójek Hoare'a, aby powiązać ze sobą wartości występujące w warunkach początkowych lub końcowych z odpowiadającymi im wartościami w wyrażeniu Jafun.

Semantyka termu, przy ustalonym programie, środowisku i stercie, jest sformalizowana przez predykat `JFIHeapSatisfiesInEnv`, który jest naturalnym przepisaniem definicji z rysunku 2.2.

7.1.3. Reguły wnioskowania

Reguły wnioskowania dla obu poziomów termów zdefiniowane są jako typy indukcyjne

```
Inductive JFIProves : JFIDeclsType -> JFITypeEnv -> JFITerm -> JFITerm -> Prop
Inductive JFIProvesOuter : JFIDeclsType -> JFITypeEnv -> JFIOuterTerm ->
                           JFIOuterTerm -> Prop
```

Typ `JFIDeclsType` to rekord zawierający program $\overline{\mathbb{C}}$, o którym wnioskujemy oraz listę niezmienników metod `invariants`. Jest on wspólny dla wszystkich reguł, a zatem żadna z nich go nie modyfikuje. Typ `JFITypeEnv` natomiast to środowisko typowe Γ, czyli słownik przypisujący nazwom zmiennych ich typy. Może on być modyfikowany przez poszczególne reguły, na przykład przy wprowadzaniu nowej zmiennej przez kwantyfikator $\exists$.

Istnienie obiektu o typie `JFIProves decls gamma p q` jest interpretowane jako dowodliwość osądu `decls, gamma` | $p \vdash q$, a sam obiekt tego typu interpretowany jest jako dowód tego osądu. Zaletą takiego podejścia jest łatwa weryfikacja dowodu osądów – jeśli można skonstruować obiekt danego typu to dowód musi być poprawny – oraz możliwość dowodzenia przez indukcję po budowie dowodu.

Przykładem prostej reguły wnioskowania jest reguła przechodniości Trans, mówiąca o że jeśli z `p` można udowodnić `q`, a z `q` można udowodnić `r`, to również z `p` można udowodnić `r`:

```
...
| JFITransRule :
    forall q decls gamma p r,
      (JFIProves decls gamma p q) -> (JFIProves decls gamma q r) ->
      (*-------------------------------------------------------*)
      JFIProves decls gamma p r
...
```

Dodatkowym założeniem, obecnym w większości reguł, jest założenie, że wszystkie zmienne wolne w termie muszą być obecne w środowisku typów. Warunek ten jest opisywane przez predykat `FreeVarsInTermAreInGamma` i pojawia się we wszystkich regułach, w których nie może on być wywnioskowany z osądów pojawiających się w przesłankach.

Na przykład reguła Asm, w której to założenie jest potrzebne, jest zapisana jako

```
...
| JFIAsmRule :
    forall decls gamma p,
      (FreeVarsInTermAreInGamma p gamma) ->
      (*---------------------------------*)
      JFIProves decls gamma p p
...
```

Wspomniane wcześniej powiązanie między wartościami logiki (typu `JFIVal`) a wartościami w wyrażeniach Jafun (typu `JFVal`) jest wymagane wszędzie we wszystkich tych regułach, gdzie pewna wartość występuje jednocześnie w termie ($\cdot = \cdot$ lub $\cdot \hookrightarrow \cdot = \cdot$) i w wyrażeniu Jafun. Przykładem prostej reguły, która wymaga takiego powiązania, jest reguła Ht-ret, mówiąca że zawsze możemy udowodnić trójkę Hoare'a $\{\texttt{True}\}w\{v.v = w\}_\emptyset$. Pierwsze w jest wyrażeniem języka Jafun, składającym się tylko z wartości. Drugie w, występujące w warunku końcowym, jest operandem operatora $=$, a zatem jest częścią termu logiki.

W poniższej regule, symbol `w_expr` reprezentuje wartość Jafun, a symbol `w` – wartość w termie (dla jasności, w poniższej trójce Hoare'a `JFITrue` to warunek początkowy, `None` oznacza brak wyjątku, a `v` to nowa zmienna związana z wartością wyrażenia). Powiązanie wartości Jafun z wartością logiki jest realizowane przez założenie `FreeVarsInValAreInGamma w gamma`.

```
...
| JFIHTRetRule :
    forall decls gamma s v w w_expr,
      w_expr = JFIValToJFVal w ->
      (*-------------------------------------------------------------*)
      JFIProves decls gamma s
        (JFIHoare JFITrue (JFVal1 w_expr) None v (JFIEq (JFIVar v) w))
...
```

7.2. Dowód poprawności

Najważniejszym pojęciem, użytym w sformułowaniu twierdzenia o poprawności jest semantyczne wynikanie. Powiemy, że przy ustalonym programie i środowisku typów Γ term P wynika semantycznie z termu S, jeśli dla dowolnej sterty h i środowiska env, takich że env zgadza się z Γ na h, jeśli h spełnia S w env, to h spełnia również P w env:

```
Definition JFISemanticallyImplies gamma s p (CC : JFProgram) :=
  forall env this h,
    JFIGammaMatchEnv h gamma env ->
    JFIHeapSatisfiesInEnv h s env this CC ->
    JFIHeapSatisfiesInEnv h p env this CC.
```

Kiedy mamy taką definicję, twierdzenie o poprawności może być sformułowane przez powiedzenie, że jeśli $\Gamma|P \vdash T$, to T wynika semantycznie z P w środowisku Γ :

```
Theorem JFISoundness : forall gamma decls p t,
  JFIProves decls gamma p t ->
  JFISemanticallyImplies gamma p t (JFIDeclsProg decls).
```

Twierdzenie to jest następnie dowodzone przez indukcję po budowie drzewa dowodu dla `JFIProves decls gamma p t` zgodnie z opisem w rozdziale 6.

Rozdział 8

Podsumowanie

Przedstawiona w niniejszej pracy logika separacyjna dla języka Jafun pozwala wnioskować o programach napisanych w języku Jafun. Dzięki wprowadzonym operatorom separacyjnym dowody można przeprowadzać osobno dla rozłącznych fragmentów sterty, a następnie scalać w dowód dla sterty utworzonej przez sumę tych fragmentów.

Dzięki sformalizowaniu logiki w systemie Coq poprawność dowodów twierdzeń o programach w języku Jafun można zweryfikować komputerowo, a udowodnione twierdzenie o poprawności logiki zapewnia, że udowodnione twierdzenia mają odzwierciedlenie w semantyce języka.

W przyszłości możliwe jest uzupełnienie logiki o kwantyfikator uniwersalny $\forall$. Dla zachowania poprawności logiki wymagałoby to jednak zmianę definicji sterty, a co za tym idzie również zmiany semantyki język Jafun. Logika bez tego kwantyfikatora jest jednak również wartościowa i umożliwia dowodzenie szerokiej gamy twierdzeń o zachowaniu programów napisanych w Jafun.

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

[3] L. Birkedal and A. Bizjak. Lecture Notes on Iris: Higher-Order Concurrent Separation Logic, 2018

[4] R. Jung, R. Krebbers, J. Jourdan, A. Bizjak, L. Birkedal and D. Dreyer. Iris from the ground up. A modular foundation for higher-order concurrent separation logic, 2018

[5] R. Jung, J. Jourdan, R. Krebbers, and D. Dreyer. 2018. RustBelt: Securing the Foundations of the Rust Programming Language. *Proc. ACM Program. Lang.* 2, POPL, Article 66, January 2018

[6] A. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *TOPLAS*, 23(5), 657–683, 2001