Rachunek Lambda i Języki Programowania

Małgorzata Biernacka

Instytut Informatyki UWr

Wykład 3

Plan

1. Ewaluacja w językach funkcyjnych
   - Strategie ewaluacji i równoważność programów
   - Ewaluacja w kontekstach
   - Maszyny abstrakcyjne

Plan

Plan

Semantyka operacyjna języków

- Znaczenie konstrukcji języka określa pewna specyfikacja procesu 'wykonania' programu
- Semantyka operacyjna może być określona w sposób formalny, np. przez system reguł dedukcji określających pewną "relację przejścia":
  - Semantyka dużych kroków definiuje relację między programami a ich wartościami
  - Semantyka małych kroków definiuje relację przejścia między "stanami" wykonania programu
- Semantyka operacyjna powinna być czytelna dla użytkownika języka, powinna umożliwiać wnioskowanie o programach oraz implementację języka
- Różne opisy semantyczne tego samego języka powinny być równoważne

Semantyka wykonywalna

- Semantyka zadana przez program (interpreter, kompilator)
- Język, którego semantykę definiujemy, nazywamy językiem definiowanym, źródłowym
- Język, w którym określona jest semantyka wykonywalna (interpreter lub kompilator) nazywamy językiem definiującym
- Język definiujący musi być wystarczająco ekspresywny, by symulować zachowanie języka definiowanego
- Semantyka, jaką określimy dla języka definiowanego zależy od semantyki języka definiującego (oraz języka docelowego w przypadku kompilatora)

Semantyka wykonywalna

- Interpreterem języka S napisanym w języku L nazywamy L-program *int*, który dla dowolnego S-programu s i danych dla programu s, będzie symulował wykonanie programu s w języku L, aż do obliczenia wartości (jeśli istnieje):

$$[\![s]\!]^S(x) = [\![int]\!]^L(s, x)$$

- Kompilatorem języka S do języka T, napisanym w języku L, nazywamy L-program *comp* taki, że

$$[\![s]\!]^S = [\![[\![comp]\!]^L(s)]\!]^T$$

Język Λ_n

- Program – dowolny zamknięty λ – *term*:

$$t := x \mid t\ t \mid \lambda x.t$$

- Wartość programu – λ-abstrakcja:

$$v := \lambda x.t$$

- Strategia normalna ('wołanie-przez-nazwę', 'call-by-name')
- Semantyka operacyjna małych kroków realizująca tę strategię:

$$\frac{}{(\lambda x.t)\ s \to_n t\ [s/x]} \qquad \frac{r \to_n s}{r\ u \to_n s\ u}$$

- Obserwacyjna równoważność:

$$t \sim_{ctx} s \Leftrightarrow \forall C.C[t] = C[s]$$

(= oznacza, że oba zatrzymują się i zwracają te same wartości, lub oba nie zatrzymują się)

Równoważność programów

- Obserwacyjna równoważność: kiedy dwa termy (fragmenty programu) są "wymienialne"? Kiedy można jeden zastąpić drugim?
- Zastosowanie: optymalizacja kodu, uproszczenie kodu
- Równoważność obserwacyjna jest wrażliwa na rozszerzenia języka

Ewaluacja w Λ_n a teoria $\lambda\beta$

1. Dla każdego termu r,

   $r \to^* v$ dla pewnej wartości $v \Leftrightarrow r \to_n^* v'$ dla pewnej wartości v'.

2. Dla każdych r, s,

   $$\lambda\beta \vdash r = s \Rightarrow r \sim_{ctx} s.$$

   Teoria $\lambda\beta$ nie jest zupełna: dowolne dwa termy, których ewaluacja się nie zatrzymuje są obserwacyjnie równoważne, ale nie muszą być równe w teorii (np. Ω i $\Omega\,(\lambda x.x)$).

Równoważność semantyki dużych i małych kroków

Twierdzenie

Dla każdego programu $t \in \Lambda_n$,

$$t \Downarrow_n v \Leftrightarrow t \to_n^* v.$$

Język Λ_v

- Program – dowolny zamknięty λ – *term*:

$$t := x \mid t\ t \mid \lambda x.t$$

- Wartość programu – λ-abstrakcja:

$$v := \lambda x.t$$

- Strategia aplikatywna ('wołanie-przez-wartość', 'call-by-value')
- Semantyka małych kroków realizująca tę strategię:

$$\frac{}{(\lambda x.t)\ v \rightarrow_v t\ [v/x]}$$

$$\frac{r \rightarrow_v s}{r\ u \rightarrow_v s\ u}$$

$$\frac{r \rightarrow_v s}{v\ r \rightarrow_v v\ s}$$

Ewaluacja w Λ_v a teoria $\lambda\beta^v$

- Jaki jest związek ewaluacji w Λ_v z teorią $\lambda\beta$?
- Zdefiniujmy teorię $\lambda\beta^v$ przez zastrzeżenie relacji β-konwersji do reguły

$$\frac{}{(\lambda x.t)\, v = t\ [v/x]}$$

- Teorię $\lambda\beta^v$ definiują reguły:

$$\frac{}{s = s} \qquad \frac{s = u}{u = s} \qquad \frac{r = s \quad s = u}{r = u}$$

$$\frac{}{(\lambda x.t)\, v = t\ [v/x]} \qquad \frac{s = r}{s\ t = r\ t}$$

$$\frac{s = t}{r\ s = r\ t} \qquad \frac{s = t}{\lambda x.s = \lambda x.t}$$

- Podobnie jak w przypadku $\lambda\beta$, definiujemy relację redukcji $\longrightarrow$ przez skierowanie równań
- Powstała relacja redukcji ma własności CR i standardyzacji

Ewaluacja w Λ_v a teoria $\lambda\beta^v$, c.d.

1. Dla każdego termu r,

   $r \to^* v$ dla pewnej wartości $v \Leftrightarrow r \to_v^* v'$ dla pewnej wartości v'.

2. Dla każdych r, s,

   $$\lambda\beta^v \vdash r = s \Rightarrow r \sim_{ctx} s.$$

   Teoria $\lambda\beta^v$ nie jest zupełna: dowolne dwa termy, których ewaluacja się nie zatrzymuje są obserwacyjnie równoważne, ale nie muszą być równe w teorii (np. Ω i $\Omega\,(\lambda x.x)$).

Translacja między Λ_v i Λ_n

- Translacja $\bar{\cdot} : \Lambda_v \to \Lambda_n$

$$
\begin{aligned}
\overline{x} &= \lambda k.k\, x \\
\overline{\lambda x.t} &= \lambda k.k\, (\lambda x.\overline{t}) \\
\overline{t_1\ t_2} &= \lambda k.\overline{t_1}\, (\lambda v_1.\overline{t_2}\, (\lambda v_2.v_1\ v_2\ k))
\end{aligned}
$$

- Translacja przeprowadza termy do stylu kontynuacyjnego (continuation-passing style, CPS)

Translacja między Λ_n i Λ_v, c.d.

Własności termów w CPS:

- Niezależność ewaluacji od strategii redukcyjnej języka definiującego:

$$\overline{t}(\lambda x.x) \rightarrow_v^* v \Leftrightarrow \overline{t}(\lambda x.x) \rightarrow_n^* v$$

- Symulacja ewaluacji:

$$t \rightarrow_v^* v \Leftrightarrow \overline{t}(\lambda x.x) \rightarrow_n^* \phi(v),$$

  gdzie $\phi(\lambda x.r) = \lambda x.\overline{r}$.
- Translacja:

$$\lambda\beta^v \vdash r = s \Rightarrow \lambda\beta^v \vdash \overline{r} = \overline{s} \Rightarrow \lambda\beta^n \vdash \overline{r} = \overline{s}$$

- Własność zachowania równoważności obserwacyjnej:

$$\overline{t}(\lambda x.x) \sim_n \overline{s}(\lambda x.x) \Rightarrow t \sim_v s$$

- Odwrotna implikacja nie zachodzi

Redeksy administratywne

- Translacja $\bar{\cdot}$:
  - narzuca porządek ewaluacji podtermów
  - nazywa częściowe wyniki obliczeń
- Translacja $\bar{\cdot}$ wprowadza dodatkowe redeksy (niewystępujące w termie źródłowym), tzw. redeksy administratywne
- Redeksy administratywne mogą być zredukowane, prowadząc do bardziej zwięzłej postaci termu
- Programując translację, można wykonać wszystkie redeksy administratywne w jednym przejściu (używając ewaluacji na poziomie metajęzyka do ich zredukowania)

Przykład

$$\overline{\lambda x.x} = \lambda k.k\,(\lambda x k'.k'\,x)$$

$$\overline{y} = \lambda k.k\,y$$

$$\overline{(\lambda x.x)\,y} = \lambda k.(\lambda x k.k\,x)\,y\,k$$

Translacja między Λ_n i Λ_v

- Translacja $\bar{\cdot} : \Lambda_v \to \Lambda_n$

$$
\begin{aligned}
\overline{x} &= x \\
\overline{\lambda x.t} &= \lambda k.k\,(\lambda x.\overline{t}) \\
\overline{t_1\ t_2} &= \lambda k.\overline{t_1}\,(\lambda v.v\ \overline{t_2}\ k)
\end{aligned}
$$

Translacja między Λ_n i Λ_v, c.d.

Własności termów w CPS:

- Niezależność ewaluacji od strategii redukcyjnej języka definiującego:

$$\overline{t}(\lambda x.x) \rightarrow_v^* v \Leftrightarrow \overline{t}(\lambda x.x) \rightarrow_n^* v$$

- Symulacja ewaluacji:

$$t \rightarrow_n^* v \Leftrightarrow \overline{t}(\lambda x.x) \rightarrow_v^* \phi(v),$$

  gdzie $\phi(\lambda x.r) = \lambda x.\overline{r}$.
- Translacja:

$$\lambda\beta^n \vdash r = s \Leftrightarrow \lambda\beta^v \vdash \overline{r} = \overline{s} \Leftrightarrow \lambda\beta^n \vdash \overline{r} = \overline{s}$$

- Własność zachowania równoważności obserwacyjnej:

$$\overline{t}(\lambda x.x) \sim_v \overline{s}(\lambda x.x) \Rightarrow t \sim_n s$$

- Odwrotna implikacja nie jest prawdziwa

Interpreter dla Λ_n

- Rozważmy nieformalną definicję iterpretera dla rachunku lambda (języka S):

$$E(Var\ x) = x$$

$$E(App\ r\ s) = (E(r))(E(s))$$

$$E(Abs\ f) = \lambda v.E(f\ v)$$

- Język definiujący L – lambda rachunek z pewną strategią ewaluacji
- Każda konstrukcja języka definiowanego jest modelowana przez tę odpowiadającą cechę języka definiującego
- Pytanie: jaka jest strategia, którą nadamy językowi definiowanemu? Co się stanie, jeśli ją zmienimy?

Interpreter dla Λ_n

- Rozważmy nieformalną definicję iterpretera dla rachunku lambda (języka S):

$$E(Var\, x) = x$$

$$E(App\, r\, s) = (E(r))(E(s))$$

$$E(Abs\, f) = \lambda v.E(f\ v)$$

- Język definiujący L – lambda rachunek z pewną strategią ewaluacji
- Każda konstrukcja języka definiowanego jest modelowana przez tę odpowiadającą cechę języka definiującego
- Pytanie: jaka jest strategia, którą nadamy językowi definiowanemu? Co się stanie, jeśli ją zmienimy?

- Jeśli L jest CBN, to $App\,(Abs\,(\lambda xy.y))\,\Omega$ ewaluuje do $\lambda y.E(y)$
- Jeśli L jest CBV, to $App\,(Abs\,(\lambda xy.y))\,\Omega$ nie zatrzymuje się

Interpreter dla Λ_n, c.d.

- Przy użyciu kontynuacji możemy narzucić konkretną strategię:

$$E\,(Var\,x)\,c = c\,x$$

$$E\,(App\,r\,s)\,c = E\,(r)\,(\lambda f.E\,(s)\,(\lambda v.f\ v\ c))$$

$$E\,(Abs\,f)\,c = c\,(\lambda v c_1.E\,(f\ v)\,c_1)$$

- Wniosek: Semantyka języka źródłowego zależy od semantyki języka definiującego – w tym przypadku nadaliśmy językowi źródłowemu semantykę aplikatywną przez jawne wyspecyfikowanie kolejności obliczeń za pomocą kontynuacji

Interpreter ze środowiskiem

- Jeśli nie chcemy używać HOAS ani podstawienia, standardowym rozwiązaniem jest interpreter ze środowiskiem (environment)
- użycie środowiska umożliwia "opóźnione podstawienie" – dopiero wtedy, gdy ewaluacja dotrze do zmiennej, wybieramy ze środowiska to, co należało podstawić (domknięcie)
- domknięcie (closure) jest parą (term, środowisko)
- środowisko jest listą par (zmienna, domknięcie)

$$e := \cdot \mid (x, c) :: e$$

$$c := (t, e)$$

- Interpreter ze środowiskiem:

$$E(Var\ x)\ e = \texttt{lookup}\ e\ x$$

$$E(App\ r\ s)\ e = (E(r)\ e)(E(s)\ e)$$

$$E(Abs\ (x,\ r))\ e = \lambda v.E(r)\ (\texttt{extend}\ x\ v\ e)$$

Plan

Semantyka redukcyjna

- Semantyka małych kroków
- Jawny kontekst obliczeń
- Otrzymywana bezpośrednio z definicji pojęcia redukcji i redukcji jednokrokowej

Strategia w Λ_n za pomocą kontekstów

- Konteksty wyrażają reguły kompatybilności:

$$\frac{r \to_n s}{r\ u \to_n s\ u}$$

- Składnia kontekstów:

$$C_n := [\,] \mid C_n[[\,]\ t]$$

- Semantykę kontekstów definiuje funkcja wstawiania:

$$[\,][r] = r$$

$$C_n[[\,]\ s][r] = C_n[(r\ s)]$$

- Wówczas strategię ewaluacji można przedstawić następująco:

$$C_n[(\lambda x.r)\ s] \to_n C_n[r\ [s/x]]$$

Semantyka redukcyjna, c.d.

- gramatyka termów
- gramatyka kontekstów
- funkcja wstawiania
- pojęcie redukcji (jeden krok obliczenia)
- wartościami są termy, które nie są redeksami i które nie dadzą się zapisać w postaci $C[t]$

Strategia w Λ_v za pomocą kontekstów

- Mamy 2 reguły kompatybilności:

$$\frac{r \to_v s}{r\ u \to_v s\ u} \qquad \frac{r \to_v s}{v\ r \to_v v\ s}$$

- Składnia kontekstów:

$$C_v := [\,] \ \mid \ C_v[[\,]\ t] \ \mid \ C_v[v\ [\,]]$$

- Definicja funkcji wstawiania:

$$[\,][r] = r$$

$$C_v[[\,]\ s][r] = C_v[r\ s]$$

$$C_v[v\ [\,]][r] = C_v[v\ r]$$

- Wówczas strategię ewaluacji można przedstawić następująco:

$$C_v[(\lambda x.r)\ v] \to_n C_v[r\ [v/x]]$$

Własność jednoznacznego rozkładu

- Jeśli strategia jest deterministyczna, to mamy własność jednoznacznego rozkładu:

Semantyka redukcyjna ma własność jednoznacznego rozkładu, jeśli dla każdego termu t niebędącego wartością, istnieje dokładnie jeden kontekst C i dokładnie jeden (potencjalny) redeks r taki, że $C[r] = t$.

- Semantyka przy CBV i CBN ma własność jednoznacznego rozkładu.

Potencjalne redeksy

- Redeksem potencjalnym nazywamy term, który może być albo redeksem właściwym, albo termem "niepoprawnym" w sensie semantyki języka ("stuck term").
- Rozważmy rozszerzenie Λ_n o stałe liczbowe $\lceil n \rceil$ i stałą S reprezentującą operację następnika.
- Wartości: $v := \lambda x.r \mid \lceil n \rceil$
- Potencjalne redeksy: $r := v\ t \mid S v$
- Dodatkowe reguły redukcji:

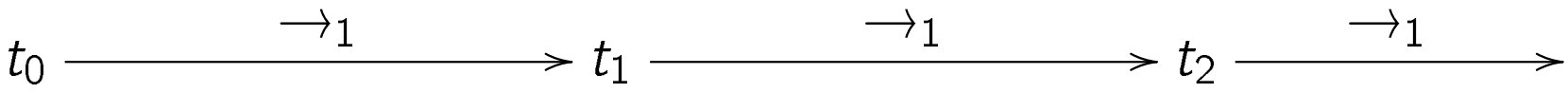
$$S \lceil n \rceil \to_n \lceil n+1 \rceil$$

- Termy postaci $\lceil n \rceil\ r$ lub $S\ \lambda x.r$ są poprawne składniowo, ale niepoprawne semantycznie – nie można ich ewaluować, ale nie są poprawnymi wartościami

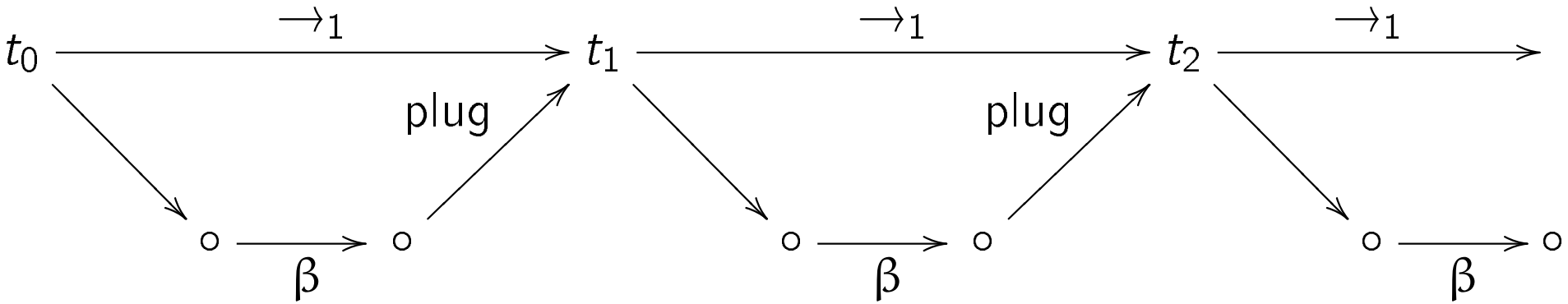
Interpreter

- Semantyka redukcyjna prowadzi do następującej implementacji procesu ewaluacji:
  1. Napisz funkcję dekomponującą term na redeks i kontekst
  2. Wykonaj β-redukcję
  3. Wstaw zredukowany term do kontekstu
  4. Powtarzaj aż do osiągnięcia wartości (jeśli istnieje)

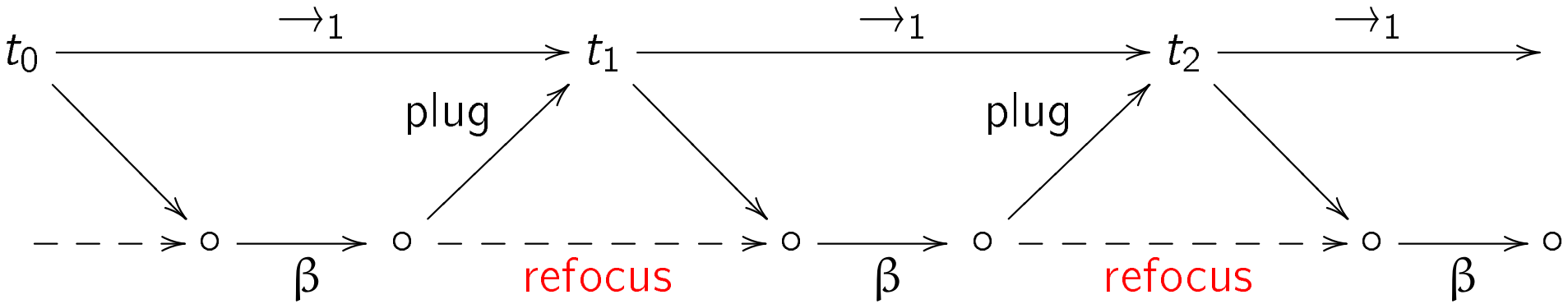
“Refocusing” – optymalizacja ewaluacji



“Refocusing” – optymalizacja ewaluacji



"Refocusing" – optymalizacja ewaluacji



- "refocusing" – optymalizacja ciągu redukcji (bez rekonstrukcji termów pośrednich)
- refocus – funkcja przejścia między stanami
- stan = (term, kontekst)
- przechodnie domknięcie funkcji refocus (iteracja) – (prawie) maszyna abstrakcyjna

Plan

Maszyny abstrakcyjne

- Abstrakcyjny model implementacji języka lub symulacji wykonania programu według pewnej strategii
- Maszyna abstrakcyjna jest określona jako funkcja przejścia między stanami (konfiguracjami):
  - stan określa chwilowy stan obliczeń (łącznie z ewentualnym stanem zasobów potrzebnych do wykonania tych obliczeń)
  - możliwe przejścia między stanami określa funkcja przejścia (determinizm)
  - wyróżniamy stan początkowy i stan końcowy (lub stany końcowe)
- Maszyna abstrakcyjna jest abstrakcyjnym modelem, który może pomijać pewne szczegóły implementacyjne, zależnie od zastosowania

Maszyna Krivine'a z podstawieniem

- Wartości: $v ::= \lambda x.t$
- Konteksty ewaluacyjne: $E ::= [\,] \mid E[[\,]\,t]$
- Przejścia maszyny:

$$
\begin{aligned}
t &\Rightarrow \langle t, [\,]\rangle \\
\langle \lambda x.t,\, E[[\,]\,t']\rangle &\Rightarrow \langle t\,[t'/x],\, E\rangle \\
\langle t_1\,t_2,\, E\rangle &\Rightarrow \langle t_1,\, E[[\,]\,t_2]\rangle \\
\langle v, [\,]\rangle &\Rightarrow v
\end{aligned}
$$

Maszyna Krivine'a ze środowiskiem

- Wartości: $v ::= (\lambda x.t,\ e)$
- Środowisko: $e ::= e_{empty} \mid e[x \mapsto (t,\ e)]$
- Konteksty ewaluacyjne: $E ::= [\,] \mid \texttt{ARG}(t, e, E)$
- Przejścia maszyny:

$$
\begin{aligned}
t &\Rightarrow \langle t,\ e_{empty},\ [\,]\rangle \\
\langle x,\ e,\ E\rangle &\Rightarrow \langle t,\ e',\ E\rangle \quad \text{gdzie } e(x) = (t,\ e') \\
\langle \lambda x.t,\ e,\ \texttt{ARG}(t', e', E)\rangle &\Rightarrow \langle t,\ e[x \mapsto (t',\ e')],\ E\rangle \\
\langle t_1\ t_2,\ e,\ E\rangle &\Rightarrow \langle t_1,\ e,\ \texttt{ARG}(t_2, e, E)\rangle \\
\langle \lambda x.t,\ e,\ [\,]\rangle &\Rightarrow (\lambda x.t,\ e)
\end{aligned}
$$

Strategia maszyny Krivine'a

- Maszyna Krivine'a jest poprawna względem semantyki małych kroków

Twierdzenie

Dla każdego programu $t \in \Lambda_n$,

$$t \to_n^* v \Leftrightarrow \langle t, e_{empty}, [\,] \rangle \Rightarrow^* (v', e),$$

gdzie (v', e) *reprezentuje wartość* v.

- Strategia maszyny symuluje strategię $\to_n$
- Każda tranzycja w maszynie odpowiada albo β-redukcji, albo regule kompatybilności, która wybiera następny podterm do ewaluacji

Środowisko a podstawienie

Realizacja β-redukcji:

- przez podstawienie

$$(\lambda x.t)\, s \quad \rightarrow \quad t\,[s/x]$$

- przez środowisko

$$\begin{aligned} \langle \lambda x.t,\ e,\ \mathtt{ARG}(t', e', E) \rangle \quad &\Rightarrow \quad \langle t,\ e[x \mapsto (t',\ e')],\ E \rangle \\ \langle x,\ e,\ E \rangle \quad &\Rightarrow \quad \langle t',\ e',\ E \rangle \quad \text{gdzie } e(x) = (t',\ e') \end{aligned}$$

Środowisko a podstawienie, c.d.

- Środowisko:
  - nie ma potrzeby wykonywać α-przemianowanie
  - umożliwia rozumowanie przez indukcję strukturalną na termach
  - podczas ewaluacji, termy nie ulegają zmianie, więc mogą być kompilowane
- Podstawienie:
  - meta-operacja
  - brak związku z praktyczną techniką implementacji – środowiskiem, co utrudnia wnioskowanie o własnościach implementacji

Maszyna CEK ze środowiskiem

- Wartości: $v ::= [x,\ t,\ e]$
- Środowisko: $e ::= e_{empty} \mid e[x \mapsto v]$
- Konteksty: $E ::= [\,] \mid \texttt{ARG}(t, e, E) \mid \texttt{FUN}(v, E)$
- Przejścia maszyny:

$$
\begin{aligned}
t &\Rightarrow \langle t,\ e_{empty},\ E\rangle_{eval} \\
\langle x,\ e,\ E\rangle_{eval} &\Rightarrow \langle E,\ e\,(x)\rangle_{apply} \\
\langle \lambda x.t,\ e,\ E\rangle_{eval} &\Rightarrow \langle E,\ [x,\ t,\ e]\rangle_{apply} \\
\langle t_1\ t_2,\ e,\ E\rangle_{eval} &\Rightarrow \langle t_1,\ e,\ \texttt{ARG}(t_2, e, E)\rangle_{eval} \\
\langle \texttt{ARG}(t, e, E),\ v\rangle_{apply} &\Rightarrow \langle t,\ e,\ \texttt{FUN}(v, E)\rangle_{eval} \\
\langle \texttt{FUN}([x,\ t,\ e], E),\ v\rangle_{apply} &\Rightarrow \langle t,\ e[x \mapsto v],\ E\rangle_{eval} \\
\langle [\,],\ v\rangle_{apply} &\Rightarrow v
\end{aligned}
$$

Strategia maszyny CEK

- Maszyna CEK jest poprawna względem semantyki małych kroków

Twierdzenie

Dla każdego programu $t \in \Lambda_v$,

$$t \to_v^* v \Leftrightarrow \langle t, e_{empty}, [\,] \rangle \Rightarrow^* (v', e),$$

gdzie (v', e) *reprezentuje wartość* v.

- Strategia maszyny symuluje strategię $\to_v$
- Każda tranzycja w maszynie odpowiada albo β-redukcji, albo regule kompatybilności, która wybiera następny podterm do ewaluacji

Maszyna SECD

- Pierwsza maszyna abstrakcyjna dla lambda rachunku (Landin '64)
- Konfiguracja maszyny $\langle S, E, C, D \rangle$ zawiera:
  - stos wartości S
  - środowisko E
  - kod C
  - stos D
- Ekstensjonalnie równoważna maszynie CEK (strategia CBV)
- Intensjonalnie realizuje inny model implementacyjny niż CEK

Maszyna SECD, c.d.

$$
\begin{array}{rcl}
t & \Rightarrow & \langle \cdot, e_{empty}, t, \cdot \rangle \\
\langle [x, t, e] :: S, E, \cdot, \langle S', E', C', D' \rangle \rangle & \Rightarrow & \langle [x, t, e] :: S', E', C', D' \rangle \\
\langle S, E, x :: C, D \rangle & \Rightarrow & \langle E(x) :: S, E, C, D \rangle \\
\langle S, E, \lambda x.t :: C, D \rangle & \Rightarrow & \langle (\lambda x.t, E) :: S, E, C, D \rangle \\
\langle (\lambda x.t, E') :: v :: S, E, ap :: C, D \rangle & \Rightarrow & \langle \cdot, E[x \mapsto v], t, \langle S, E, C, D \rangle \rangle \\
\langle S, E, (r\ s) :: C, D \rangle & \Rightarrow & \langle S, E, s :: r :: ap :: C, D \rangle \\
\langle v :: S, e_{empty}, \cdot, \cdot \rangle & \Rightarrow & v
\end{array}
$$

Maszyna SECD vs. CEK

- SECD: W momencie wywołania funkcji bieżący stan maszyny jest zachowywany na stosie
- CEK: Kontekst jest rozszerzany w momencie ewaluacji funkcji i argumentu; wywołanie funkcji zawsze zmniejsza kontekst
- Te własności powodują, że maszyny konsumują różną ilość miejsca na kontekst:
  - Ewaluacja Ω jest nieskończona na obu maszynach
  - Ilość miejsca na stos konsumowana przez CEK w tej ewaluacji jest ograniczona
  - Stos w SECD rośnie nieograniczenie z każdym wywołaniem funkcji
- Środowisko może rosnąć nieograniczenie w obu przypadkach (tę wadę można wyeliminować)