R. D. Tennent, The Denotational Semantics of Programming Languages [1976]

Programowanie 2009 - seminarium grupy zaawansowanej

Jakub Tarnawski

Instytut Informatyki Uniwersytetu Wrocławskiego

1 lipca 2009

See the little phrases go,
Watch their funny antics.
The men who make them wiggle so
Are teachers of semantics.

Motywacja dla rozwoju semantyk

- precyzyjna i kompletna referencja
- niezależność od implementacji
- standaryzacja terminologii pozwala dostrzec podobieństwa
- optymalizacja (lub nawet generacja) kompilatorów
- dowodzenie własności programów

Funkcje semantyczne

- zdefiniowano bardzo prosty język LOOP, zbiory zmiennych, wyrażeń, instrukcji, programów, itd.
- wartości funkcji semantycznych zależą od zawartości *store* (pamięci)
- np. $[\![\xi]\!]\sigma = \sigma(\xi)$ (ξ - zmienna)
- np. $[\![\xi := E]\!]\sigma = \sigma[\xi := [\![E]\!]]$
- np. $[\![\Gamma_1; \Gamma_2]\!] = [\![\Gamma_2]\!] \circ [\![\Gamma_1]\!]$
- itd.

LOOP jest prosty i imperatywny

- nie ma goto i etykiet
- nie ma λ-wyrażeń
- nie ma funkcji ani procedur
- nie ma problemów z dziedzinami funkcji semantycznych

Autor na tym przykładzie pokazał podstawy denotacji, abstrahując na razie od pojęć środowiska/kontekstu oraz kontynuacji.

Języki aplikatywne

Autor definiuje je jako mające właśność *referential transparency*, czyli wyrażenia bez skutków ubocznych. Inaczej: języki deklaratywne, funkcyjne. Przykłady wyrażeń:

1. let $x = 5$ in $x + 3$
2. $x + 3$ where $x = 5$
3. let $f(x) = x + 3$ in $f(5)$
4. $\sum_{i=1}^{6}(i + 3)$
5. $f(5)$ where $f(n) = \begin{cases} 1 & \text{jeśli } n = 0 \\ n \cdot f(n-1) & \text{w p. p.} \end{cases}$

W każdym przykładzie identyfikatory są wiązane z wartościami; dlatego wartościowanie wyrażeń odbywa się przy zadanym *środowisku* (wartościowaniu zmiennych wolnych wyrażenia).

Języki aplikatywne

Autor definiuje je jako mające właśność *referential transparency*, czyli wyrażenia bez skutków ubocznych. Inaczej: języki deklaratywne, funkcyjne. Przykłady wyrażeń:

1. let $x = 5$ in $x + 3$
2. $x + 3$ where $x = 5$
3. let $f(x) = x + 3$ in $f(5)$
4. $\sum_{i=1}^{6}(i + 3)$
5. $f(5)$ where $f(n) = \begin{cases} 1 & \text{jeśli } n = 0 \\ n \cdot f(n-1) & \text{w p. p.} \end{cases}$

W każdym przykładzie identyfikatory są wiązane z wartościami; dlatego wartościowanie wyrażeń odbywa się przy zadanym *środowisku* (wartościowaniu zmiennych wolnych wyrażenia).

Języki aplikatywne

Autor definiuje je jako mające właśność *referential transparency*, czyli wyrażenia bez skutków ubocznych. Inaczej: języki deklaratywne, funkcyjne. Przykłady wyrażeń:

1. let $x = 5$ in $x + 3$
2. $x + 3$ where $x = 5$
3. let $f(x) = x + 3$ in $f(5)$
4. $\sum_{i=1}^{6}(i + 3)$
5. $f(5)$ where $f(n) = \begin{cases} 1 & \text{jeśli } n = 0 \\ n \cdot f(n-1) & \text{w p. p.} \end{cases}$

W każdym przykładzie identyfikatory są wiązane z wartościami; dlatego wartościowanie wyrażeń odbywa się przy zadanym *środowisku* (wartościowaniu zmiennych wolnych wyrażenia).

Języki aplikatywne

Autor definiuje je jako mające właśność *referential transparency*, czyli wyrażenia bez skutków ubocznych. Inaczej: języki deklaratywne, funkcyjne. Przykłady wyrażeń:

1. let $x = 5$ in $x + 3$
2. $x + 3$ where $x = 5$
3. let $f(x) = x + 3$ in $f(5)$
4. $\sum_{i=1}^{6}(i + 3)$
5. $f(5)$ where $f(n) = \begin{cases} 1 & \text{jeśli } n = 0 \\ n \cdot f(n-1) & \text{w p. p.} \end{cases}$

W każdym przykładzie identyfikatory są wiązane z wartościami; dlatego wartościowanie wyrażeń odbywa się przy zadanym *środowisku* (wartościowaniu zmiennych wolnych wyrażenia).

Języki aplikatywne

Autor definiuje je jako mające właśność *referential transparency*, czyli wyrażenia bez skutków ubocznych. Inaczej: języki deklaratywne, funkcyjne. Przykłady wyrażeń:

1. let $x = 5$ in $x + 3$
2. $x + 3$ where $x = 5$
3. let $f(x) = x + 3$ in $f(5)$
4. $\sum_{i=1}^{6}(i + 3)$
5. $f(5)$ where $f(n) = \begin{cases} 1 & \text{jeśli } n = 0 \\ n \cdot f(n-1) & \text{w p. p.} \end{cases}$

W każdym przykładzie identyfikatory są wiązane z wartościami; dlatego wartościowanie wyrażeń odbywa się przy zadanym *środowisku* (wartościowaniu zmiennych wolnych wyrażenia).

Struktura aplikatywna

Okazuje się, że (tak jak w λ-rachunku) wystarczy niewiele, aby zadać całą strukturę takich wyrażeń:

- atomy
- aplikacje
- abstrakcje

Niech $[\![.]\!] : Exp \to \ldots$ - interpretacja wyrażeń.

Example (1,2)

$[\![\text{let } x = 5 \text{ in } x + 3]\!] = [\![x + 3 \text{ where } x = 5]\!] = [\![(\lambda x.x + 3)(5)]\!]$

Example (3)

$[\![\text{let } f(x) = x + 3 \text{ in } f(5)]\!] = [\![\text{let } f = \lambda x.x + 3 \text{ in } f(5)]\!] =$
$[\![(\lambda f.f(5))(\lambda x.x + 3)]\!]$

A co z sumą i silnią?

Example (4)

$\sum_{i=1}^{6}(i+3)$

Można to zapisać jako $Sigma(1, 6, \lambda i.i + 3)$, gdzie *Sigma* zdefiniowano rekurencyjnie, ale nie wiadomo, czy ta definicja jest poprawna.

A co z sumą i silnią?

Example (5)

$$f(5) \text{ where } f(n) = \begin{cases} 1 & \text{jeśli } n = 0 \\ n \cdot f(n-1) & \text{w p. p.} \end{cases}$$

Rozważamy równanie ze względu na zmienną funkcyjną f:

$$f = \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \cdot f(n-1)$$

Niech Y - funkcjonał znajdujący punkt stały, tzn. $YF = F(YF)$.
Np. dla silnii mamy

$$F(g) = \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \cdot g(n-1)$$

Więc silnię można definiować tak:

$$f(5) \text{ where } f = YF \text{ where } F(g)(n) = \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \cdot g(n-1),$$

co umiemy interpretować.

Kombinator punktu stałego

Gdyby ograniczać się do λ-rachunku, to mamy

Twierdzenie

Istnieje kombinator punktu stałego Y taki, że $YF =_\beta F(YF)$.

Dowód.

Np. $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. □

Twierdzenie o punkcie stałym

Twierdzenie (Kleene)

Niech L będzie porządkiem zupełnym, a $f : L \to L$ funkcją ciągłą. Wtedy najmniejszym punktem stałym f jest kres górny ciągu $\{f^n(\bot) \mid n \in \mathbb{N}\}$:

$$\bot \leq f(\bot) \leq f(f(\bot)) \leq f^3(\bot) \leq \ldots$$

Do uogólnienia tego wyniku potrzebne są metody teorii dziedzin.

Gramatyka języka AEXP

B: Bas (wartości bazowe, literały) z interpretacją Υ
I: Ide (identyfikatory zmiennych)
E: Exp (wyrażenia)

Definicja

$$E ::= B \mid I \mid \lambda I.E \mid E_1 E_2$$

Denotacja języka AEXP

Niech:

D - wartości denotowalne

$\rho : U = Ide \to D$ - środowiska

$\phi : F = D \to D$ - funkcje

Definicja ($[\![.]\!] : Exp \to U \to D$)

$[\![B]\!]\rho = \Upsilon(B)$

$[\![I]\!]\rho = \rho(I)$

$[\![\lambda I.E]\!]\rho = \phi$, *gdzie* $\phi(\delta) = [\![E]\!](\rho[I := \delta])$

$[\![E_1 E_2]\!]\rho = \phi(\delta)$, *gdzie* $\phi = [\![E_1]\!]\rho$ *oraz* $\delta = [\![E_2]\!]\rho$

FAIL

Ta definicja jest niepoprawna.

Czemu jest źle (1)

Example

$[\![\lambda I.E]\!]\rho = \phi \in F$
Ale przeciwdziedziną funkcji $[\![.]\!]\rho$ jest D.

Example

$[\![E_1 E_2]\!]\rho = \phi(\delta)$, gdzie $\phi = [\![E_1]\!]\rho$ oraz $\delta = [\![E_2]\!]\rho$
czyli $\delta \in D, \phi \in D$, ale $\phi : D \to D$, tzn. $\phi \in F$

Można mieć nadzieję, że $F \subseteq D$.
Możemy stosować

- injekcje (wstrzyknięcia) (ϕ in D)
- projekcje (rzutowania) ($\epsilon \mid F$)

co sformalizujemy później. Ale to nie wystarczy.

Czemu jest źle (2)

- funkcje wyższego rzędu są problematyczne ze względu na rekursję
- w AEXP mieliśmy przestrzeń funkcji $F = D \to D$
- skoro $F \subseteq D$, to dla każdej funkcji ϕ znaczenie $\phi(\phi)$ ma być zdefiniowane, co prowadzi do sprzeczności:

Dowód.

Niech p będzie predykatem spełnionym wtw, gdy jego argument po zaaplikowaniu do siebie samego zwraca fałsz.
Tzn. $p(q) = \neg q(q)$.
Wtedy $p(p) = \neg p(p)$. □

Czemu jest źle (3)

Naszej definicji nie da się też zapisać w Haskellu.

```
data Var = X | Y | Z deriving Eq
data LambdaE = Abs Var LambdaE | App LambdaE LambdaE | Var Var

get _ [] = error "dsfargeg"
get x ((y,v):ys) | x == y = v
                 | otherwise = get x ys

denot (Var var) store = get var store
denot (Abs var lambda) store = \x -> denot lambda ((var,x):store)
```

Już po tej deklaracji dostajemy:

```
Occurs check: cannot construct the infinite type: t = t -> t1
  Expected type: [(Var, t)]
  Inferred type: [(Var, t -> t1)]
In the second argument of ‘(:)’, namely ‘store’
```

Teoria dziedzin

Teorię dziedzin stworzył Dana Scott w latach '60, korzystając z teorii krat i topologii.

- dziedzina - "typ danych"
- funkcje (w tym wyższego rzędu) wystarczająco ogólne, aby pozwolić na zapis naturalnych modeli obliczeń (np. rekursja, samoaplikacja)
- ale wystarczająco ograniczone, by system pozostał niesprzeczny
- podstawowa cecha: ciągi przybliżeń (pewnych częściowych obiektów) w dziedzinie powinny być zbieżne
- granica też powinna należeć do dziedziny i być zachowywana przez operacje (*ciągłe* przekształcenia)

Proste dziedziny

Proste dziedziny to skończone lub przeliczalne zbiory uzupełnione o elementy $\bot$ i $\top$. Ozn. $A^\circ = A \cup \{\top, \bot\}$.

Example

$N = \{\ldots, -2, -1, 0, 1, \ldots\}^\circ$
$T = \{true, false\}^\circ$

W takich prostych dziedzinach zadajemy porządek *przybliżeń* $\leq$:

Definicja

$(\forall x)\ \bot \leq x \leq \top$
inne pary są nieporównywalne

Tworzenie dziedzin

Definicja

Niech D, D_1 i D_2 - dziedziny. Definiujemy następujące dziedziny:

1. $D_1 \times D_2$ *(iloczyn)*
2. $D_1 + D_2$ *(suma)*
3. $D_1 \to D_2$ *(funkcje)*
4. $D^n = D \times D \times \cdots \times D$ *(listy długości n)*
5. $D^* = D^0 + D^1 + D^2 + \ldots$ *(skończone listy)*

Do każdej dziedziny należą dodatkowo $\bot$ i $\top$.

Porządek przybliżeń wyprowadzamy ze składowych.

Example ($D_1 \to D_2$)

$$f \leq g \;\Leftrightarrow\; (\forall x : D_1)\; f(x) \leq g(x)$$

Inspekcje, injekcje i projekcje

Dla sumy $X = ... + Y + ...$ definiujemy:

Definicja $(x : X)$

$$xeY = \begin{cases} T & \textit{jeśli } x \textit{ odpowiada pewnemu } y : Y \\ F & \textit{w p. p.} \end{cases}$$

Definicja $(x : X)$

$$x \mid Y = \begin{cases} y & \textit{jeśli } x \textit{ odpowiada } y : Y \\ \bot & \textit{gdy } x \textit{ nie odpowiada żadnemu } y : Y \end{cases}$$

Definicja $(y : Y)$

y *in* $X = x$, *gdzie* $x : X$ *odpowiada* y

Ciągłość, strictness

Na prostych dziedzinach ciągłość $\Leftrightarrow$ monotoniczność:

Definicja

f jest monotoniczna $\Leftrightarrow (\forall x, y)\ x \leq y \Rightarrow f(x) \leq f(y)$

Każda **funkcja częściowa** na **zbiorze** może być rozszerzona do **ciągłej funkcji całkowitej** na odpowiadającej **dziedzinie**:

1. $f(\bot) = \bot$,
2. $f(\top) = \top$,
3. $f(x) = \bot$, jeśli funkcja częściowa jest niezdefiniowana dla x.

Taką funkcję nazywamy *doubly strict*. Funkcje nie muszą takie być:

Example

if-then-else(true, 2, $\bot$) $= 2 \neq \bot$

Rekursja

Dla ustalonej dziedziny D szukamy kombinatora punktu stałego Y_D, który rozwiązuje równania postaci $f = F(f)$ dla $F : D \to D$. Mamy z monotoniczności

$$\bot \leq F(\bot) \leq F(F(\bot)) \leq \ldots \leq F^i(\bot) \leq \ldots$$

i ten ciąg, z ciągłości, zbiega do granicy f takiej, że

$$(\forall i \geq 0)\ F^i(\bot) \leq f \quad \text{oraz} \quad F(f) = f$$

Twierdzenie

Dla każdej dziedziny D istnieje ciągła funkcja $Y_D : (D \to D) \to D$ taka, że dla każdego ciągłego $F : D \to D$:

1. $Y_D(F) = \lim_{i\to\infty} F^i(\bot)$ *jest rozwiązaniem równania* $f = F(f)$
2. $Y_D(F)$ *przybliża każde inne rozwiązanie*

Samoaplikacja

Można też rekurencyjnie definiować dziedziny i ich elementy.

Example

Można skonstruować dziedzinę D taką, że:

1. $D = B + (D \to D)$
2. B - wartości bazowe

czyli każde $\delta : D$ odpowiada albo wartości bazowej, albo funkcji $\phi : D \to D$, którą można zaaplikować do każdego elementu D, w tym do ϕ in D.