3 czerwca 2008 *Mateusz Styrczula*[1]

Programowanie

Notatki do wykładów[2]

Streszczenie

Haskell, ewaluatory wyrażeń, monady, klasy typów, semantyki denotacyjne, semantyki operacyjne, transformatory monad, leniwość, funkcje monolityczne, Strachey, funkcje inkrementacyjne, predykaty, baba-w-babie, łamigłówki, drzewa czerwono-czarne, teorie modeli, dowody indukcyjne, metodologie programowania, Wadler, cukierki o smaku kaszanki, sinusy, Tarski, cyklisty, generatory, polimorfizm, algebry, modele, formuły, predykaty ...

Trzeba kształtować młode umysły. Gdybyśmy z Haskellem zaczynali (podobnie jak chcą z nauką Religii) w przedszkolu, to polska informatyka wyglądałaby zupełnie inaczej.

ToMasz Wierzbicki

1 Wykład 04.03.2008

To co się dzieje w komputerze, to nie „magia”. John von Neumann – program i dane w jednej pamięci operacyjnej. Opowieść o metodologiach programowania, podział języków na te „niskiego poziomu” i „wysokiego poziomu”. A C++ – jaki jest, każdy widzi. Leniwość, gorliwość, strukturalizacja kodu, `goto` jest złe, instrukcja przypisania jest do bani, człowiek potrafi zapamiętać 2.5 bita informacji... Komputery dobrze radzą sobie z dużą ilością takich samych (bądź bardzo podobnych) „obiektów”, a ludzie dobrze radzą sobie z małą ilością „obiektów” (ale obiekty te mogą być zupełnie różne od siebie – krzesło, samolot, obiad...). Wstępik do Haskell'a. Chcemy pisać specyfikacje a nie „kodować”.

2 Wykład 06.03.2008

1. http://kno.ii.uni.wroc.pl/ii/ – (kształcenie na odległość...) – Moodle – Programowanie – Strachey
2. Rekursja strukturalna...
3. „List comprehensions”:
   - `[e |` **True**`] = [e]`
   - `[e | b, q] =` **if** `b` **then** `[e | q]` **else** `[]`

[1]rozdziały oznaczone „(TWi)” są autorstwa ToMasza Wierzbickiego i pochodzą z udostępnionych przez niego notatek

[2]a w zasadzie luźne myśli, hasła i zlepki dziwnych symboli...

- [e | p ← l, q] = **let**
```
  f p = [e | q]
  f _ = []
in
  concatMap f l
```

- [e | q] = [e | q, **True**]

3 Wykład 11.03.2008

3.1 Dwa fajne sposoby na generowanie liczb pierwszych

```
primes :: [Integer]
primes = map head $ iterate (λ (x:xs) → [y | y ← xs, y `mod` x ≠ 0]) [2..]

primes :: [Integer]
primes = 2 : [n | n ← [3..], all (λ p → n `mod` p ≠ 0)
                              (takeWhie (λ p → p ×p ≤ n) primes)]
```

3.2 Fajny sposób na ciąg fibonacciego

też był omówiony i pokazany ...

3.3 Monoid (półgrupa z jedynką)

$$\langle X, \oplus, e\rangle$$

- łączna: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- ma obustronny element neutralny:
  $e \oplus x = x$
  $x \oplus e = x$

3.4 Klasy typów

```
class Monoid a where
     (××) :: a → a → a
        e :: a

instance Monoid Integer where
     (××) = (+)
        e = 0

class Show a where
     show :: a → Show
```

3.5 O monadach

```
class Monad m where                          -- m :: * -> *
     (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a

    (u >>= (\x -> v)) >>= (\y -> w)   =   u >>= (\x -> (v >>= \y -> w))
                     return x >>= f   =   f x
                      u >>= return    =   u
```

4 Wykład 13.03.2008

4.1 Monoid raz jeszcze

Monoid (a, plus, zero)

```
class Monoid a where
    plus :: a -> a -> a
    zero :: a

(x `plus` y) `plus` z   =   x `plus` (y `plus` z)
        zero `plus` x   =   x
        x `plus` zero   =   x
```

4.2 O ciągu fibonacciego

Specyfikacja ciągu fibonacciego:

$$f_0 = 0$$
$$f_1 = 0$$
$$f_{n+2} = f_{n+1} + f_n$$

Programik w Haskellu:

```
fib :: Num a => a -> a
fib 0 = 1
fib 1 = 1
fib (n+2) = fib (n+1) + fib n
```

- memoizacja,
- programowanie dynamiczne,
- kompresja ścieżek,

Szybki fib co się do while'a skompiluje:

```
ffib _ f0 0 = f0
ffib f1 _ 1 = f1
ffib f1 f0 n = ffib (f1+f0) f1 (n-1)
```

Można go uogólnić:

```
fib :: (Monoid a, Num n) ⇒ a → a → n → a
fib _ f0 0 = f0
fib f1 _ 1 = f1
fib f1 f0 n = fib (f1 `plus` f0) f1 (n−1)
```

`Integer` może być instancją monoidu:

```
instance Monoid Integer where
    plus = (+)
    zero = 0
```

ale możemy bardziej uogólnić:

```
instance Num n => Monoid n where    -- <---- tu nawet może być rekursja
    plus = (+)
    zero = 0
```

dygresja o klasie `Show`:

```
instance Show a => Show (a, a) where
    show (x, y) = "(" ++ show x ++ ", " ++ show y ++ ")"
```

`String` jest aliasem `type String = [Char]` i trzeba ustawić flagę `-XTypeSynonymInstances`

```
instance Monoid String where
    plus = (++)
    zero = ""
```

albo tak:

```
instance Monoid [a] where
    plus = (++)
    zero = []
```

to sobie możemy napisać tak:

```
h = fib "a" "b" 3 ...
```

napiszmy szybki „power”:

```
power :: (Monoid t, Integral n) ⇒ t → n → t
power x 0 = zero
power x n
    | n `mod` 2 == 0 = y `plus` y
    | otherwise      = y `plus` y `plus` x where
      y = power x (n `div` 2)

type Matrix2x2 a = ((a, a), (a, a))

instance Num a ⇒ Monoid (Matrix2x2 a) where
    ((a11, a12), (a21, a22)) `plus` ((b11, b12), (b21, b22))
                                              = ((a11×b11 + a12×b21) ...
    zero = ((1,0), (0,1))
```

zatem nowy fibonacci tak wygląda:

```
fib :: (Integral n, Num a) ⇒ n → a
fib n = snd . snd power a0 $ n−1 where
    a0 = ((0,1), (1,1))
```

4.3 Monady

```
--        * -> *
--          ::
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a

  {-
    (m >>= (\a -> n)) >>= (\b -> p )  =  m >>= (\a -> (n >>= \b -> p))
    return a >>= (\x -> m)  =  (\x -> m) a

    return a >>= f   =   f a

    m >>= return  =  m
  -}
```

return – „owiń w sreberko”

Mały przykład wyjaśniający działanie monad:

```
data Tree  = Node Tree Int Tree | Leaf deriving Show

renumber :: Tree -> Tree
renumber = snd . aux 1
    aux n Leaf = (n, Leaf)
    aux n (Node t1 _ t2) = (n'', Node t1' n' t2') where
            (n', t1') = aux n t1
            (n'', t2') = aux (n+1) t2
```

Słowo kluczowe `newtype` – działa tak jak `data`, ale nie tworzy konstruktora, kompilator może lepiej optymalizować, konstruktor widoczny tylko na poziomie kodu źródłowego...

```
newtype IntState a = IntState (Int -> (Int, a))
-- ściema lekka, bo IntState musi najpierw należeć do klasy Functor,
-- żeby móc być monadą...

instance Monad IntState where
    return a = IntState (\n -> (n, a))
    (IntState st1) >>= f  =  IntState (\m -> let (n, s1) = st1 m
                                                 IntState st2 = f s1
                                             in
                                                 st2 n)
```

Wyposażmy użytkownika w jakieś narzędzia operujące na tym liczniku:

```
fetch :: IntState Int
fetch = IntState (\n -> (n, n))

assign :: Int -> IntState ()
assign n = IntState (\_ -> (n, ()))

run :: Int -> IntState a -> a
run n (IntState st) = snd $ st n

renumber :: Tree -> Tree
renumber = run 0 . aux where
    aux :: Tree -> IntState Tree
    aux Leaf = return Leaf
    aux (Node t1 _ t2) = aux t1 >>= \t1' -> (fetch >>=
                                     \n -> (assign (n+1)) >>=
                                     \() -> (aux t2 >>=
                                     \t2' -> return (Node t1' n t2')))
```

Powyższy zapis jest bardzo nieczytelny. Mamy taki oto cukier:

$$\begin{array}{l} do \\ \quad e_1 \\ \quad p \leftarrow e_2 \\ \quad let \quad p = e_3 \\ \quad \vdots \end{array}$$

- $\begin{array}{l} do \\ \quad x \leftarrow e_1 \\ \quad e_2 \end{array}$ jest tym samym co $e_1 \gg= (\lambda\, x \to e_2)$
- $\begin{array}{l} do \\ \quad e_1 \\ \quad e_2 \end{array}$ jest tym samym co $e_1 \gg= (\lambda\, \_ \to e_2)$

Przepiszmy `renumber` używając powyższego cukru syntaktycznego:

```
renumber = run 0 . aux where
    aux Leaf = return Leaf
    aux (Node t1 _ t2) = do
        t1' <- aux t1
        n <- fetch
        assign (n+1)
        t2' <- aux t2
        return (Node t1' n t2')
```

5 Wykład 18.03.2008

Głównie source w Haskellu...

6 Wykład 20.03.2008

Slajdy w PDF'ach i source w Haskellu...

7 Wykład 27.03.2008

7.1 Semantyka Denotacyjna

1. Rachunek predykatów formuł pierwszego rzędu

$$\begin{aligned}
&\varphi ::= R(t_1, ..., t_n) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \varphi_1 \Leftrightarrow \varphi_2 \mid \forall x\varphi \mid \exists x\varphi \\
&t ::= x \mid f(t_1, ..., t_n) \\
&\Sigma = \{\mathcal{R}_i, f_j\} - \textit{sygnatura}
\end{aligned}$$

2. A. Tarski, 1934

$$[\![\cdot]\!] : \mathcal{T} \to A \qquad [\![0]\!] = 0 \in A$$

3. Interpretacja zmiennych
   Niech $\mathcal{X}$ – zbiór zmiennych.
   Interpretacja zmiennych: $\eta : \mathcal{X} \to A$ – dowolne odwzorowanie

$$[\![\cdot]\!] : \mathcal{T} \to (\mathcal{X} \to A) \to A$$

$$\begin{cases} [\![\mathcal{X}]\!]_\eta = \eta(x) \\ [\![f(t_1, ..., t_n)]\!]_\eta = f^{\mathfrak{M}}([\![t_1]\!]_\eta, ..., [\![t_n]\!]_\eta) \end{cases}$$

Skoro mamy termy, to możemy przejść do punktu „a co znaczą całe formuły?”:

$$\begin{aligned}
&\mathfrak{M} = \langle \mathbb{N}, 1, +, < \rangle \\
&\Sigma = \{J, P, M\}, \text{ gdzie} \\
&J^{\mathfrak{M}} = 1 \\
&P^{\mathfrak{M}} = (+) \\
&M^{\mathfrak{M}} = (<)
\end{aligned}$$

Wprowadzimy sobie specjalną relację ($\vDash$):

$$\mathfrak{M}, \eta \vDash \varphi \qquad (\ \eta - \text{„wartościowanie”})$$

(odpowiednikiem homomorfizmów[3] w świecie relacji są kongruencje)

$$\begin{array}{rcl} \mathfrak{M}, \eta \models R(t_1, ..., t_2) & \text{wtw.} & R^{\mathfrak{M}}([\![t_1]\!]_\eta, ..., [\![t_2]\!]_\eta) \\ \mathfrak{M}, \eta \models \neg\varphi & \text{wtw.} & \text{nieprawda, że} \quad \mathfrak{M} \models \varphi \\ \mathfrak{M}, \eta \models \varphi_1 \vee \varphi_2 & \text{wtw.} & \mathfrak{M}, \eta \models \varphi_1 \quad \text{lub} \quad \mathfrak{M}, \eta \models \varphi_2 \\ \mathfrak{M}, \eta \models \forall x \varphi & \text{wtw.} & \text{dla każdego } a \in A \quad \mathfrak{M}, \eta[x \to a] \models \phi \\ \text{gdzie:} & & \end{array}$$

$$\eta[x \to a](y) = \begin{cases} a, & \text{jeżeli } y = x \\ \eta(y) & \text{w p.p.} \end{cases}$$

Chcielibyśmy mieć jakieś efektywne procedury sprawdzania co jest prawdziwe, a co nie. Tym się zajmuje dualna do teorii modeli „teoria dowodów" (w informatyce „semantyka operacyjna"). Jest takie fajne twierdzenie, mówiące „Udowodnić można tylko to co jest prawdą".

7.2 Rodzaje semantyki formalnej (TWi)

W definicjach wielu starszych języków (np. Algolu 60, Algolu 68 i in.) opisywano abstrakcyjną maszynę danego języka, a następnie zadawano znaczenie programu w sposób nieformalny, opisując słownie, jak się zmienia stan takiej maszyny w trakcie wykonywania programu. Język naturalny jest jednak niejednoznaczny. Lepiej użyć formalizmu matematycznego. *Semantyka formalna* to metoda ścisłego, matematycznego określenia, jak zmienia się stan maszyny abstrakcyjnej na skutek wykonania programu. Wyróżnia się następujące rodzaje semantyki formalnej:

- *Semantyka denotacyjna.*
  Programom przyporządkowuje się pewne obiekty matematyczne, zwane ich *interpretacjami* lub *denotacjami*, stąd nazwa tego rodzaju semantyki. Przeważnie są to funkcje. W przypadku języka z wykładu będą to odwzorowania $f : \mathbb{S} \to \mathbb{S}$ określające, jak zmienia się stan maszyny na skutek wykonania programu (tj. takie, że jeśli $\sigma \in \mathbb{S}$ jest stanem maszyny przed wykonaniem programu, którego denotacją jest f, to $f(\sigma)$ jest stanem maszyny po wykonaniu tego programu). W tym celu definiuje się *ad hoc* pewną algebrę nad ustaloną sygnaturą, zwaną *modelem* opisywanego języka programowania, a język traktuje się jako zbiór termów nad tą sygnaturą. Znaczenie programu jest jego interpretacją (jako termu) w modelu. Jest to klasyczna metoda nadawania znaczenia wyrażeniom języka, inspirowana technikami znanymi z *teorii modeli*, tj. działu logiki matematycznej, w którym nadaje się w ten sposób znaczenie formułom logicznym. Metodę denotacyjną opisu języków programowania wynalazł w latach 60-tych zeszłego wieku Christopher Strachey. Na potrzeby semantyki denotacyjnej Dana Scott stworzył tzw. *teorię dziedzin*, tj. matematyczną teorię budowy modeli języków programowania.

- *Semantyka algebraiczna.*
  Model języka programowania zadaje się za pomocą zbioru aksjomatów równościowych. Jest to technika zbliżona do sposobu, w jaki definiuje się i bada różne struktury (np. grupy) w algebrze ogólnej, stąd nazwa tego rodzaju semantyki.

- *Semantyka operacyjna.*
  To podejście najsilniej akcentuje rolę maszyny abstrakcyjnej języka. Definiuje się pojęcie podobne do tego, które mówiąc o automatach skończonych nazwaliśmy *funkcją przejścia*

[3] „... a państwo nie lubicie homomorfizmów, niestety, bo jesteście homofobami ..."

automatu. Znaczenie programu opisuje się podając, jakie *operacje* wykona maszyna realizująca program, stąd nazwa tego rodzaju semantyki. We współczesnej formie formalizm ten został rozwinięty przez Gordona Plotkina w latach 70-tych zeszłego wieku w postaci tzw. *strukturalnej semantyki operacyjnej.*

- *Semantyka aksjomatyczna.*
  Definiuje się specjalny formalizm logiczny do wyrażania własności programów i system wnioskowania do dowodzenia tych własności. Język jest wówczas opisany przez zbiór odpowiednich *aksjomatów* i reguł wnioskowania, stąd nazwa tego rodzaju semantyki. Stworzyli ją w latach 60-tych zeszłego wieku R. W. Floyd i C. A. R. Hoare. Semantyka aksjomatyczna bywa też nazywana *logiką Floyda-Hoare'a.*

8 Wykład 01.04.2008

8.1 Przypomnienie

Ostatnio sobie rozmawialiśmy o logice. To jeszcze parę słów przypomnienia. Zaczęliśmy opisania sobie rachunku predykatów formuł pierwszego rzędu.

Mamy jakby 2 kategorie gramatyczne. Termy budujemy tak:

$$t ::= f(t_1, ..., t_n),\ n \geq 0\ |x \text{ – to jest algebra termów.}$$

Na term patrzymy jak na drzewo.
Jak chcemy zobaczyć „co jest w języku” to patrzymy na jego składnię abstrakcyjną.
Mamy tu zasadę indukcyjną.

$$\varphi ::= R(t_1, ...t_n) \mid \neg\varphi \mid \varphi_1 \oplus \varphi_2 \mid \forall x\varphi \mid \exists x\varphi$$
$$\oplus ::= \vee \mid \wedge \mid \Rightarrow \mid \Leftrightarrow$$

$$\varphi,\ \eta,\ \mathfrak{M} \qquad \eta \models^{\mathfrak{M}} \varphi$$

$$\models^{\mathfrak{M}} \varphi \text{ wtw. gdy dla każdego możliwego } \eta \text{ zachodzi } \eta \models^{\mathfrak{M}} \varphi$$

W rachunku zdań „światy” to są możliwe wartościowania zmiennych.
Chcemy poznawać takie prawdy, które obowiązują w każdym możliwym rozważanym świecie (tautologia).

Tautologia – formuła prawdziwa.

$$\models \varphi \text{ wtw. gdy } \models^{\mathfrak{M}} \varphi \text{ dla każdego } \mathfrak{M}.$$

To była teoria modeli. A teraz możemy zostać w samym języku. Centralnym punktem naszego zainteresowania jest język. Chcemy mieć automatyczne narzędzia weryfikacji co jest prawdziwe, a co nie.

8.2 Teoria Dowodu

Przyjmujemy sobie pewne zdania za prawdziwe, oraz mamy pewne reguły wyprowadzania nowych zdań. Każdy sobie może taki system wyprowadzania wymyślić[4]:

1. Aksjomaty (które są tak naprawdę kombinatorami)

$$\Gamma \vdash \alpha \to \beta \to \alpha$$

$$\Gamma \vdash (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$$

$$\Gamma \vdash \alpha \to \alpha$$

2. reguły wnioskowania:

$$\frac{\Gamma \vdash \alpha \quad \Gamma \vdash \alpha \to \beta}{\Gamma \vdash \beta}$$

itp...

mówimy:

$$\begin{array}{l} \vDash \varphi - \varphi \text{ jest prawdziwa} \\ \vdash \varphi - \varphi \text{ jest dowodliwa} \end{array}$$

Ustalamy sobie konkretny model i staramy się sprawdzić co jest prawdziwe w tym modelu.

$$\vDash^{\mathfrak{M}} \varphi \qquad \vdash^{T} \varphi$$

Jeśli coś udowodnię to to jest prawdziwe. Ale jest cała masa rzeczy prawdziwych, których nie mogę udowodnić. Zawsze prawd jest więcej niż to co jesteśmy w stanie wywnioskować w samym systemie.

8.3 Mały wstęp do składni

1. Świat naszych instrukcji wygląda tak:

$$\begin{aligned} c &::= \textbf{skip} \mid X := e \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{if } b \textbf{ then } c \mid \textbf{while } b \textbf{ do } c \mid c_1; c_2 \\ e &::= n \mid X \mid e_1 \oplus e_2 \\ \oplus &::= + \mid - \mid \times \mid \textbf{div} \mid \textbf{mod} \\ b &::= e_1 \otimes e_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \end{aligned}$$

[4]np. formalizacja „Hilbertowska” w whitebooku...

2. Budujemy model $\mathfrak{M}$:
   - **Typy:**
     - $\pi : \mathcal{X} \to \mathbb{Z}$
     - $\pi = \mathbb{Z}^{\mathcal{X}}$
     - $[\![c]\!] : \pi \to \pi$
     - $[\![c]\!] \in \pi^{\subseteq \pi}$
     - $[\![e]\!] : \pi \to \mathbb{Z}$
     - $[\![b]\!] : \pi \to \mathbb{B}$, gdzie $\mathbb{B} = \{T, F\}$
   - **Przykładowe funkcje semantyczne:**
     - $[\![\text{skip}]\!](\pi) = \pi$
     - $[\![X := e]\!](\pi) = \pi\,[X/[\![e]\!](\pi)]$
     - $[\![\text{if } b \text{ then } c]\!](\pi) = \begin{cases} \pi, & \text{gdy } [\![b]\!](\pi) = F \\ [\![c]\!](\pi), & \text{w p.p.} \end{cases}$
     - $[\![\text{while } b \text{ do } c]\!](\pi) = \begin{cases} \pi, & \text{jeżeli } [\![b]\!](\pi) = F \\ [\![\text{while } b \text{ do } c]\!]([\![c]\!]\pi), & \text{w p.p.} \end{cases}$

9 Wykład 03.04.2008

9.1 Składnia

Ostatnio zaczęliśmy badać języki imperatywne[5].

- a – wyrażenia arytmetyczne,
- b – wyrażenia boolowskie,
- c – instrukcje,
- n – stałe całkowitoliczbowe,
- X – identyfikatory,

$$\begin{aligned}
a &::= n \mid X \mid a \oplus a \\
\oplus &::= + \mid - \mid \times \mid \mathbf{div} \mid \mathbf{mod} \\
b &::= a_1 \otimes a_2 \mid \neg b \mid b_1 \oslash b_2 \\
\oslash &::= \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \\
\otimes &::= < \mid > \mid \leq \mid \geq \mid = \mid \neq \\
c &::= \mathbf{skip} \mid \mathbf{abort} \mid X := a \mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{if}\ b\ \mathbf{then}\ c \mid \mathbf{while}\ b\ \mathbf{do}\ c \mid c_1; c_2
\end{aligned}$$

9.2 Semantyka Denotacyjna

1. maszyna abstrakcyjna,
   Bieżący stan pamięci można opisać jako $\Pi : \mathcal{X} \to \mathbb{Z}$, a zbiór stanów pamięci to $\Pi = \mathbb{Z}^{\mathcal{X}}$.

2. Denotacją (znaczeniem) programu jest funkcja zmiany zawartości pamięci $[\![c]\!] : \Pi \to \Pi$ (funkcja z pamięci w pamięć). Mamy tu funkcje częściowe.

[5] z lekkim obrzydzeniem (w kontekście Haskella)

3. Dziedzina interpretacji programów: $\Pi^{\subseteq\Pi}$

Mały przykład: $[\![\text{while } \mathit{true} \text{ do skip}]\!] = \phi = \bot$

Intuicja: $Dom([\![c]\!]) = \{\pi \in \Pi : \text{program uruchomiony w stanie } \pi \text{ się zatrzyma}\}$

Denotacje wyrażeń arytmetycznych i boolowskich są proste:

- $[\![a]\!] \subseteq \mathbb{Z}^{\Pi}$ – całkowite,
- $[\![b]\!] \subseteq \mathbb{B}^{\Pi}$ – całkowite ($\mathbb{B} = \{T, F\}$),

9.2.1 Funkcja semantyczna

- $[\![\text{skip}]\!]\pi = \pi$
- $[\![\text{abort}]\!] = \bot$
- $[\![X := a]\!](\pi) = \pi[X/[\![a]\!](\pi)]$
- $[\![c_1; c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]$
- $[\![\text{if } b \text{ then } c]\!](\pi) = \begin{cases} \pi, & \text{gdy } [\![b]\!](\pi) = F \\ [\![c]\!](\pi), & \text{w p.p.} \end{cases}$
- $[\![\text{if } b \text{ then } c_1 \text{ else } c_2]\!](\pi) = \begin{cases} [\![c_1]\!](\pi), & \text{jeśli } [\![b]\!](\pi) = T \\ [\![c_2]\!](\pi), & \text{jeśli } [\![b]\!](\pi) = F \end{cases}$
- $[\![\text{while } b \text{ do } c]\!](\pi) = \begin{cases} \pi, & \text{jeżeli } [\![b]\!](\pi) = F \\ [\![\text{while } b \text{ do } c]\!]([\![c]\!]\pi), & \text{w p.p.} \end{cases}$

Z denotacją funkcji while mamy pewne problemy. Spójrzmy na równości opisujące funkcję „silnia”:

$$(*) \begin{cases} 0! = 1 \\ n! = n \times (n-1)!, \ n > 0 \end{cases}$$

W jakim sensie te równości są definicją funkcji silnia?
Istnieje dokładnie jedna funkcja całkowita $f : \mathbb{N} \to \mathbb{N}$. Tę jedyną funkcję nazywamy funkcję definiowaną przez równości $(*)$.

Problemy:

1. $(*) \begin{cases} f(0) = 1 \\ f(2n) = 2n \times (2n-1) \times f(2n-2), \ n > 0 \end{cases}$

   Funkcja „silnia” spełnia te równości.
   Ale jest jeszcze continuum innych funkcji całkowitych $g : \mathbb{N} \to \mathbb{N}$ spełniających te równości.

2. Problem poważniejszy:
   $(*) \begin{cases} f(0) = 1 \\ f(1) = 13 \\ f(n) = n \times f(n-1), \ n > 0 \end{cases}$

   Nie istnieje żadna funkcja spełniająca te równości!

Rozważmy funkcje częściowe $f : D \to \mathbb{N}$, gdzie $\{2n \mid n \in \mathbb{N}\} \subseteq D$. Czy istnieje wśród nich najmniejsza funkcja?
Intuicja: Jeżeli zbiór takich funkcji jest niepusty, to zawsze będzie w nim istnieć najmniejsza funkcja.

$$\left\langle \mathbb{N}^{\subseteq \mathbb{N}}, \subseteq \right\rangle \text{ – zbiór funkcji częściowych z relacją zawierania.}$$

$$f = \lambda\, n \to \begin{cases} 1, & \text{gdy } n = 0 \\ 2n \times (2n-1) \times f(2n-2), & \text{w p.p.} \end{cases}$$

$$f = \left(\lambda\, g \to \lambda\, n \to \begin{cases} 1, & \text{gdy } n = 0 \\ 2n \times (2n-1) \times f(2n-2), & \text{w p.p.} \end{cases} \right) f$$

Funkcja f spełnia równość $(*)$ wtw gdy jest punktem stałym operatora:

$$\Phi(g)(n) = \begin{cases} 1, & \text{gdy } n = 0 \\ 2n \times (2n-1) \times f(2n-2), & \text{w p.p.} \end{cases}$$

Chcemy pokazać, że istnieje najmniejszy punkt stały tego operatora.

9.2.2 Powtórka z logiki

- Jeżeli $\langle X, \leq \rangle$ jest kratą zupełną, a $\Phi : X \to X$ jest funkcją monotoniczną, to Φ posiada najmniejszy punkt stały.
- Jeżeli $\langle X, \leq \rangle$ jest porządkiem zupełnym, $\Phi : X \to X$ jest ciągła, to Φ ma najmniejszy punkt stały a_0 i $a_0 = \bigvee \{f^n(\perp)\}$.

Fakt. *$\left\langle \mathbb{N}^{\subseteq \mathbb{N}}, \subseteq \right\rangle$ jest porządkiem zupełnym, zaś operator Φ jest ciągły. Zatem Φ posiada najmniejszy punkt stały.*

$$f(\pi) = \begin{cases} \pi, & g(\pi) = F \\ f(h(\pi)), & w\ p.p. \end{cases}$$

Definicja.

$$\Phi(f)(\pi) = \begin{cases} \pi, & \text{gdy } g(\pi) = F \\ f(h(\pi)), & \text{w p.p.} \end{cases}$$

Fakt. *$\left\langle \mathbb{N}^{\subseteq \mathbb{N}}, \subseteq \right\rangle$ jest porządkiem zupełnym, zaś operator Φ jest ciągły. Z twierdzenia o punkcie stałym istnieje funkcja spełniająca równość $(*)$.*

$[\![\cdot]\!] : P \to (\Pi \to \Pi)$

10 Wykład 08.04.2008

10.1 Wstęp

Omówimy sobie inne sposoby zadawania znaczenia (denotacji) języków programowania.
Dana Scott (ok. 1965) – Teoria dziedzin (Algol 60).
Spróbujmy przeprowadzić sobie rozumowanie (zamiast „zaglądać” do świata i stwierdzać, czy coś jest prawdziwe, czy nie). Zbudujmy sobie zatem **system wnioskowania**[6].

Definicja. System wnioskowania to zbiór aksjomatów oraz zbiór reguł wnioskowania.

Najstarszym systemem wnioskowania jest **System Hilbertowski**.

[6] siedzimy sobie w pustelni i myślimy...

10.2 Semantyka operacyjna

Język który w całości opisano semantyką operacyjną to SML.

Definicja. S.O.S. – Strukturalna Semantyka Operacyjna (Gordon Plotkin, ok. 1970)

„Sądy”, które będziemy wypowiadać będą troszkę „skażone” naszym *meta-językiem* (nie będą one do końca formalne).

10.2.1 Semantyka Dużych Kroków

1. $\langle a, \pi \rangle \to n$ ($\pi : X \to \mathbb{Z}$, $\pi \in \Pi$, $n \in \mathbb{N}$),
2. $\langle b, \pi \rangle \to T, F \in \mathbb{B}$,
3. $\langle c, \pi \rangle \to \pi' \in \Pi$,

Definicja. Dowód to ciąg wypowiedzi w języku, o takiej własności, że każda wypowiedź jest aksjomatem, albo konkluzją z poprzednich aksjomatów (i konkluzji) na podstawie reguł wnioskowania. Informatycy lubią patrzeć na dowód jak na drzewo w którego liściach są aksjomaty, a jego wierzchołkach – reguły. W korzeniu jest formuła ψ, która jest tezą tego dowodu.

Chcemy mieć narzędzie, którym udowonimy, że jeśli jakiś program c jest uruchomiony w pamięci π to da jakąś pamięć π' ($\langle c, \pi \rangle \to \pi'$).

Taki system (narzędzie) będzie zdefiniowany indukcyjnie (to będzie indukcja strukturalna – system sterowany składnią).

Rozważmy sobie język:

$$
\begin{aligned}
a &::= n \mid X \mid a \oplus a \\
\oplus &::= + \mid - \mid \times \mid \mathbf{div} \mid \mathbf{mod} \\
b &::= a_1 \otimes a_2 \mid \neg b \mid b_1 \oslash b_2 \\
\oslash &::= \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \\
\otimes &::= < \mid > \mid \leq \mid \geq \mid = \mid \neq \\
c &::= \mathbf{skip} \mid \mathbf{abort} \mid X := a \mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{if}\ b\ \mathbf{then}\ c \mid \mathbf{while}\ b\ \mathbf{do}\ c \mid c_1; c_2
\end{aligned}
$$

I mamy kilka przykładowych jego reguł wnioskowania:

- $\dfrac{}{\langle X, \pi \rangle \to \pi(X)}$,
- $\dfrac{}{\langle n, \pi \rangle \to n}$,
- $\dfrac{\langle a_1, \pi \rangle \to n_1 \quad \langle a_2, \pi \rangle \to n_2}{\langle a_1 \oplus a_2, \pi \rangle \to n}$, $n = n_1 \oplus n_2$,
- $\dfrac{\langle b_1, \pi \rangle \to T \quad \langle b_2, \pi \rangle \to T}{\langle b_1 \vee b_2, \pi \rangle \to T}$,
- $\dfrac{\langle b_1, \pi \rangle \to T \quad \langle b_2, \pi \rangle \to F}{\langle b_1 \vee b_2, \pi \rangle \to T}$,
- $\dfrac{\langle b_1, \pi \rangle \to F \quad \langle b_2, \pi \rangle \to T}{\langle b_1 \vee b_2, \pi \rangle \to T}$,
- $\dfrac{\langle b_1, \pi \rangle \to F \quad \langle b_2, \pi \rangle \to F}{\langle b_1 \vee b_2, \pi \rangle \to F}$,

A jakby to wyglądało w C (albo Haskellu) – chodzi o kolejność wykonywania tych działań:

- $\dfrac{\langle b_1, \pi \rangle \to T}{\langle b_1 \vee b_2, \pi \rangle \to T}$,
- $\dfrac{\langle b_1, \pi \rangle \to F \quad \langle b_2, \pi \rangle \to x}{\langle b_1 \vee b_2, \pi \rangle \to x}$, $x \in \{T, F\}$,

Zmierzamy do opisania semantyki operacyjnej dla instrukcji:

- $\dfrac{}{\langle \text{skip}, \pi \rangle \to \pi}$,
- $\dfrac{}{\langle \text{abort}, \pi \rangle \to \textit{wybuchnij!}}$, abort definiujemy przez **brak jego definicji**!,
- $\dfrac{\langle a, \pi \rangle \to n}{\langle X := a, \pi \rangle \to \pi[X/n]}$,
- $\dfrac{\langle c_1, \pi \rangle \to \pi' \quad \langle c_2, \pi' \rangle \to \pi''}{\langle c_1; c_2, \pi \rangle \to \pi''}$,
- $\dfrac{\langle b, \pi \rangle \to F}{\langle \text{if } b \text{ then } c, \pi \rangle \to \pi}$,
- $\dfrac{\langle b, \pi \rangle \to T \quad \langle c, \pi \rangle \to \pi'}{\langle \text{if } b \text{ then } c, \pi \rangle \to \pi'}$,
- $\dfrac{\langle b, \pi \rangle \to T \quad \langle c, \pi \rangle \to \pi' \quad \langle \text{while } b \text{ do } c, \pi' \rangle \to \pi''}{\langle \text{while } b \text{ do } c, \pi \rangle \to \pi''}$,
- $\dfrac{\langle b, \pi \rangle \to F}{\langle \text{while } b \text{ do } c, \pi \rangle \to \pi}$,

10.2.2 Semantyka Małych Kroków

$$\langle c, \pi \rangle \to \langle c', \pi' \rangle$$

A co jak program się zakończy?

$$\langle c, \pi \rangle \to \pi'$$

Rozważmy poprzedni język programowania i zadajmy mu reguły wnioskowania dla semantyki małych kroków:

- $\dfrac{}{\langle \text{skip}, \pi \rangle \to \pi}$,
- $\dfrac{\langle a, \pi \rangle \overset{*}{\to} n}{\langle X := a, \pi \rangle \to \pi[X/n]}$,
- $\dfrac{}{\langle a, \pi \rangle \to \langle a', \pi \rangle}$,

Zdefiniujmy „$\overset{*}{\to}$":

- $\dfrac{}{\langle a, \pi \rangle \to n}$,
- $\dfrac{\langle a, \pi \rangle \to n}{\langle a, \pi \rangle \overset{*}{\to} n}$,
- $\dfrac{\langle a, \pi \rangle \to \langle a', \pi \rangle \quad \langle a', \pi \rangle \overset{*}{\to} n}{\langle a, \pi \rangle \overset{*}{\to} n}$,

Wyrażenia arytmetyczne:

- $\dfrac{}{\langle X, \pi \rangle \to \pi(X)}$,
- $\dfrac{}{\langle n, \pi \rangle \to n}$,
- $\dfrac{}{\langle n_1 \oplus n_2, \pi \rangle \to n}$, $n = n_1 \oplus n_2$,
- $\dfrac{\langle a_1, \pi \rangle \to \langle a_1' \pi \rangle}{\langle a_1 \oplus a_2, \pi \rangle \to \langle a_1' \oplus a_2, \pi \rangle}$,
- $\dfrac{\langle a_2, \pi \rangle \to \langle a_2', \pi \rangle}{\langle n_1 \oplus a_2, \pi \rangle \to \langle n_1 \oplus a_2', \pi \rangle}$,

A jak to będzie ze złożeniem instrukcji?

- $\dfrac{\langle c_1, \pi \rangle \to \langle c_1', \pi' \rangle}{\langle c_1; c_2, \pi \rangle \to \langle c_1'; c_2, \pi' \rangle}$,
- $\dfrac{\langle c_1, \pi \rangle \to \pi'}{\langle c_1; c_2, \pi \rangle \to \langle c_2, \pi' \rangle}$,

A co z while'm?

- $$\frac{\langle b,\pi\rangle \overset{*}{\to} T \qquad \langle c,\pi\rangle \overset{*}{\to} \pi'}{\langle \text{while } b \text{ do } c,\pi\rangle \to \langle \text{while } b \text{ do } c,\pi'\rangle}$$

- $$\frac{\langle b,\pi\rangle \overset{*}{\to} F}{\langle \text{while } b \text{ do } c,\pi\rangle \to \pi}$$

Wzięliśmy sobie pewien język (który wszyscy intuicyjnie rozumieliśmy) i następnie zadaliśmy semantykę denotacyjną: $[\![c]\!](\pi) = \pi'$ (funkcję semantyczną z pamięci w pamięć) i dzisiaj określiliśmy semantykę operacyjną $\langle c,\pi\rangle \to \pi'$. Mamy zatem...

Twierdzenie. O równoważności semantyk:

$$[\![c]\!](\pi) = \pi' \quad \Leftrightarrow \quad \vdash \langle c,\pi\rangle \to \pi'$$

11 Wykład 10.04.2008

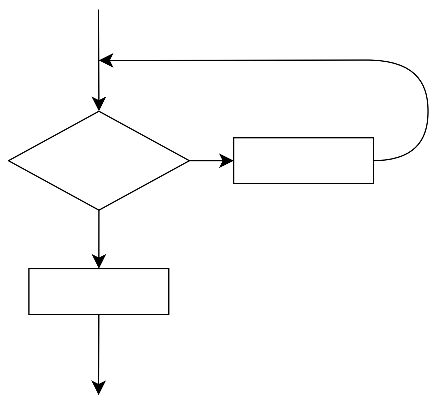
11.1 Wstęp

Semantyka operacyjna (strukturalna) – myślimy sobie o abstrakcyjnej maszynie i pokazujemy krok po kroku co ona zrobi, żeby „dotrzeć" do wyniku:

- wielkich kroków (albo „semantyka naturalna"),
- małych kroków („S.O.S.") – $\langle p,\pi\rangle \overset{1}{\to} \langle p',\pi'\rangle \overset{1}{\to} \langle p',\pi''\rangle \overset{1}{\to} \ldots \overset{1}{\to} \pi^{(n+1)}$,

Mamy też semantykę denotacyjną. Ale to nie wszystko – jest cała masa semantyk.

11.2 Semantyka aksjomatyczna – Floyd



Schemacik blokowy (np. while'a...)

Jest tak, że chcemy móc wnioskować coś o programach – przed wykonaniem programu są spełnione pewne warunki, więc chcemy dowieść, że po jego wykonaniu też będą „jakieś" spełnione[7].
Sformalizujmy sobie tą semantykę:

[7]np. w Macroco$cie budują sobie taki system, żeby dowieść, że Windows nie ma błędów... A poważniej, to udało im się dowieść, że w pewnym systemie nie będzie dereferowania NULL-pointerów (i to jakiś wielki CRAY wymyślił systemem wnioskowania w ciągu miesiąca, czy jakoś...)

- $\vdash \{\varphi\}\ C\ \{\psi\}$, gdzie C to program, lewy nawias wąsaty oznacza, że coś zachodzi przed wykonaniem programu, a prawy nawias wąsaty oznacza, że coś zachodzi po wykonaniu programu,
- termy to będą: $t ::= n \mid X \mid f(t_1, ..., t_2),\ f \in \Sigma = \{+, -, \times, !, ...\}$,
- formuły to: $\varphi ::= R(t_1, ..., t_n) \mid \varphi \overset{\wedge}{\vee} \psi \mid \neg\varphi \mid \forall n\varphi \mid \exists n\varphi$,
- jak u Tarskiego – robimy sobie prawdziwy model (ale z pamięcią): $\mathfrak{M} = \langle \mathbb{Z}, \cdot^{\mathfrak{M}} \rangle$ oraz $\pi, \eta \vDash^{\mathfrak{M}} \varphi$,
- ważne! Chcemy mieć tak:
$$\vdash \{\varphi\}\ C\ \{\psi\}$$
wtedy dla każdej $\pi \in \Pi$ (pamięci), dla każdego η (wartościowania):
$$\text{jeśli } \pi, \eta \vDash \varphi \text{ i } \pi \in Dom[\![C]\!], \text{ to}$$
$$[\![C]\!]\pi, \eta \vDash \psi$$

Ten system wnioskowania będzie troszkę „hybrydowy” – oprócz $\vdash \{\varphi\}\ C\ \{\psi\}$ będziemy mieli jeszcze (chcemy móc zapytać, czy $2 + 2 = 4$):

$$\vDash \varphi \text{ wtw } \pi, \eta \vDash \varphi \text{ dla każdej } \pi \in \Pi \text{ i każdego } \eta$$

- $\dfrac{}{\{\varphi\}\ \text{skip}\ \{\varphi\}}$,
- $\dfrac{}{\{\varphi\}\ \text{abort}\ \{\psi\}}$,
- $\dfrac{}{\{\varphi[X/e]\}\ X := e\ \{\varphi\}}$,
- $\dfrac{\{\varphi\}\ c_1\ \{\psi\} \quad \{\psi\}\ c_2\ \{\varrho\}}{\{\varphi\}\ c_1; c_2\ \{\varrho\}}$,
- $\dfrac{\{b \wedge \varphi\}\ c_1\ \{\psi\} \quad \{\neg b \wedge \varphi\}\ c_2\ \{\psi\}}{\{\varphi\}\ \text{if } b \text{ then } c_1 \text{ else } c_2\ \{\psi\}}$,
- $\dfrac{\{\varphi \wedge b\}\ c\ \{\psi\} \quad \vDash \varphi \wedge \neg b \Rightarrow \psi}{\{\varphi\}\ \text{if } b \text{ then } c\ \{\psi\}}$,
- $\dfrac{\{b \wedge \varphi\}\ c\ \{\varphi\}}{\{\varphi\}\ \text{while } b \text{ do } c\ \{\varphi \wedge \neg b\}}$, gdzie φ jest **niezmiennikiem pętli**,
- reguły osłabiania[8]:
  - $\dfrac{\{\varphi\}\ c\ \{\psi\} \quad \vDash \psi \Rightarrow \varrho}{\{\varphi\}\ c\ \{\varrho\}}$,
  - $\dfrac{\vDash \varphi \Rightarrow \psi \quad \{\psi\}\ c\ \{\varrho\}}{\{\varphi\}\ c\ \{\varrho\}}$,

[8] a.k.a. „system pytania siekierki” (lub „wyroczni”) o to, czy jeśli $x = 2 + 2$ to $x = 4$

12 Wykład 15.04.2008

12.1 Wprowadzenie

Mamy $\vDash 2+2=4$, ale często wystarcza $\vdash 2+2=4$ (bo arytmetyka nie musi być wcale „taka skomplikowana”) do weryfikowania poprawności programów.

Definicja.

- $\vdash \{\varphi\}c\{\psi\} \quad \Rightarrow \quad \vDash \{\varphi\}c\{\psi\}$ – mówimy, że taki system jest **adekwatny**
- $\vDash \{\varphi\}c\{\psi\} \quad \Rightarrow \quad \vdash \{\varphi\}c\{\psi\}$ – mówimy, że taki system jest **zupełny**

Ale z zupełnością jest pewien problem[9].

Weźmy sobie nasz imperatywny język:

$$
\begin{aligned}
a &::= n \mid X \mid a \oplus a \\
\oplus &::= + \mid - \mid \times \mid \mathbf{div} \mid \mathbf{mod} \\
b &::= a_1 \otimes a_2 \mid \neg b \mid b_1 \oslash b_2 \\
\oslash &::= \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \\
\otimes &::= < \mid > \mid \leq \mid \geq \mid = \mid \neq \\
c &::= \mathbf{skip} \mid \mathbf{abort} \mid X := a \mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{if}\ b\ \mathbf{then}\ c \mid \mathbf{while}\ b\ \mathbf{do}\ c \mid c_1; c_2
\end{aligned}
$$

Można dołożyć asercje do języka – każda instrukcja w tym języku będzie miała „z przodu” i „z tyłu” jakieś asercje:

$$
\begin{aligned}
c ::= {} & \{\varphi\}\ \mathbf{skip}\ \{\psi\} \mid \ldots \mid \{\varphi\}\ \mathbf{if}\ b\ \mathbf{then}\ \{\varphi'\}\ c\ \{\varphi''\}\ ;\ \{\psi\} \\
& \mid \{\varphi\}\ \mathbf{while}\ b\ \mathbf{do}\ \{\varphi'\}\ c\ \{\varphi''\}\ ;\ \{\psi\} \\
& \mid \{\varphi\}\ c_1;\ \{\psi\}\ c_2\ \{\varrho\} \\
& \mid \{\varphi\}\{\varphi'\}\ c\ \{\psi\} \\
& \mid \{\varphi\}\ c\ \{\psi\}\{\psi'\}
\end{aligned}
$$

W Haskellu mamy rekonstrukcję typów[10] – zatem nie wszędzie musimy pisać typy. Są jednak miejsca, gdzie takie typy trzeba napisać (bo rekonstrukcja typów jest czasami nierozstrzygalna, ale weryfikacja poprawności już tak – nawet liniowa).

12.2 Asercje

Problem z dowodzeniem poprawności programów polega na tym, że tablica może być za mała... Popatrzmy na taki programik obliczający silnię:

```
⋆  {X = n}
   S := 1
   while X ≠ 1 do
       S := X × S;
       X := X − 1;
   done
⋆  {S = n!}
```

[9] mamy np. niezupełność arytmetyki

[10] system typów Haskella jest dużo bardziej skomplikowany niż w SMLu

Problematyczny dowód to takie drzewo (wieeeelkie, niefajne, nudne, żmudne i paskudne)[11]:

$$\dfrac{\dfrac{}{\text{tu}} \quad \dfrac{\dfrac{}{\textit{inne}}\ \dfrac{\textit{coś...}}{\textit{reguły}}}{\text{możemy}} \quad \dfrac{}{\text{wstawić regułę na złożenie... } \{\varphi\}} \quad \dfrac{\textit{rutututu}}{\textit{blablablabla...}}}{\{X = n\}\ S := 1, \textbf{while } x \neq 1 \textbf{ do } ...\ \textit{blablabla}\ \{S = n!\}}$$

Potrzebujemy jakiegoś lepszego mechanizmu (bo przekonanie kogoś, że program liczy silnię za pomocą takiego drzewka nie jest raczej spektakularnym sukcesem...).

Będziemy sobie robili coś takiego jak „dedukcja asercji" (rekonstrukcja):

```
⋆   {X = n}
⋆   {X = n ∧ 1 = 1}
    S := 1
⋆   {X = n ∧ S = 1}
⋆   {S · X! = n!}
    while X ≠ 1 do
⋆       {S · X! = n! ∧ X ≠ 1}
⋆       {(X · S) · (X − 1)! = n!}
        S := X × S;
⋆       {S · (X − 1)! = n!}
        X := X − 1;
⋆       {S · X! = n!}
    done
⋆   {S · X! = n! ∧ ¬X ≠ 1}
⋆   {S = n!}
```

A co będzie, jeśli ten program wywołamy tak, że w początkowym stanie pamięci będzie miał za X podstawione 0? Oczywiście program się zapętli (X nigdy nie przyjmie wartości 1 – zakładamy, że mamy komórki pamięci o nieskończonej pojemności).

Definicja. Częściowa poprawność:

$$\vDash \{\varphi\}c\{\psi\} \quad \text{wtw} \quad \forall\pi\exists\eta\, (\pi, \eta \vDash \varphi \wedge \pi \in Dom[\![c]\!]) \Rightarrow [\![c]\!]\pi, \eta \vDash \psi$$

Prawdą jest, że przed wykonaniem programu c zachodzi formuła φ a po jego wykonaniu ψ

wtedy i tylko wtedy, gdy

dla każdej pamięci π istnieje wartościowanie η takie, że
jeśli prawdziwa jest formuła φ (przy wybranej pamięci π i wartościowaniu η) oraz program c się zatrzyma
to formuła ψ jest prawdziwa w pamięci $[\![c]\!]\pi$ i przy wartościowaniu η.

Definicja. Całkowita poprawność:

$$\vDash [\varphi]c[\psi] \quad \text{wtw} \quad \forall\pi\exists\eta\, \pi, \eta \Rightarrow (\pi \in Dom[\![c]\!] \wedge [\![c]\!]\pi, \eta \vDash \psi)$$

Prawdą jest, że przed wykonaniem programu c zachodzi formuła φ a po jego wykonaniu ψ

[11] rysowanie tego drzewa zajęło pierwszą godzinę wykładu

wedy i tylko wtedy, gdy

dla każdej pamięci π istnieje wartościowanie η takie, że
program c się zatrzyma i ψ jest prawdziwa w pamięci $[\![c]\!]\pi$ i przy wartościowaniu η.

Definicja. Dobry porządek – taki porządek, w którym nie istnieje żaden nieskończony malejący łańcuch.

Popatrzmy jak udekorować program asercjami w taki sposób, aby mieć dowód całkowitej poprawności:

$\star$ $[X \geq 1 \wedge X = n]$
$\star$ $[X \geq 1 \wedge X = n \wedge 1 = 1]$
`S` $:= 1$
$\star$ $[X \geq 1 \wedge X = n \wedge S = 1]$
$\star$ $[S \cdot X! = n! \wedge X \geq 1]$
<u>**while**</u> `X` $\neq 1$ <u>**do**</u>
  $\star$ $[S \cdot X! = n! \wedge X \geq 1 \wedge X = i \wedge X \neq 1]$
  $\star$ $[(X \cdot S) \cdot (X-1)! = n! \wedge X - 1 \geq 1 \wedge X - 1 < i]$
  `S` $:=$ `X` $\times$ `S`;
  $\star$ $[S \cdot (X-1)! = n! \wedge X - 1 \geq 1 \wedge X - 1 < 1]$
  `X` $:=$ `X` $- 1$;
  $\star$ $[S \cdot X! = n! \wedge X \geq 1 \wedge X < i]$
<u>**done**</u>
$\star$ $[S \cdot X! = n! \wedge X \geq 1 \wedge \neg X \neq 1]$
$\star$ $[S = n!]$

13 Wykład 17.04.2008

13.1 Ciąg dalszy semantyk

Dalej mówimy o semantykach aksjomatycznych. Przyjęliśmy, że „prawdziwym” światem (jego opisem) jest semantyka denotacyjna $[\![c]\!] : \Pi \hookrightarrow \Pi$[12]. Przypomnijmy:

- $\{\varphi\}\ c\ \{\psi\}$ – częściowa poprawność,
- $[\varphi]\ c\ [\psi]$ – całkowita poprawność,

Przed przystąpieniem do jakiegokolwiek zadania programistycznego opisuje się jego specyfikację (w języku naturalnym) – jest to odpowiednikiem opisu odpowiednich formuł φ i ψ. Po co w takim razie robić takie opisy bardzo formalnie? Metody formalne sugerują różne metody programowania – np. pojęcie niezmiennika. Dla przykładu rozważmy algorytm (program) obliczający największy wspólny dzielnik:

[12]konwencja notacyjna jest taka, że używamy symbolu „$\hookrightarrow$” w celu podkreślenia, że mowa o funkcjach **częściowych**

$\star$ $[X = x \wedge Y = y \wedge x > 0 \wedge y \geq 0]$
$\star$ $[\gcd(X,Y) = \gcd(x,y) \wedge X > 0 \wedge Y \geq 0]$
while `Y` $\neq 0$ **do** $[\gcd(X,Y) = \gcd(x,y) \wedge Y > 0 \wedge Y \geq 0 \wedge Y \neq 0 \wedge Y = i]$
$\star$ $\quad [\gcd(Y, X\ mod\ Y) = \gcd(x,y) \wedge Y > 0 \wedge X\ mod\ Y \geq 0]$
$\quad$ `Z = X` **mod** `Y`
$\star$ $\quad [\gcd(Y,Z) = \gcd(x,y) \wedge Y > 0 \wedge Z \geq 0]$
$\quad$ `X = Y`
$\star$ $\quad [\gcd(X,Z) = \gcd(x,y) \wedge X > 0 \wedge Z \geq 0]$
$\quad$ `Y = Z`
$\star$ $\quad [\gcd(X,Y) = \gcd(x,y) \wedge X > 0 \wedge Y \geq 0 \wedge Y < i]$
done
$\star$ $[\gcd(X,Y) = \gcd(x,y) \wedge x > 0 \wedge y \geq 0 \wedge \neg Y \neq 0]$
$\star$ $[X = \gcd(x,y)]$

Początkowo nie mamy pomysłu jak taki program napisać. Pomyślmy sobie o jakimś niezmienniku. Chcemy jakoś tak zmieniać liczby X i Y, żeby nie zaburzyć warunku $\{gcd(X,Y) = gcd(x,y) \wedge x > 0 \wedge y \geq 0\}$[13]. Uprawiamy tu taką (fajną) technikę: nie przejmując się pisanym programem wymyślamy dowód poprawności i dopisujemy odpowiednie fragmenty programu w taki sposób, aby „przepchnąć” dowód. Znajdźmy sobie jakieś równości opisujące największy wspólny dzielnik:

1. $\gcd(X,0) = X,\ X \neq 0$,
2. $\gcd(X,Y) = \gcd(Y, X\ mod\ Y)$, gdzie $Y > 0$ i $X > 0$,

Aby napisane asercje stanowiły dowód **całkowitej** poprawności musimy jeszcze znaleźć pewną miarę, która w czasie działania programu będzie się zmniejszać. W powyższym przykładzie mamy np. Y (stąd warunkiem wejścia do pętli jest m.in. $Y = i$ i tuż przed wykonaniem „obrotu” mamy $Y < i$).

13.2 O algorytmie unifikacji

Algorytm znany z logiki. Obrazuje przydatną technikę wyprowadzania programu (algorytmu) ze specyfikacji.

$$mgu(E) \cdot \Theta = \Theta$$

13.3 Najsłabsze warunki wstępne

$$\mathrm{wp} : P \times \Phi \to \Phi$$

Chcemy mieć najogólniejsze warunki – zbiór pamięci początkowych, które zagwarantują że program będzie spełniał jakiś (konkretny) warunek. Np. popatrzmy na funkcję silnia:

$$\mathrm{wp}(c, S = n!) = \begin{array}{l} (X = n \wedge X > 0) \\ (X = n \wedge n > 0) \\ \ldots \end{array}$$

[13]początkowo pisaliśmy dowód częściowej poprawności i stąd nawiasy „{”, „}”

Problem polega na tym, że mamy $\aleph_0$ różnych warunków wstępnych na różne sposoby opisujących to samo[14]. Żeby się z tym jakoś uporać wprowadzimy klasy abstrakcji. Mamy zatem relację równoważności:

$$\varphi \sim \psi \quad \text{wtw} \quad \vDash \varphi \Leftrightarrow \psi$$

Teraz dla

$$\mathrm{wp} : P \times \Phi/_\sim \to \Phi/_\sim$$

chcemy, żeby

$$\mathrm{wp}(c, \psi) = \text{ najsłabsza formuła } \varphi \text{ taka, że } \vDash [\varphi]\; c\; [\psi].$$

Cóż to takiego najsłabsza formuła[15]?

- $\mathrm{wp}(\mathbf{skip}, \varphi) = \varphi$
- $\mathrm{wp}(\mathbf{abort}, \varphi) = \mathbf{False}$
- $\mathrm{wp}(X := e, \varphi) = \varphi[X/e]$
- $\mathrm{wp}(c_1; c_2, \varphi) = \mathrm{wp}(c_1, \mathrm{wp}(c_2, \varphi))$
- $\mathrm{wp}(\mathbf{if}\; b\; \mathbf{then}\; c, \varphi) = (b \wedge \mathrm{wp}(c, \varphi)) \vee (\neg b \wedge \varphi)$
- $\mathrm{wp}(\mathbf{if}\; b\; \mathbf{then}\; c_1\; \mathbf{else}\; c_2, \varphi) = (b \wedge \mathrm{wp}(c_1, \varphi)) \vee (\neg b \wedge \mathrm{wp}(c_2, \varphi))$

Z while'm oczywiście są problemy:

Definicja.
Niech $\{\varphi_k\}_{k=0}^{\infty}$ – rodzina formuł.
$\bigvee_{k=0}^{\infty} \varphi_k$ oznacza klasę abstrakcji formuł $[\psi]$, takich że $\forall \pi \exists \eta\, (\pi, \eta \vDash \psi \;\Leftrightarrow\; \exists k\, \pi, \eta \vDash \varphi_k)$

- $\mathrm{wp}(\mathbf{while}\; b\; \mathbf{do}\; c, \varphi) = \bigvee_{k=0}^{\infty} H_k$, gdzie $\begin{cases} H_0 & = \neg b \wedge \varphi \\ H_{k+1} & = b \wedge \mathrm{wp}(c, H_k) \end{cases}$ i H_k oznacza formułę, że **while** wykonał k iteracji.

13.4 Termy algebraiczne

$$\Sigma \text{ – sygnatura} \qquad \mathrm{ar} : \Sigma \to \mathbb{N} \qquad \mathcal{X} \text{ – zbiór zmiennych} \qquad \tau(\Sigma, \mathcal{X})$$

1. $x \in \mathcal{X} \Rightarrow x \in \tau(\Sigma, \mathcal{X})$
2. $t_1, ..., t_n \in \tau(\Sigma, \mathcal{X})$ i $f \in \Sigma$ i $\mathrm{ar}(f) = n$,
   to para złożona z f i ciągu $t_1, ..., t_n$ też jest termem z $\tau(\Sigma, \mathcal{X})$
3. Zbiór $\tau(\Sigma, \mathcal{X})$ jest najmniejszym zbiorem spełniającym warunki 1. i 2.

Można powiedzieć, że jest to **składnia abstrakcyjna**. Ale do porozumiewania się potrzebujemy jakiejś **składni konkretnej**. To jest język, który budujemy sobie po to, aby móc mówić coś o jakichś światach. Te światy to algebry.

[14] np. zamiast „umarł pani mąż” górnik mógł powiedzieć „przepraszam... czy to wdowa Kowalska?”...
[15] wp – „weakest precondition”

13.5 Algebry

$$\mathscr{A} = \left\langle A, \cdot^{\mathscr{A}} \right\rangle \text{ – algebra nad sygnaturą } \Sigma$$
$$\text{jeżeli } \cdot^{\mathscr{A}} : \Sigma \to \bigcup_{n \geq 0} (A^n \to A), \text{ gdzie } f^{\mathscr{A}} : A^n \to A \text{ dla } n = \mathrm{ar}(f)$$
$$(\cdot^{\mathscr{A}} \text{ – interpretacja symboli w algebrze}).$$

Definicja.

- $[\![\cdot]\!]^{\mathscr{A}}_{\eta} : \tau(\Sigma, \mathcal{X}) \to A$
- $\begin{cases} [\![\mathcal{X}]\!]^{\mathscr{A}}_{\eta} = \eta(x) \\ [\![f(t_1, ..., t_n)]\!]^{\mathscr{A}}_{\eta} = f^{\mathscr{A}}([\![t_1]\!]^{\mathscr{A}}_{\eta}, ..., [\![t_n]\!]^{\mathscr{A}}_{\eta}) \end{cases}$

14 Wykład 22.04.2008

14.1 Algebra abstrakcyjna

Będziemy ją rozumieć jako opis termów pierwszego rzędu.

14.1.1 Słowo wstępne (TWi)

Matematyka jest sztuką budowania abstrakcyjnych modeli rzeczywistości. Logika matematyczna zajmuje się badaniem sposobów opisu, mówienia i rozumowania na temat rzeczywistości. W świecie istnieją różne obiekty. Gdy staną się one przedmiotem naszego zainteresowania, zaczynamy o nich mówić. Matematycznymi modelami naszych wypowiedzi są *termy* (*wyrażenia*), a zbiorów obiektów, o których mówimy — *algebry*. Wcześniej opisywaliśmy języki w sposób bardzo konkretny, jako ciągi znaków. Obecnie rozważymy metody ich abstrakcyjnego opisu.

14.1.2 Składnia

Term to abstrakcyjny obiekt matematyczny; np: $2 + x$ to para $\langle +, \langle 2, x \rangle \rangle$ (oczywiście należy patrzeć – bądź „lubimy" patrzeć – na tą parę jak na odpowiednie drzewko).

- Zaczynamy od tego, że wybieramy sobie zbiór gatunków $\mathscr{S}$[16].
- Typ algebraiczny: niepusty ciąg gatunków.
  Zapisujemy go przeważnie w postaci $(a_1, ..., a_n) \to b$, gdy $n > 0$, lub b gdy $n = 0$.
- Chcemy klasyfikować symbole funkcyjne w termach: $(+) :: (a, a) \to a$, bądź $(<) :: (a, a) \to b$.
- Zbiór typów: $\mathbb{T}$,
- Sygnatura $\Sigma = \{\Sigma_\tau\}_{\tau \in \mathbb{T}_{\mathscr{S}}}$, $\Sigma_\tau \cap \Sigma_{\tau'}$, gdy $\tau \neq \tau'$
  Czasem też tak: $|\bigcup \Sigma_\tau| < \infty$.
- Zmienne $\{\mathcal{X}_a\}_{a \in \mathscr{S}}$,
- Termy: $\{\tau_a(\Sigma, \mathcal{X})\}_{a \in \mathscr{S}}$
  1. $\mathcal{X}_a \subseteq \tau_a(\Sigma, \mathcal{X})$, dla $a \in \mathscr{S}$,

[16] gatunki to są takie „bazowe" typy, jak np. **int**, **bool** i co kto lubi...

2. jeżeli $t_i \in \tau_{a_i}(\Sigma, \mathcal{X})$ i $f \in \Sigma_{(a_1,...,a_n)\to b}$ to $f(t_1,...,t_n) \in \tau_b(\Sigma, \mathcal{X})$.
3. $\{\tau_a(\Sigma, \mathcal{X})\}_{a\in\mathscr{S}}$ jest rodziną najmniejszych zbiorów spełniających 1. i 2.

- „język while":

$$
\begin{aligned}
\mathscr{A} &::= \mathscr{N} \mid \mathscr{J} \mid \mathscr{A} \oplus \mathscr{A} \\
\oplus &::= + \mid - \mid \times \mid \mathbf{div} \mid \mathbf{mod} \\
\mathscr{B} &::= \mathscr{A} \otimes \mathscr{A} \mid \neg\mathscr{B} \mid \mathscr{B} \otimes \mathscr{A} \mid \mathbf{true} \mid \mathbf{false} \\
\otimes &::= \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \\
\otimes &::= < \mid > \mid \leq \mid \geq \mid = \mid \neq \\
\mathscr{C} &::= \mathbf{skip} \mid \mathbf{abort} \mid \mathscr{J} := \mathscr{A} \mid \mathbf{if}\ \mathscr{B}\ \mathbf{then}\ \mathscr{C}\ \mathbf{else}\ \mathscr{C} \mid \mathbf{if}\ \mathscr{B}\ \mathbf{then}\ \mathscr{C} \mid \mathbf{while}\ \mathscr{B}\ \mathbf{do}\ \mathscr{C} \mid \mathscr{C};\mathscr{C}
\end{aligned}
$$

- to koniec machania rękami, teraz... formalnie:
  - $\mathscr{S} = \{a, b, c, i\}$,
  - $\Sigma_a = \{n : n \in \mathbb{Z}\}$,
  - $\Sigma_{(a,a)\to a} = \{+, -, \times, \mathbf{div}, \mathbf{mod}\}$,
  - $\Sigma_b = \{\mathbf{true}, \mathbf{false}\}$,
  - $\Sigma_{b\to b} = \{\mathbf{not}\}$,
  - $\Sigma_{(b,b)\to b} = \{\mathbf{and}, \mathbf{or}\}$,
  - $\Sigma_{(a,a)\to b} = \{<, \leq, >, \geq, =, \neq\}$,
  - $\Sigma_c = \{\mathbf{skip}, \mathbf{abort}\}$,

  Tu lekko sobie głowę rozbiliśmy i dojrzeliśmy do „lvalue"... potrzebujemy jeszcze:
  - $\Sigma_{i\to a} = \{!\}$,
  - $\Sigma_{(i,a)\to c} = \{:=\}$,
  - $\Sigma_{(b,c)\to c} = \{\mathbf{if}, \mathbf{while}\}$,
  - $\Sigma_{(c,c)\to c} = \{;\}$,
  - $\Sigma_{(b,c,c)\to c} = \{\mathbf{if\ else}\}$,
  - $\Sigma_\tau = \phi$ dla τ nie wymienionych wyżej,

  Programy w języku while $\tau(\Sigma_{\mathbf{while},\phi})$,

14.1.3 Algebra, interpretacje zmiennych, interpretacje termów

- Algebra $\mathscr{A} = \left\langle \{A_a\}_{a\in\mathscr{S}}, \cdot^{\mathscr{A}} \right\rangle$, gdzie:
  - $\{A_a\}_{a\in\mathscr{S}}$ – uniwersum,
  - $\cdot^{\mathscr{A}}$ – interpretacja symboli w algebrze,
- Dla każdego $f \in \Sigma_{(a_1,...,a_n)\to b}$ jego interpretacja $f^{\mathscr{A}} : A_{a_1} \times ... \times A_{a_n} \to A_b$.
- Interpretacje zmiennych $\{\eta_a\}_{a\in\mathscr{S}}$, gdzie $\eta_a : \mathcal{X}_a \to A_a$.

Definicja. Interpretacja termów w algebrze: $\begin{cases} [\![f(t_1,...,t_n)]\!]^{\mathscr{A}}_\eta = f^{\mathscr{A}}([\![t_1]\!]^{\mathscr{A}}_\eta, ..., [\![t_n]\!]^{\mathscr{A}}_\eta) \\ [\![x]\!]^{\mathscr{A}}_\eta = \eta(x), \text{ gdzie } x \in \mathcal{X} \end{cases}$

14.1.4 Semantyka denotacyjna algebraicznie

- $A_a = \mathbb{Z}^{\subseteq\Pi}$, gdzie $\mathbb{Z}$ - zbiór liczb całkowitych
- $A_b = \mathbb{B}^{\subseteq\Pi}$, gdzie $\mathbb{B} = \{T, F\}$,
- $A_c = \Pi^{\subseteq\Pi}$, gdzie $\Pi = \mathbb{Z}^{\mathbb{L}}$,
- $A_i = \mathbb{L}$ i $\mathbb{L}$ – zbiór adresów,

Musimy jeszcze wyinterpretować wszystko co zdefiniowaliśmy wyżej:

- $X^{\mathscr{A}} \in \mathbb{L}$, $X^{\mathscr{A}} = \mathrm{alloc}(X)$, gdzie $\mathrm{alloc} : J \to \mathbb{L}$ jest alokatorem pamięci,
- interpretacja stałej: $n^{\mathscr{A}}(\pi) = n$,
- interpretacja plusa: $+^{\mathscr{A}}(f,g)\pi = f(\pi) + g(\pi)$,
- tak samo z minusem i mnożeniem,
- $\mathbf{div}^{\mathscr{A}}(f,g)\pi = \begin{cases} \lfloor \frac{f(\pi)}{g(\pi)} \rfloor, & \text{gdy } g(\pi) \neq 0 \\ \text{nieokreślone} & \text{w p.p.} \end{cases}$,
- $\mathbf{skip}^{A}\pi = \pi$,
- $\mathbf{abort}^{A}\pi = \text{nieokreślone}$,
- $:=^{\mathscr{A}}(l,f)\pi = \pi[l/f(\pi)]$,
- $;^{\mathscr{A}}(f,g) = g \circ f$,
- $\mathbf{if}^{\mathscr{A}}(f,g)\pi = \begin{cases} \pi, & \text{gdy } \pi \in Dom(f) \text{ i } f(\pi) = F \\ g(\pi), & \text{gdy } \pi \in Dom(f) \cap Dom(g) \text{ i } f(\pi) = T \\ \text{nieokreślone}, & \text{w p.p.} \end{cases}$
- $\mathbf{while}^{\mathscr{A}}(f,g)\pi = \begin{cases} \pi, & \text{gdy } \pi \in Dom(f) \wedge f(\pi) = F \\ \mathbf{while}^{\mathscr{A}}(f,g)(g(\pi)), & \text{gdy } \pi \in Dom(f) \cap Dom(g) \wedge f(\pi) = T \\ \text{nieokreślone}, & \text{w p.p.} \end{cases}$

  a formalnie

  $\mathbf{while}^{\mathscr{A}} = \mathrm{Fix}\Phi$, gdzie $(\Phi h)(f,g)\pi = \begin{cases} \pi, & \text{gdy } \pi \in Dom(f) \wedge f(\pi) = F \\ h(f,g)(g\pi), & \text{gdy } \pi \in Dom(f) \cap Dom(g) \wedge f(\pi) = T \\ \text{nieokreślone} & \text{w p.p.} \end{cases}$

Ten język jest strasznie prosty, ale pozwala robić skomplikowane rzeczy...

- $\{\tau_a(\Sigma, \mathcal{X})\}_{a \in \mathscr{S}}$,

Podstawienie: skończony zbiór par $[x_i/t_i]_{i=1}^n$ taki, że dla każdego $i = 1..n$ gatunek x_i jest równy gatunkowi termu t_i oraz $x_i \neq x_j$ dla $i \neq j$.

- $\Theta_a : \mathcal{X}_a \to \tau_a(\mathcal{X}, \Sigma)$ gdzie $\Theta_a(y) = \begin{cases} t_i, & \text{gdy } x_i = y \text{ dla pewnego } i \\ y, & \text{w p.p.} \end{cases}$,
- $\Theta_a : \tau_a(\Sigma, \mathcal{X}) \to \tau_a(\Sigma, \mathcal{X}) \begin{cases} x\Theta_a = ... \\ f(t_1, ..., t_n)\Theta_a = f(t_1\Theta_a, ..., t_n\Theta_a) \end{cases}$,

14.1.5 Sygnatury i termy (TWi)

Niech $\mathcal{S} \neq \varnothing$ będzie niepustym (przeważnie skończonym) zbiorem *gatunków* (*rodzajów*, *sorts*). Jego elementy zwykle oznaczamy literami a, b itd, niekiedy z indeksami. Skończony niepusty ciąg gatunków $\langle a_1, \ldots, a_n, b\rangle$ dla $n \geq 0$ i $a_1, \ldots, a_n, b \in \mathcal{S}$ nazywamy *typem algebraicznym* (krócej *typem*), oznaczamy σ, τ, ρ itd, niekiedy z indeksami i zapisujemy w postaci $a_1 \times \ldots \times a_n \to b$ dla $n > 0$ oraz b dla $n = 0$. Liczbę n nazywamy *arnością* typu. Zbiór typów algebraicznych oznaczamy $\mathbb{T}_1(\mathcal{S})$. Z każdym typem τ związujemy zbiór Σ^τ *symboli typu* τ. Symbole typu τ oznaczamy f^τ, g^τ itd, niekiedy z indeksami. *Arnością* (*liczbą argumentów*) symbolu nazywamy arność jego typu. Symbole o arności 0, tj. typu $\tau = a \in \mathcal{S}$ nazywamy *stałymi* i oznaczamy c^a, d^a itd, niekiedy z indeksami. Symbole o arności 1 nazywamy *unarnymi*, symbole o arności 2 zaś *binarnymi*. Z każdym gatunkiem $a \in \mathcal{S}$ związujemy (zwykle przeliczalny nieskończony) zbiór $\mathcal{X}^a$ *zmiennych gatunku* a. Zmienne gatunku a oznaczamy x^a, y^a, z^a itd, niekiedy z indeksami. Zakładamy, że zbiory Σ^τ i $\mathcal{X}^a$ są parami rozłączne, choć czasem dopuszczamy pewne wyjątki. Rodzinę $\Sigma = \{\Sigma^\tau\}_{\tau \in \mathbb{T}_1(\mathcal{S})}$ nazywamy *sygnaturą algebraiczną* (krócej *sygnaturą*), rodzinę $\mathcal{X} = \{\mathcal{X}^a\}_{a \in \mathcal{S}}$ zaś *rodziną zmiennych*. Gdy nie prowadzi to do nieporozumień, pomijamy oznaczenie typu lub gatunku i piszemy f, c, x zamiast f^τ, c^a, x^a. Piszemy też $x \in \mathcal{X}$ na oznaczenie faktu, że $x \in \mathcal{X}^a$ dla pewnego $a \in \mathcal{S}$ itp.

Zbiory *termów* (*wyrażeń*) gatunku $a \in \mathcal{S}$ nad sygnaturą Σ i zbiorem zmiennych $\mathcal{X}$, oznaczane $\mathcal{T}^a(\Sigma, \mathcal{X})$ dla $a \in \mathcal{S}$, definiujemy indukcyjnie:

1. każda zmienna gatunku a jest termem tego gatunku, tj. $\mathcal{X}^a \subseteq \mathcal{T}^a(\Sigma, \mathcal{X})$;
2. jeżeli $t_i \in \mathcal{T}^{a_i}(\Sigma, \mathcal{X})$ dla $i = 1, \ldots, n$ i $n \geq 0$ oraz $f \in \Sigma^{a_1 \times \ldots \times a_n \to b}$, to para złożona z symbolu f i ciągu termów $\langle t_1, \ldots, t_n\rangle$ jest termem gatunku b, tj.
$$\langle f, \langle t_1, \ldots, t_n\rangle\rangle \in \mathcal{T}^b(\Sigma, \mathcal{X})$$
3. każdy term można zbudować używając reguł 1 i 2.

Term $\langle f, \langle t_1, \ldots, t_n\rangle\rangle$ przeważnie zapisujemy w notacji prefiksowej z nawiasami, tj. w postaci $f(t_1, \ldots, t_n)$, choć dla niektórych symboli binarnych przyjmujemy notację infiksową. Termy przedstawiamy także graficznie w postaci drzewa o wierzchołkach etykietowanych symbolami z sygnatury. Jeżeli c jest symbolem o arności 0 (stałą), to term złożony z tego symbolu zapisujemy po prostu jako c. Nie rozpatrujemy osobno przypadku symboli o arności 0. Pisząc „term $f(t_1, \ldots, t_n)$ dla $n \geq 0$" mamy na myśli term $f(t_1, \ldots, t_n)$ gdy $n > 0$ i term f, gdy $n = 0$. Podobnie $a_1 \times \ldots \times a_n \to b$ dla $n = 0$ oznacza typ b. Używając skróconego zapisu sumy zbiorów $\bigcup A_i$ przyjmujemy, że suma pustej rodziny zbiorów jest zbiorem pustym.

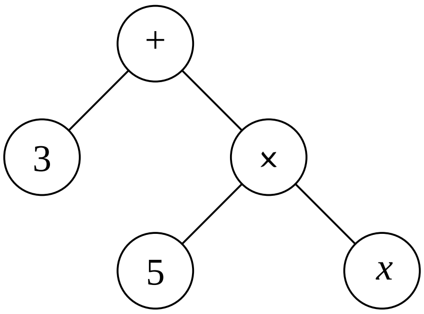
Przykład. Rozważmy zbiór gatunków $\mathcal{S} = \{N, B\}$, sygnaturę

$$\begin{array}{rcl}
\Sigma^N & = & \{0, 1, 2, \ldots\} \\
\Sigma^B & = & \{T, F\} \\
\Sigma^{B \to B} & = & \{\neg\} \\
\Sigma^{N \times N \to N} & = & \{+, \times\} \\
\Sigma^{N \times N \to B} & = & \{\leq, \geq, <, >, =, \neq\} \\
\Sigma^{B \times B \to B} & = & \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}
\end{array}$$

przy czym zbiory Σ^τ dla pozostałych typów τ są puste, oraz zbiory zmiennych

$$\begin{array}{rcl}
\mathcal{X}^N & = & \{x, y, z, \ldots\} \\
\mathcal{X}^B & = & \{p, q, r, \ldots\}
\end{array}$$

Przykładem termu gatunku N jest $+(3, \times(5, x))$. Wygodniej go jednak zapisać w postaci $3+5\times x$. Jego graficzna reprezentacja jest przedstawiona na rysunku. Przykładem termu gatunku B jest $\leq(x, +(4, y))$. Możemy go czytelniej zapisać w postaci $x \leq 4 + y$.



Graficzne przedstawienie termu

Przykład. Rozważmy zbiór gatunków $\mathcal{S} = \{\textit{obiekt}, \textit{fakt}\}$, sygnaturę

$$\begin{aligned}
\Sigma^{\textit{obiekt}} &= \{\text{pies}, \text{kot}, \text{stół}\}\\
\Sigma^{\textit{obiekt}\to\textit{obiekt}} &= \{\text{mały}, \text{wysoki}, \text{brzydki}\}\\
\Sigma^{\textit{obiekt}\to\textit{fakt}} &= \{\text{biegnie}, \text{śpi}\}\\
\Sigma^{\textit{obiekt}\times\textit{obiekt}\to\textit{fakt}} &= \{\text{gryzie}, \text{liże}\}
\end{aligned}$$

przy czym zbiory Σ^τ dla pozostałych typów τ są puste, oraz zbiory zmiennych

$$\begin{aligned}
\mathcal{X}^{\textit{obiekt}} &= \{X, Y, Z, \ldots\}\\
\mathcal{X}^{\textit{fakt}} &= \{P, Q, R, \ldots\}
\end{aligned}$$

Przykładami termów gatunku *obiekt* są

$$\begin{array}{ccc}
\text{pies} & \text{mały}(\text{pies}) & \text{wysoki}(\text{brzydki}(\text{kot}))\\
\text{mały}(\text{mały}(\text{stół})) & X & \text{mały}(X)
\end{array}$$

zaś gatunku *fakt* są

$$\begin{array}{ccc}
\text{biegnie}(\text{wysoki}(\text{stół})) & \text{śpi}(\text{brzydki}(\text{kot})) & \text{liże}(\text{mały}(\text{pies}), \text{kot})\\
\text{gryzie}(X, \text{brzydki}(X)) & Q & \text{liże}(X, Y)
\end{array}$$

Termy zdefiniowane wyżej nazywamy *termami pierwszego rzędu*. Za ich pomocą nie można wyrazić *kwantyfikacji* (*wiązania*) zmiennych. W teorii języków programowania wielkie znaczenie mają termy *wyższych rzędów*, tzw. *lambda termy* lub *lambda wyrażenia*, wyposażone w mechanizm wiązania zmiennych. Są one uniwersalnym językiem, w którym można wyrazić całą matematykę.

14.1.6 Termy nad pojedynczym gatunkiem (TWi)

Bardzo często rozważa się szczególny przypadek, gdy zbiór gatunków $\mathcal{S}$ jest jednoelementowy. Wówczas typy różnią się jedynie arnością i nie potrzeba ich używać. Mamy wtedy jeden zbiór zmiennych $\mathcal{X}$, zbiory Σ_n *symboli o arności* n, sygnaturę (zwaną sygnaturą *jednogatunkową*) $\Sigma = \{\Sigma_n\}_{n\geq 0}$ i jeden zbiór termów $\mathcal{T}(\Sigma, \mathcal{X})$, zadany następującą definicją indukcyjną:

1. $\mathcal{X} \subseteq \mathcal{T}(\Sigma, \mathcal{X})$;
2. jeżeli $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$ dla $n \geq 0$ oraz $f \in \Sigma_n$, to $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})$;
3. każdy term można zbudować używając reguł 1 i 2.

Sygnaturę nad więcej niż jednoelementowym zbiorem gatunków będziemy nazywać sygnaturą *wielogatunkową*.

15 Wykład 24.04.2008

15.1 O unifikacji

W 1965 r. stało się coś bardzo interesującego – urodził się prof. Marcinkowski... Ale nie było to najważniejszym wydarzeniem tego roku[17]. W tym roku Alan Robinson zaczął sobie myśleć o automatycznym dowodzeniu twierdzeń i odkrył unifikację.

- o skolemizacji...
- $\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} R_{i,j}(t_1, ..., t_k))$
- o rezolucji...

Powiedzieliśmy, że podstawienie to taki skończony zbiór par:

- $\theta = [x_i/t_i]_{i=1}^{n} \quad x_i \in \mathcal{X},\ t_i \in \tau(\Sigma, \mathcal{X}),\ x_i \neq x_j$ dla $i \neq j$,
- podstawienie będziemy zapisywać postfiksowo: $t\theta_1\theta_2$,
- $\begin{cases} x\theta = t_i, & \text{gdy } x = x_i \\ x\theta = x, & \text{gdy } x \neq x_i \text{ dla } i = 1, ..., n \\ f(t_1, ..., t_n)\theta = f(t_1\theta, ..., t_n\theta), & \end{cases}$,
- $Dom\theta = \{x_1, ..., x_n\}$,
- $\theta : \tau(\Sigma, \mathcal{X}) \to (\Sigma, \mathcal{X})$,
- $\iota = [] \qquad \iota : \tau(\Sigma, \mathcal{X}) \to (\Sigma, \mathcal{X})$ – funkcja identycznościowa,
- θ_1, θ_2 – podstawienie,
- $\theta_1 \cdot \theta_2$ – złożenie df: $t(\theta_1\theta_2) = (t\theta_1)\theta_2$,
- $Dom(\theta_1) \cap Dom(\theta_2) = \phi$ to $[x_i/t_i] \cup [y_j/s_j] = [x_i/t_i, y_j/s_j]$,
- Obserwacja: nie zawsze $\theta_1 \cdot \theta_2 = \theta_1 \cup \theta_2$. Jest tak gdy: $\bigcup_{x \in Dom(\theta_1)} FV(x\theta_1) \cap Dom(\theta_2) = \phi$,
- $(S, \cdot, \iota)$ – półgrupa z jedynką = monoid (i S – rodzina podstawień),

Definicja. Bardzo ważna. Podstawienie θ_1 jest co najmniej tak ogólne jak θ_2 jeżeli istnieje podstawienie ϱ takie, że $\theta_1\varrho = \theta_2$. Wtedy piszemy, że $\theta_1 \leq \theta_2$.

Definicja. Term t_1 jest co najmniej tak ogólny jak t_2 jeżeli istnieje ϱ takie, że $t_1\varrho = t_2$. Piszemy wtedy $t_1 \leq t_2$.

Fakt. *$\leq$ na podstawieniach oraz $\leq$ na termach są częściowymi praporządkami.*

Definicja.

- $\theta_1 \sim \theta_2$ wtw $\theta_1 \leq \theta_2 \leq \theta_1$ to $\langle S/_\sim, \cdot, \iota, \leq \rangle$ – monoid z porządkiem,
- $t_1 \sim t_2$ wtw $t_1 \leq t_2 \leq t_1$ to $\langle \tau(\Sigma, \mathcal{X}/_\sim), \leq \rangle$ – zbiór uporządkowany,

Widzimy, że mamy tu bardzo ciekawą przestrzeń o bardzo ciekawej strukturze.

Definicja. Unifikator termów t_1 i t_2 – podstawienie θ takie, że $t_1\theta = t_2\theta$.

[17] choć znaczącym na pewno...

15.2 Podstawienie (TWi)

Nie wyjaśniliśmy do tej pory roli zmiennych w termach. *Zbiór zmiennych termu* t, oznaczany $\mathrm{FV}(t)$ (od *free variables*; tu akurat wszystkie zmienne są wolne), definiujemy indukcyjnie:

$$\begin{aligned} \mathrm{FV}(x) &= \{x\}, \qquad \text{dla } x \in \mathcal{X} \\ \mathrm{FV}(f(t_1,\ldots,t_n)) &= \mathrm{FV}(t_1) \cup \ldots \cup \mathrm{FV}(t_n) \end{aligned}$$

Termy nie zawierające zmiennych nazywamy *termami stałymi* (*ground terms*). Zatem term t jest stały, gdy $\mathrm{FV}(t) = \varnothing$. Zbiór termów stałych gatunku a to $\mathcal{T}^a(\Sigma,\varnothing)$. Termy stałe, np. mały(pies) nazywają ustalone obiekty. Zmienne w termach pozwalają opisać całe zbiory takich obiektów, np. mały(X) nie nazywa pojedynczego obiektu, tylko jest *schematem*, opisuje zbiór wszystkich termów mały(t), gdzie t jest dowolnym termem gatunku *obiekt*, np. mały(kot), mały(pies) itd. Term mały(t) otrzymujemy *podstawiając* term t w miejsce zmiennej X w termie mały(X). Formalnie *podstawienie* jest skończonym zbiorem par zmiennych i termów zapisywanym w postaci

$$\theta = [x_1/t_1,\ldots,x_n/t_n]$$

gdzie $x_i \in \mathcal{X}^{a_i}$ i $t_i \in \mathcal{T}^{a_i}(\Sigma,\mathcal{X})$, dla $i = 1,\ldots,n$. Zwróćmy uwagę, że gatunek zmiennej x_i musi się zgadzać z gatunkiem termu t_i. Wynik podstawienia $\theta = [x_1/t_1,\ldots,x_n/t_n]$ w termie t oznaczamy $t\theta$ (a więc w tzw. zapisie postfiksowym) i definiujemy indukcyjnie:

$$\begin{aligned} x_i\theta &= t_i, \quad \text{dla } i = 1,\ldots,n \\ y\theta &= y, \quad \text{dla } y \in \mathcal{X},\ y \neq x_i \text{ dla } i = 1,\ldots,n \\ f(s_1,\ldots,s_m)\theta &= f(s_1\theta,\ldots,s_m\theta) \end{aligned}$$

Dla przykładu mały$(X)[X/\text{kot}]$ = mały(kot), zaś gryzie$(X,Y)[Y/X]$ = gryzie(X,X). Zatem na podstawienie można patrzeć jak na odwzorowanie

$$\theta : \bigcup_{a\in\mathcal{S}} \mathcal{T}^a(\Sigma,\mathcal{X}) \to \bigcup_{a\in\mathcal{S}} \mathcal{T}^a(\Sigma,\mathcal{X})$$

które termom przyporządkowuje termy (tego samego gatunku).

Fakt. *Jeżeli* $t \in \mathcal{T}^a(\Sigma,\mathcal{X})$ *i* θ *jest podstawieniem, to* $t\theta \in \mathcal{T}^a(\Sigma,\mathcal{X})$.

Podstawienia można *składać*, tak jak wszelkie odwzorowania. Formalnie *złożeniem podstawień* θ_1 i θ_2 nazywamy podstawienie zapisywane $\theta_1\theta_2$, takie że

$$t(\theta_1\theta_2) = (t\theta_1)\theta_2$$

dla każdego termu $t \in \bigcup_{a\in\mathcal{S}} \mathcal{T}^a(\Sigma,\mathcal{X})$.

Zbiór zmiennych $\mathrm{Dom}(\theta) = \{x_1,\ldots,x_n\}$ nazywamy *dziedziną* (*nośnikiem*) podstawienia $\theta = [x_1/t_1,\ldots,x_n/t_n]$. Podstawienie *identycznościowe* (o pustej dziedzinie) oznaczamy $[]$. Dla każdego termu $t \in \bigcup_{a\in\mathcal{S}} \mathcal{T}^a(\Sigma,\mathcal{X})$ zachodzi $t[] = t$. Podstawienie $[]$ jest zatem faktycznie identycznością. Dla podstawień

$$\begin{aligned} \theta_1 &= [x_1/t_1,\ldots,x_n/t_n] \\ \theta_2 &= [y_1/s_1,\ldots,y_m/s_m] \end{aligned}$$

o rozłącznych dziedzinach definiujemy ich *sumę* wzorem

$$\theta_1 \cup \theta_2 = [x_1/t_1,\ldots,x_n/t_n,y_1/s_1,\ldots,y_m/s_m]$$

Lemat. *(O podstawianiu) Jeżeli podstawienia* $\theta_1 = [x_1/t_1, \ldots, x_n/t_n]$ *i* θ_2 *mają rozłączne dziedziny, to*

$$\theta_1\theta_2 \quad = \quad [x_1/t_1\theta_2, \ldots, x_n/t_n\theta_2] \cup \theta_2$$

Mówimy, że term s jest *ukonkretnieniem* (*konkretyzacją, instancją*) termu t, jeżeli istnieje podstawienie θ, takie, że $t\theta = s$.

Na podstawieniach wprowadzamy relację częściowego praporządku $\leq$: mówimy, że podstawienie θ_1 jest *co najmniej tak ogólne*, jak podstawienie θ_2, co zapisujemy $\theta_1 \leq \theta_2$, jeżeli istnieje podstawienie ρ, takie że $\theta_1\rho = \theta_2$. Niech $\theta_1 \sim \theta_2$ jeśli $\theta_1 \leq \theta_2$ i $\theta_2 \leq \theta_1$.

Fakt. *Relacja* $\leq$ *na podstawieniach jest częściowym praporządkiem (jest zwrotna i przechodnia), zaś* $\sim$ *jest relacją równoważności. Relacja* $\leq$ *na klasach równoważności*

$$[\theta_1]_\sim \leq [\theta_2]_\sim \quad \Longleftrightarrow \quad \theta_1 \leq \theta_2$$

jest poprawnie określona i jest częściowym porządkiem (jest zwrotna, przechodnia i słabo antysymetryczna). Od tej pory będziemy często utożsamiać podstawienia równoważne i de facto *rozważać nie podstawienia, tylko ich klasy abstrakcji modulo* $\sim$. *Każda para podstawień ma wówczas kres dolny, tj.* najmniej ogólne uogólnienie*, oznaczane* $\theta_1 \wedge \theta_2$ *(rodzina podstawień z relacją* $\leq$ *jest* dolną półkratą*). Elementem najmniejszym jest podstawienie identycznościowe* $[]$. *Kres górny pary podstawień nie zawsze istnieje, elementu największego w zbiorze podstawień nie ma (z wyjątkiem przypadków o zdegenerowanej sygnaturze).*

15.3 Algorytm unifikacji (TWi)

Równaniem nazywamy parę termów tego samego gatunku. Równanie zapisujemy zwykle w postaci $t \stackrel{?}{=} s$. Zbiór równań nazywamy *układem równań*. Podstawienie θ, takie że $t\theta = s\theta$ nazywamy *unifikatorem* pary termów t i s. Ogólniej, unifikatorem układu równań $\{t_i \stackrel{?}{=} s_i\}_{i=1}^n$ nazywamy podstawienie θ, takie że $t_i\theta = s_i\theta$ dla $i = 1, \ldots, n$. Najbardziej ogólny unifikator termów t i s oznaczamy $\mathrm{mgu}(t, s)$ (*most general unifier*). Najbardziej ogólny unifikator układu równań oznaczamy $\mathrm{mgu}\{t_1 \stackrel{?}{=} s_1, \ldots, t_n \stackrel{?}{=} s_n\}$. Najbardziej ogólny unifikator nie zawsze istnieje, jeśli jednak zbiór unifikatorów danego równania jest niepusty, to istnieje wśród nich unifikator najbardziej ogólny. *Zadanie unifikacji* to problem znalezienia najbardziej ogólnego unifikatora dla zadanego równania (lub ogólniej układu równań). Algorytm znajdowania najbardziej ogólnego unifikatora pary termów:

$R \leftarrow \{t \overset{?}{=} s\}$, gdzie $t \overset{?}{=} s$ jest równaniem do rozwiązania
$\theta \leftarrow []$
dopóki $R \neq \varnothing$ wykonuj poniższe czynności:
  wybierz dowolne równanie $t \overset{?}{=} s$ z R i usuń je z R
  jeżeli $t \overset{?}{=} s$ jest postaci
    $x \overset{?}{=} x$ dla $x \in \mathcal{X}$, to pomiń je
    $x \overset{?}{=} t$ lub $t \overset{?}{=} x$, gdzie $x \in \mathcal{X}$ i $x \neq t$, to
      jeśli $x \notin \mathrm{FV}(t)$, to
        $R \leftarrow R[x/t]$
        $\theta \leftarrow \theta[x/t]$
      w przeciwnym razie koniec, unifikator nie istnieje
    $f(t_1, \ldots, t_n) \overset{?}{=} f(s_1, \ldots, s_n)$, to
      $R \leftarrow R \cup \{t_1 \overset{?}{=} s_1, \ldots, t_n \overset{?}{=} s_n\}$
    $f(t_1, \ldots, t_n) \overset{?}{=} g(s_1, \ldots, s_m)$, przy czym $f \neq g$, to
      koniec, unifikator nie istnieje
θ jest poszukiwanym najogólniejszym unifikatorem

Algorytm znajdowania najbardziej ogólnego unifikatora pary termów

Zmienne występujące w tym algorytmie, to układ równań R i podstawienie θ. Początkowo R zawiera wejściowe równanie do rozwiązania, $R = \{s \overset{?}{=} t\}$, zaś θ jest podstawieniem identycznościowym, $\theta = []$. W głównej pętli algorytmu są wykonywane pewne transformacje pary $\langle R, \theta\rangle$. Napis $R[x/t]$ oznacza wynik podstawienia $[x/t]$ we wszystkich termach układu równań R, natomiast $\theta[x/t]$ jest złożeniem podstawień θ i $[x/t]$. Dowód poprawności algorytmu polega na uzasadnieniu następującego faktu.

Fakt. *Niech $\langle R', \theta'\rangle$ będzie wynikiem wykonania dowolnej transformacji występującej w algorytmie unifikacji. Wówczas*

$$\{\theta\rho \mid \rho \text{ jest unifikatorem } R\} \quad = \quad \{\theta'\rho' \mid \rho' \text{ jest unifikatorem } R'\}$$

Niech $s \overset{?}{=} t$ będzie wejściowym równaniem, θ zaś wynikiem pracy algorytmu (za chwilę uzasadnimy, że algorytm zawsze się zatrzymuje). Z faktu 15.3 wynika natychmiast przez indukcję względem liczby kroków algorytmu, że

$$\{\rho \mid \rho \text{ jest unifikatorem równania } s \overset{?}{=} t\} \quad = \quad \{\theta\rho \mid \text{dla dowolnego } \rho\}$$

gdyż na końcu układ R jest pusty, każde podstawienie ρ jest więc jego unifikatorem. Zatem wszystkie unifikatory równania $s \overset{?}{=} t$ są postaci $\theta\rho$. Najogólniejszym z nich jest oczywiście θ. W przypadkach, w których algorytm twierdzi, że unifikator nie istnieje łatwo sprawdzić, że nie istnieje unifikator wyróżnionego w danym kroku równania, a więc i całego układu R. Na mocy faktu 15.3 nie istnieje zatem również unifikator wyjściowego równania. Aby dowieść, że algorytm się nie zapętla, definiujemy pewną „miarę złożoności" układu R. Jest to para nieujemnych liczb całkowitych $\langle n, m\rangle$, gdzie n jest liczbą zmiennych występujących w R, zaś m — liczbą wystąpień wszystkich zmiennych i symboli w R. Zbiór par nieujemnych liczb całkowitych jest dobrze uporządkowany relacją porządku leksykograficznego. Pozostaje sprawdzić, że wykonanie jakiejkolwiek transformacji układu $\langle R, \theta\rangle$ powoduje zmniejszenie naszej miary złożoności. Ponieważ w zbiorze dobrze uporządkowanym nie istnieją nieskończone ciągi ściśle malejące, po wykonaniu skończonej liczby kroków algorytm musi się zatrzymać.

Przykład. Niech sygnatura jednogatunkowa Σ zawiera binarny symbol f i unarny symbol g. Zmiennymi są X, Y, Z itd. Znajdziemy najogólniejszy unifikator równania

$$f(X, f(Y, g(Y))) \stackrel{?}{=} f(Z, Z)$$

Początkowo $R = \{f(X, f(Y, g(Y))) \stackrel{?}{=} f(Z, Z)\}$, wybieramy zatem wejściowe równanie. Pasuje do niego przedostatnia tranformacja opisana w algorytmie unifikacji. Tworzymy zatem nowy układ $R = \{X \stackrel{?}{=} Z, f(Y, g(Y)) \stackrel{?}{=} Z\}$, a podstawienie θ nadal jest identycznościowe. Teraz możemy wybrać jedno z dwóch równań. Rozważmy np. pierwsze. Jest ono postaci $x \stackrel{?}{=} t$, gdzie $x \notin \mathrm{FV}(t)$. Zatem w drugim równaniu układu R dokonujemy podstawienia $[X/Z]$ i podstawienie to składamy z podstawieniem θ. Otrzymujemy $R = \{f(Y, g(Y)) \stackrel{?}{=} Z\}$ i $\theta = [X/Z]$. W kolejnym kroku algorytmu rozważamy równanie $f(Y, g(Y)) \stackrel{?}{=} Z$. Podobnie jak w poprzednim, jest ono postaci $x \stackrel{?}{=} t$, gdzie $x \notin \mathrm{FV}(t)$. Mamy więc $R = \varnothing$ i $\theta = [X/Z][Z/f(Y, g(Y))] = [X/f(Y, g(Y)), Z/f(Y, g(Y))]$, gdzie ostatnia równość jest prawdziwa na mocy lematu o podstawianiu. Najogólniejszym unifikatorem równania $f(X, f(Y, g(Y))) \stackrel{?}{=} f(Z, Z)$ jest zatem

$$[X/f(Y, g(Y)), Z/f(Y, g(Y))]$$

Zauważmy, że proces rozwiązywania układu równań R bardzo przypomina metodę rozwiązywania układów równań liniowych zwaną *eliminacją Gaussa.* *Eliminacji struktury* (transformacji równania $f(t_1, \ldots, t_n) \stackrel{?}{=} f(s_1, \ldots, s_n)$ do układu $\{t_1 \stackrel{?}{=} s_1, \ldots, t_n \stackrel{?}{=} s_n\}$) odpowiada *normalizacja* równania liniowego, *eliminacji zmiennej* (usunięciu równania $x \stackrel{?}{=} t$ z układu) — podobna eliminacja w algorytmie Gaussa. Tam również w miejsce eliminowanej zmiennej wstawia się jej wyliczoną wartość. W algorytmie Gaussa otrzymujemy ostatecznie układ w tzw. *postaci rozwikłanej.* W algorytmie unifikacji rolę takiego układu pełni podstawienie.

16 Wykład 29.04.2008

16.1 Teorie syntaktyczne (TWi)

Termy są narzędziem pozwalającym nazywać pewne obiekty. Dotychczas żąglowaliśmy tymi nazwami nie próbując nadać im żadnego znaczenia. Sygnatura, określająca postać termów, mówi jedynie, co jest poprawną wypowiedzią, a co nie. Gra rolę słownika ortograficznego. Trudno byłoby zrozumieć sens zdania w nieznanym nam języku posługując się przy tym słownikiem ortograficznym! Słowniki wyjaśniające znaczenie słów podają zwykle równoważne opisy tej samej rzeczy na zasadzie „*to* jest to to samo co *tamto*". Możemy podobnie postąpić z termami, definiując pojęcie *równości.* Jest to para termów tego samego gatunku, zawierających przeważnie zmienne, zapisana w postaci $t = s$, np. $1 + 2 = 3$, $(x + y) \times z = x \times z + y \times z$ lub prawdziwy(przyjaciel) = Cudak. Wybrane równości przyjmujemy za spełnione *ad hoc* i nazywamy *aksjomatami równościowymi.* Wyliczając aksjomaty pragniemy podać zasady utożsamiania termów, uważania ich za równoważne. Definiujemy zatem pewną relację równości termów. Relacja równości powinna być relacją równoważności: być zwrotna, przechodnia i symetryczna. Nadto powinna być *monotoniczna*: jeżeli uznaliśmy termy t i s (być może zawierające zmienną x) za równe, to cokolwiek podstawilibyśmy za zmienną x w obu termach jednocześnie nie powinno tej relacji równości zaburzyć. Np. jeśli $x + y = y + x$, to także $(z + w) + y = y + (z + w)$, $1 + 2 = 2 + 1$ itd. Za zmienną x w obu termach nie potrzeba nawet wstawiać tego samego termu. Wystarczy, że wstawimy tam termy, o których wiemy, że są równe. Zatem będziemy mówić, że binarna, określona na termach relacja R jest *monotoniczna*, jeżeli

$$tRs \wedge rRu \implies (t[x/r])R(s[x/u])$$

$$\frac{}{t = t}\ (\text{Refl}) \qquad \frac{t = s}{s = t}\ (\text{Sym}) \qquad \frac{s = r \quad r = t}{s = t}\ (\text{Trans}) \qquad \frac{t = s \quad r = u}{t[x/r] = s[x/u]}\ (\text{Mon})$$

Reguły wnioskowania dla równościowych teorii syntaktycznych

Dla uproszczenia definicji rozważmy zbiór termów $\mathcal{T}(\Sigma, \mathcal{X})$ nad jednogatunkową sygnaturą Σ i zbiorem zmiennych $\mathcal{X}$ (przypadek sygnatury wielogatunkowej można rozważyć analogicznie, definicje się jednak nieco komplikują). *Równościową teorią syntaktyczną* zadaną przez zbiór aksjomatów A i oznaczaną $\mathrm{Th}^{\vdash}(A)$ nazywamy najmniejszą monotoniczną relację równoważności zawierającą zbiór A. Niekiedy będziemy pisać $A \vdash t = s$ na oznaczenie faktu, że $(t = s) \in \mathrm{Th}^{\vdash}(A)$ i mówić, że równość $t = s$ jest *twierdzeniem teorii* $\mathrm{Th}^{\vdash}(A)$. Poprawność definicji wymaga dowodu (elementy najmniejsze nie zawsze istnieją). Niech $\{R_\kappa\}_\kappa$ będzie rodziną wszystkich monotonicznych relacji równoważności zawierających zbiór A. Rodzina ta jest niepusta, bo należy do niej relacja totalna (zawierająca wszystkie pary termów). Ponieważ przekrój dowolnej liczby monotonicznych relacji równoważności jest monotoniczną relacją równoważności, to $\bigcap_\kappa R_\kappa$ jest najmniejszą monotoniczną relacją równoważności zawierającą zbiór A. Definicja jest zatem poprawna.

Relację $\mathrm{Th}^{\vdash}(A)$ można też budować indukcyjnie. Niech

$$\begin{aligned}
R_0 &= A \cup \{t = t \mid t \in \mathcal{T}(\Sigma, \mathcal{X})\} \\
R_{n+1} &= R_n \cup \{s = t \mid (t = s) \in R_n\} \cup \{s = t \mid (s = u), (u = t) \in R_n\} \cup \\
&\quad\ \{t[x/r] = s[x/u] \mid (t = s), (r = u) \in R_n\} \\
R &= \bigcup_{n=0}^{\infty} R_n
\end{aligned}$$

Wówczas $R = \mathrm{Th}^{\vdash}(A)$. Wpierw pokazujemy indukcyjnie względem n, że $R_n \subseteq \mathrm{Th}^{\vdash}(A)$ dla każdego $n \geq 0$, więc $R \subseteq \mathrm{Th}^{\vdash}(A)$. Następnie zauważamy, że R jest monotoniczną relacją równoważności zawierającą A, mamy więc $\mathrm{Th}^{\vdash}(A) \subseteq R$. Aby sprawdzić, czy równość $t = s$ należy do teorii $\mathrm{Th}^{\vdash}(A)$ możemy zacząć od aksjomatów ze zbioru A i równości $t = t$ dla $t \in \mathcal{T}(\Sigma, \mathcal{X})$ i stosować następujące reguły wnioskowania:

1. jeżeli $(t = s) \in \mathrm{Th}^{\vdash}(A)$, to także $(s = t) \in \mathrm{Th}^{\vdash}(A)$;
2. jeżeli $(s = r) \in \mathrm{Th}^{\vdash}(A)$ i $(r = t) \in \mathrm{Th}^{\vdash}(A)$, to także $(s = t) \in \mathrm{Th}^{\vdash}(A)$;
3. jeżeli $(t = s) \in \mathrm{Th}^{\vdash}(A)$ i $(r = u) \in \mathrm{Th}^{\vdash}(A)$, to także $(t[x/r] = s[x/u]) \in \mathrm{Th}^{\vdash}(A)$;

tak długo, aż dojdziemy do równości, która nas interesuje. Powyższy mechanizm można wygodnie opisać w postaci tzw. *formalnego systemu wnioskowania*, składającego się ze zbioru aksjomatów A i zestawu czterech *reguł wnioskowania*. Równość $t = s$ jest twierdzeniem teorii $\mathrm{Th}^{\vdash}(A)$, jeżeli istnieje *drzewo dowodu*, w którego korzeniu znajduje się równość $t = s$ a w liściach — aksjomaty ze zbioru A lub równość $t = t$, tj. aksjomat (Refl).

16.2 Algebry (TWi)

Niech $\Sigma = \{\Sigma^a\}_{a \in \mathcal{S}}$ będzie sygnaturą nad zbiorem gatunków $\mathcal{S}$. *Algebrą* (*strukturą algebraiczną*) o sygnaturze Σ nazywamy parę $\mathfrak{A} = \langle \mathcal{A}, \cdot^{\mathfrak{A}} \rangle$ złożoną z rodziny $\mathcal{A} = \{A^a\}_{a \in \mathcal{S}}$ zbiorów zwanych *dziedzinami* (*nośnikami*, *uniwersami*) algebry (po jednym dla każdego gatunku) i odwzorowania $\cdot^{\mathfrak{A}}$ zwanego *interpretacją symboli funkcyjnych*, które każdemu symbolowi sygnatury $f \in \Sigma^{a_1 \times \ldots \times a_n \to b}$ przyporządkowuje funkcję

$$f^{\mathfrak{A}} : A^{a_1} \times \ldots \times A^{a_n} \to A^b$$

Przykład. Rozważmy sygnaturę jednogatunkową $\Sigma = \{\Sigma_n\}_{n=0}^{\infty}$, w której $\Sigma_0 = \{e\}$, $\Sigma_2 = \{\otimes\}$ i pozostałe zbiory symboli są puste. Przykładami algebr o sygnaturze Σ są

1. addytywna półgrupa liczb naturalnych $\mathfrak{N}^+ = \langle \mathbb{N}, \cdot^{\mathfrak{N}} \rangle$, w której $\mathbb{N}$ jest zbiorem liczb naturalnych, $e^{\mathfrak{N}^+} = 0$, a $\otimes^{\mathfrak{N}^+}$ jest operacją dodawania;
2. multiplikatywna półgrupa liczb naturalnych $\mathfrak{N}^\times = \langle \mathbb{N}, \cdot^{\mathfrak{N}^\times} \rangle$, w której $\mathbb{N}$ jest zbiorem liczb naturalnych, $e^{\mathfrak{N}^\times} = 1$, a $\otimes^{\mathfrak{N}^\times}$ jest operacją mnożenia;
3. półgrupa słów $\mathfrak{W} = \langle \Xi^*, \cdot^{\mathfrak{W}} \rangle$, w której Ξ^* jest zbiorem słów nad pewnym alfabetem Ξ, $e^{\mathfrak{W}}$ jest słowem pustym, a $\otimes^{\mathfrak{W}}$ jest operacją konkatenacji słów.

Przykład. Rozważmy zbiór gatunków $\mathcal{S} = \{B, N\}$ i sygnaturę, w której $\Sigma^B = \{\mathtt{T}, \mathtt{F}\}$, $\Sigma^N = \{\mathtt{0}, \mathtt{1}, \ldots\}$, $\Sigma^{N\times N\to N} = \{\mathtt{+}, \mathtt{*}\}$, $\Sigma^{B\times B\to B} = \{\mathtt{|}, \mathtt{\&}\}$, $\Sigma^{B\times N\times N\to N} = \{\mathtt{?}\}$, a pozostałe zbiory symboli są puste. Przykładem algebry o tej sygnaturze jest $\mathfrak{A} = \langle \{A^B, A^N\}, \cdot^{\mathfrak{A}} \rangle$, w której $A^B = \{T, F\}$, $A^N = \mathbb{N}$ (zbiór liczb naturalnych), $\mathtt{T}^{\mathfrak{A}} = T$, $\mathtt{F}^{\mathfrak{A}} = F$, $\mathtt{0}^{\mathfrak{A}} = 0$, $\mathtt{1}^{\mathfrak{A}} = 1$ itd, $\mathtt{+}^{\mathfrak{A}}$ jest operacją dodawania liczb naturalnych, $\mathtt{*}^{\mathfrak{A}}$ jest operacją mnożenia liczb naturalnych, $\mathtt{|}^{\mathfrak{A}}$ jest alternatywą a $\mathtt{\&}^{\mathfrak{A}}$ koniunkcją wartości logicznych, zaś $\mathtt{?}^{\mathfrak{A}}(T, n, m) = n$ i $\mathtt{?}^{\mathfrak{A}}(F, n, m) = m$ dla dowolnych liczb naturalnych $n, m \in \mathbb{N}$.

Przykład. Niech Σ będzie sygnaturą nad zbiorem gatunków $\mathcal{S}$ a $\mathcal{X}$ rodziną zmiennych oraz

$$f^{\mathfrak{T}}(t_1, \ldots, t_n) \quad = \quad f(t_1, \ldots, t_n)$$

dla $f \in \Sigma^{a_1\times\ldots\times a_n\to b}$, $t_i \in \mathcal{T}^{a_i}(\Sigma, \mathcal{X})$ dla $i = 1, \ldots, n$ i $a_1, \ldots, a_n, b \in \mathcal{S}$. Wówczas $\mathfrak{T}(\Sigma, \mathcal{X}) = \langle \{T^a(\Sigma, \mathcal{X})\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{T}} \rangle$ jest algebrą o sygnaturze Σ. Nazywamy ją *algebrą termów*.

Rozważmy algebrę $\mathfrak{A} = \langle \{A^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{A}} \rangle$ o sygnaturze Σ nad zbiorem gatunków $\mathcal{S}$. Niech $\mathcal{X} = \{\mathcal{X}^a\}_{a\in\mathcal{S}}$ będzie rodziną zmiennych. Dowolną rodzinę odwzorowań $\eta = \{\eta^a\}_{a\in\mathcal{S}}$, takich że $\eta^a : \mathcal{X}^a \to A^a$ nazywamy *interpretacją zmiennych* w algebrze $\mathfrak{A}$. Interpretacja symboli funkcyjnych wraz z interpretacją zmiennych jednoznacznie zadają interpretację $[\![\cdot]\!]_\eta^{\mathfrak{A}}$ dowolnych termów w algebrze $\mathfrak{A}$:

$$\begin{aligned} [\![x^a]\!]_\eta^{\mathfrak{A}} \quad &= \quad \eta^a(x^a) \\ [\![f(t_1, \ldots, t_n)]\!]_\eta^{\mathfrak{A}} \quad &= \quad f^{\mathfrak{A}}([\![t_1]\!]_\eta^{\mathfrak{A}}, \ldots, [\![t_n]\!]_\eta^{\mathfrak{A}}) \end{aligned}$$

dla $x^a \in \mathcal{X}^a$, $f \in \Sigma^{a_1\times\ldots\times a_n\to b}$ i $t_i \in \mathcal{T}^{a_i}(\Sigma, \mathcal{X})$, $i = 1, \ldots, n$ oraz $a, a_1, \ldots, a_n, b \in \mathcal{S}$.

Fakt. *Jeżeli wartościowania zmiennych η_1 i η_2 zgadzają się na zbiorze zmiennych termu t, tj. $\eta_1^a(x^a) = \eta_2^a(x^a)$ dla $x^a \in \mathrm{FV}(t)$, to $[\![t]\!]_{\eta_1}^{\mathfrak{A}} = [\![t]\!]_{\eta_2}^{\mathfrak{A}}$. W szczególności wartościowanie termu stałego nie zależy od wartościowania zmiennych. Będziemy więc pisać $[\![t]\!]^{\mathfrak{A}}$, pomijając wartościowanie zmiennych, gdy $\mathrm{FV}(t) = \varnothing$.*

16.3 Homomorfizmy algebr (TWi)

Rozważmy dwie algebry $\mathfrak{A} = \langle \{A^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{A}} \rangle$ i $\mathfrak{B} = \langle \{B^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{B}} \rangle$ o tej samej sygnaturze Σ. Rodzinę odwzorowań $h = \{h^a\}_{a\in\mathcal{S}}$, gdzie $h^a : A^a \to B^a$ dla $a \in \mathcal{S}$, nazywamy *homomorfizmem*, jeżeli

$$h^b(f^{\mathfrak{A}}(u_1, \ldots, u_n)) \quad = \quad f^{\mathfrak{B}}(h^{a_1}(u_1), \ldots, h^{a_n}(u_n))$$

dla $f \in \Sigma^{a_1\times\ldots\times a_n\to b}$, $u_i \in A_i$ dla $i = 1, \ldots, n$ oraz $a_1, \ldots, a_n, b \in \mathcal{S}$. Homomorfizm będący bijekcją (odwzorowaniem różnowartościowym i „na") nazywamy *izomorfizmem*. Jeżeli istnieje izomorfizm z algebry $\mathfrak{A}$ na algebrę $\mathfrak{B}$, to mówimy, że algebry $\mathfrak{A}$ i $\mathfrak{B}$ są *izomorficzne*. Izomorfizm algebr jest relacją równoważności.

Przykład. Rozważmy algebry $\mathfrak{R}^+ = \langle \mathbb{R}, \cdot^{\mathfrak{R}^+} \rangle$ i $\mathfrak{R}^\times = \langle \mathbb{R}^+, \cdot^{\mathfrak{R}^\times} \rangle$ nad jednogatunkową sygnaturą zawierającą stałą e i binarny symbol $\otimes$, gdzie $\mathbb{R}$ jest zbiorem liczb rzeczywistych a $\mathbb{R}^+$ zbiorem liczb rzeczywistych dodatnich, $\otimes^{\mathfrak{R}^+}$ jest dodawaniem a $\otimes^{\mathfrak{R}^\times}$ mnożeniem liczb rzeczywistych, $e^{\mathfrak{R}^+} = 0$ i $e^{\mathfrak{R}^\times} = 1$. Wówczas funkcja logarytmiczna $\ln : \mathbb{R}^+ \to \mathbb{R}$ jest homomorfizmem z algebry $\mathfrak{R}^\times$ w algebrę $\mathfrak{R}^+$, bo

$$\ln\left(e^{\mathfrak{R}^\times}\right) = \ln\left(1\right) = 0 = e^{\mathfrak{R}^+}$$

$$\ln\left(u_1 \otimes^{\mathfrak{R}^\times} u_2\right) = \ln\left(u_1 \times u_2\right) = \ln\left(u_1\right) + \ln\left(u_2\right) = \ln\left(u_1\right) \otimes^{\mathfrak{R}^+} \ln\left(u_2\right)$$

dla dowolnych dodatnich liczb rzeczywistych $u_1, u_2 \in \mathbb{R}^+$. Ponieważ jest ona także bijekcją, to jest izomorfizmem algebr $\mathfrak{R}^\times$ i $\mathfrak{R}^+$.

Przykład. W każdej algebrze $\mathfrak{A}$ wartościowanie termów dla dowolnego wartościowania zmiennych jest homomorfizmem z algebry termów w algebrę $\mathfrak{A}$.

Przykład. Podstawienie jest homomorfizmem z algebry termów w nią samą.

16.4 Podalgebry i algebry generowane (TWi)

Podalgebrą algebry $\mathfrak{A} = \langle \{A^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{A}} \rangle$ o sygnaturze Σ nad zbiorem gatunków $\mathcal{S}$ nazywamy rodzinę zbiorów $\{B^a\}_{a\in\mathcal{S}}$, taką, że:

1. $B^a \subseteq A^a$ dla $a \in \mathcal{S}$;
2. $f^{\mathfrak{A}}(u_1, \ldots, u_n) \in B^b$ dla $f \in \Sigma^{a_1\times\ldots\times a_n \to b}$, $u_i \in B^{a_i}$, $a_1, \ldots, a_n, b \in \mathcal{S}$ (zbiory B^a są zamknięte ze względu na działania $f^{\mathfrak{A}}$).

Podalgebra $\{B^a\}_{a\in\mathcal{S}}$ jest algebrą $\mathfrak{B} = \langle \{B^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{B}} \rangle$ o sygnaturze Σ, jeśli położyć $f^{\mathfrak{B}} = f^{\mathfrak{A}}\restriction_{B^{a_1}\times\ldots\times B^{a_n}}$ dla $f \in \Pi^{a_1\times\ldots\times a_n \to b}$ i $a_1, \ldots, a_n, b \in \mathcal{S}$ (działania $f^{\mathfrak{B}}$ są obcięciami działań $f^{\mathfrak{A}}$ do dziedzin algebry $\mathfrak{B}$).

Niech $\mathcal{G} = \{G^a\}_{a\in\mathcal{S}}$ będzie rodziną zbiorów, taką, że $G^a \subseteq A^a$. *Algebrą generowaną* przez rodzinę $\mathcal{G}$ nazywamy najmniejszą (w sensie relacji inkluzji dziedzin) podalgebrę $\mathfrak{A}(\mathcal{G}) = \langle \{B^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{A}(\mathcal{G})} \rangle$ algebry $\mathfrak{A}$, taką, że $G^a \subseteq B^a$ dla każdego $a \in \mathcal{S}$. Uzasadnienia wymaga poprawność powyższej definicji (postulujemy, by odpowiednie zbiory były najmniejsze w pewnej klasie, podczas gdy elementy najmniejsze nie zawsze istnieją). Niech zatem $\{\mathfrak{B}_\kappa\}_\kappa$, gdzie $\mathfrak{B}_\kappa = \langle \{B^a_\kappa\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{B}_\kappa} \rangle$, będzie rodziną wszystkich podalgebr algebry $\mathfrak{A}$, których dziedziny zawierają zbiory G^a. Rodzina ta jest niepusta, bo należy do niej sama algebra $\mathfrak{A}$. Niech $B^a = \bigcap_\kappa B^a_\kappa$ dla $a \in \mathcal{S}$. Rodzina $\{B^a\}_{a\in\mathcal{S}}$ jest, jak łatwo sprawdzić, podalgebrą algebry $\mathfrak{A}$. Nadto jest ona najmniejszą podalgebrą, której dziedziny zawierają zbiory G^a.

16.5 Zasada indukcji (TWi)

Twierdzenie. (Zasada indukcji strukturalnej) Rozważmy rodzinę predykatów $\Phi = \{\Phi^a\}_{a\in\mathcal{S}}$ określonych na dziedzinach algebry $\mathfrak{A} = \langle \{A^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{A}} \rangle$ o sygnaturze Σ, tj. niech $\Phi^a \subseteq A^a$ dla $a \in \mathcal{S}$. Niech $\{G^a\}_{a\in\mathcal{S}}$ będzie rodziną zbiorów, taką, że $G^a \subseteq A^a$ dla $a \in \mathcal{S}$ i niech $\mathfrak{A}(\mathcal{G}) = \langle \{B^a\}_{a\in\mathcal{S}}, \cdot^{\mathfrak{A}(\mathcal{G})} \rangle$ będzie algebrą generowaną przez rodzinę $\{G^a\}_{a\in\mathcal{S}}$. Jeżeli

1. $G^a \subseteq \Phi^a$ dla $a \in \mathcal{S}$;
2. $\Phi^{a_1}(u_1) \wedge \ldots \wedge \Phi^{a_n}(u_n) \Longrightarrow \Phi^b(f^{\mathfrak{A}}(u_1, \ldots, u_n))$ dla $f \in \Pi^{a_1\times\ldots\times a_n \to b}$, $u_i \in B^{a_i}$ i $a_1, \ldots, a_n, b \in \mathcal{S}$,

to $\Phi^a(u)$ dla każdego $u \in B^a$ i $a \in \mathcal{S}$.

Dowód. Z warunku 2 rodzina $\{\Phi^a\}_{a\in\mathcal{S}}$ jest podalgebrą algebry $\mathfrak{A}$. Z warunku 1 jest więc podalgebrą algebry $\mathfrak{A}$, której dziedziny zawierają zbiory G^a. Ponieważ algebra generowana jest najmniejszą algebrą o powyższych własnościach, więc $B^a \subseteq \Phi^a$ dla $a \in \mathcal{S}$. $\square$

Niech Σ będzie sygnaturą nad zbiorem gatunków $\mathcal{S}$, a $\mathfrak{T}$ algebrą termów stałych o sygnaturze Σ. Wtedy $\mathfrak{T}(\varnothing) = \mathfrak{T}$. Mamy więc w szczególności:

Twierdzenie. (Zasada indukcji strukturalnej dla termów) Niech Σ będzie sygnaturą nad zbiorem gatunków $\mathcal{S}$. Rozważmy rodzinę $\Phi = \{\Phi^a\}_{a\in\mathcal{S}}$ predykatów określonych na zbiorach termów $\{\mathcal{T}^a(\Sigma,\varnothing)\}_{a\in\mathcal{S}}$ nad sygnaturą Σ, tj. niech $\Phi^a \subseteq \mathcal{T}^a(\Sigma,\varnothing)$ dla $a\in\mathcal{S}$. Jeżeli

$$\Phi^{a_1}(t_1)\wedge\ldots\wedge\Phi^{a_n}(t_n) \quad\Longrightarrow\quad \Phi^b(f(t_1,\ldots,t_n))$$

dla $f\in\Sigma^{a_1\times\ldots\times a_n\to b}$, $t_i\in\mathcal{T}^{a_i}(\Sigma,\varnothing)$ i $a_1,\ldots,a_n,b\in\mathcal{S}$, to $\Phi^a(t)$ dla $t\in\mathcal{T}^a(\Sigma,\varnothing)$ i $a\in\mathcal{S}$.

16.6 Konstruktory (TWi)

Niech $\mathfrak{A} = \langle\{A^a\}_{a\in\mathcal{S}},\cdot^{\mathfrak{A}}\rangle$ będzie algebrą o sygnaturze Σ i niech Γ będzie sygnaturą zawierającą wybrane symbole sygnatury Σ (tj. $\Gamma^\tau\subseteq\Sigma^\tau$ dla $\tau\in\mathbb{T}_1(\mathcal{S})$). Niech $\mathfrak{A}\restriction_\Gamma = \langle\{A^a\}_{a\in\mathcal{S}},\cdot^{\mathfrak{A}}\restriction_\Gamma\rangle$ będzie obcięciem algebry $\mathfrak{A}$ do sygnatury Γ. Jeżeli wartościowanie termów $[\![\cdot]\!]$ w algebrze $\mathfrak{A}$ jest izomorfizmem z algebry termów stałych $\mathfrak{T} = \langle\{T^a(\Gamma,\varnothing)\}_{a\in\mathcal{S}},\cdot^{\mathfrak{T}}\rangle$ o sygnaturze Γ na algebrę $\mathfrak{A}\restriction_\Gamma$, to sygnaturę Γ nazywamy *zbiorem konstruktorów* algebry $\mathfrak{A}$. Jeżeli Γ jest zbiorem konstruktorów algebry $\mathfrak{A} = \langle\{A^a\}_{a\in\mathcal{S}},\cdot^{\mathfrak{A}}\rangle$, to $\mathfrak{T} = \langle\{T^a(\Gamma,\varnothing)\}_{a\in\mathcal{S}},\cdot^{\mathfrak{T}}\rangle$ można rozważać jako algebrę o sygnaturze Σ, kładąc $f^{\mathfrak{T}}$ takie, by

$$[\![f^{\mathfrak{T}}(t_1,\ldots,t_n)]\!] \quad=\quad f^{\mathfrak{A}}([\![t_1]\!],\ldots,[\![t_n]\!])$$

dla f nie należących do Γ. Wówczas algebra termów $\mathfrak{T}$ jest izomorficzna z algebrą $\mathfrak{A}$.

Jeżeli Γ jest zbiorem konstruktorów algebry $\mathfrak{A}$, to każdy element algebry $\mathfrak{A}$ ma jednoznaczną nazwę w postaci termu nad sygnaturą Γ. Możemy także dowodzić własności algebry $\mathfrak{A}$ przez indukcję strukturalną względem zbioru konstruktorów Γ.

Nie każda algebra posiada zbiór konstruktorów. Niektóre zaś posiadają wiele zbiorów konstruktorów. Pojęcie zbioru konstruktorów ma wielkie znaczenie w teorii specyfikacji algebraicznych.

16.7 Teorie semantyczne (TWi)

Rozważmy algebrę $\mathfrak{A} = \langle\{A^a\}_{a\in\mathcal{S}},\cdot^{\mathfrak{A}}\rangle$ nad sygnaturą Σ. Powiemy, że równość $t = s$ jest *spełniona* w algebrze $\mathfrak{A}$, co oznaczamy $\mathfrak{A}\vDash t = s$, jeżeli $[\![t]\!]^{\mathfrak{A}}_\eta = [\![s]\!]^{\mathfrak{A}}_\eta$ dla każdego wartościowania zmiennych η. Jeżeli E jest zbiorem równości, to mówimy, że jest on *spełniony* w algebrze $\mathfrak{A}$, co oznaczamy $\mathfrak{A}\vDash E$, jeżeli $\mathfrak{A}\vDash t = s$ dla każdej równości $(t = s)\in E$. Zbiór równości spełnionych w algebrze $\mathfrak{A}$ oznaczamy $\mathrm{Th}(\mathfrak{A}) = \{t = s \mid \mathfrak{A}\vDash t = s\}$ i nazywamy *teorią algebry* $\mathfrak{A}$. Klasą algebr *zdefiniowaną* przez zbiór równości (aksjomatów) E nazywamy klasę $\mathcal{E}(E) = \{\mathfrak{A}\mid\mathfrak{A}\vDash E\}$. *Równościową teorią semantyczną* zadaną przez zbiór równości E nazywamy zbiór równości spełnionych w każdej algebrze spełniającej E:

$$\mathrm{Th}^{\vDash}(E) = \{(t = s)\mid\forall\mathfrak{A}.(\mathfrak{A}\vDash E\Rightarrow\mathfrak{A}\vDash t = s)\} = \bigcap_{\mathfrak{A}:\mathfrak{A}\vDash E}\mathrm{Th}(\mathfrak{A})$$

Twierdzenie. Dla każdego zbioru równości E jest $\mathrm{Th}^{\vdash}(E) = \mathrm{Th}^{\vDash}(E)$.

Możemy więc mówić po prostu o teorii równościowej $\mathrm{Th}(E)$ nie zaznaczając, czy jest ona syntaktyczna, czy semantyczna. Pojęcia te są bowiem zbieżne.

Można zadać pytanie, czy istnieje jedna konkretna algebra $\mathfrak{A}$, taka, że jej teoria jest teorią zadaną przez ustalony zbiór równości E, tj. czy istnieje algebra $\mathfrak{A}$, taka, że $\mathrm{Th}(\mathfrak{A}) = \mathrm{Th}(E)$. Nie zawsze tak jest. Jeśli jednak każda algebra spełniająca E ma wszystkie nośniki niepuste, to taką algebrą jest $\mathfrak{A} = \langle \mathcal{T}(\Sigma, \mathcal{X})/_\sim, \cdot^{\mathfrak{A}} \rangle$ gdzie $t \sim s \iff (t = s) \in \mathrm{Th}(E)$ i $f^{\mathfrak{A}}([t_1]_\sim, \ldots, [t_n]_\sim) = [f(t_1, \ldots, t_n)]_\sim$.

Niech $\mathcal{A}$ będzie klasą algebr o sygnaturze Σ. Algebra $\mathfrak{A} \in \mathcal{A}$ jest *algebrą początkową w klasie* $\mathcal{A}$, jeżeli dla każdej algebry $\mathfrak{B} \in \mathcal{A}$ istnieje dokładnie jeden homomorfizm z algebry $\mathfrak{A}$ w algebrę $\mathfrak{B}$. *Algebrą początkową dla zbioru równości E* nazywamy algebrę początkową w klasie $\mathcal{E}(E)$.

Twierdzenie. Algebrą początkową dla zbioru równości E jest algebra

$$\mathfrak{P} = \langle \mathcal{T}(\Sigma, \varnothing)/_\sim, \cdot^{\mathfrak{P}} \rangle$$

gdzie $t \sim s \iff (t = s) \in \mathrm{Th}(E)$ i $f^{\mathfrak{P}}([t_1]_\sim, \ldots, [t_n]_\sim) = [f(t_1, \ldots, t_n)]_\sim$.

W algebrze początkowej są spełnione te równości między termami *stałymi*, które są dowodliwe ze zbioru równości E. W algebrze początkowej mogą być jest spełnione inne równości (pomiędzy termami zawierającymi zmienne), które nie są dowodliwe ze zbioru równości E.

17 Wykład 06.05.2008

17.1 Mini-Haskell

$$\tau(\Sigma, \mathcal{X}) \quad \Sigma = \{0, 1, :, Nil, ++\}$$

```
data Nat where
    0 :: Nat
    1 :: Nat

data List where
    Nil :: List
    (:) :: Nat -> List -> List

(++) :: List -> List -> List
Nil ++ ys = ys                    \
(x:xs) ++ ys = x : (xs ++ ys)     /  <-   to jest to ,,E'' (niżej)
```

Definicja. Semantyka Operacyjna - najmniejsza relacja monotoniczna będąca praporządkiem i zawierająca następujące równości:

- $\dfrac{}{t \to t}$,
- $\dfrac{t \to s \quad s \to r}{t \to r}$,
- $\dfrac{t \to s \quad u \to v}{u[x/t] \to s[x/v]}$,
- $\dfrac{(t = s) \subset E}{t \to s}$,

redex = reducible expression

Definicja. Wyrażenie w postaci normalnej, to takie wyrażenie, które nie zawiera redexów.

Definicja. System jest konfluentny jeżeli wartość wyrażenia nie zależy od kolejności redukowania redexów.

Definicja. System (albo pojedynczy term) jest normalizowalny, jeśli posiada postać normalną. Powiemy, że jest mocno normalizowalny, jeżeli każda kolejność redukcji prowadzi do postaci normalnej.

Definicja. Jeżeli zawsze zamykamy najbardziej zewnętrzny redex położony najbardziej na lewo, taka strategia to strategia normalna (leniwa).

Definicja. Jeżeli zawsze zamykamy najbardziej wewnętrzny redex położony najbardziej na lewo to taka strategia = strategia aplikatywna (gorliwa).

17.2 ...

$\Sigma = \{0, 1, :, Nil, ++\}$ – taką sobie mamy algebrę dwugatunkową. Jej uniwersa to:

- $U_{Nat} = \{0, 1\} \quad \cancel{\cup\{\perp\}}$,
- $U_{List} = U_{Nat}^* \quad \cancel{\cup\{\perp\}}$,

i tak otrzymaliśmy semantykę denotacyjną.

17.3 Semantyka algebraiczna

$$E \vdash t = s \qquad E \text{ – skończony zbiór równości.}$$

Twierdzenie. Dla każdej algebry $\mathcal{A}$, w której $A \vDash E$ jest $A \vDash t = s$.

$$E \vdash t = s \Leftrightarrow E \vDash t = s$$

Chcemy mieć jedną, konkretną algerbę $\mathcal{A}$ (i chcemy, żeby ona była izomorficzna z semantyką denotacyjną!).

- **no junk** – algebra ma być osiągalna,
- **no confusion** – utożsamia tylko te elementy, które musi

Definicja. Algebra początkowa: Taka algebra A, że istnieje dla każdej innej algebry dokładnie jeden homomorfizm przekształacający tą algebrę w inną algebrę.

18 Wykład 08.05.2008

18.1 Smutek i nostalgia

Opuszczamy fajny świat semantyki algebraicznej, a szkoda... Bo to bardzo prościutka teoria była.

```
{-# RULE "assoc ++"
    forall xs ys zs . (xs ++ ys) ++ zs = xs ++ (ys ++ zs)
#-}
```

18.2 Rachunek Lambda

Alonzo Church

$$\Lambda \quad = \quad x \quad | \quad \underbrace{\Lambda\Lambda}_{aplikacja} \quad | \quad \underbrace{\lambda x.\Lambda}_{abstrakcja} \quad | \quad c$$

gdzie c to sygnatura (a „czyste rachunki lambda” jej nie zawierają). Powyżej podaliśmy lambda-notację.

- $\forall x.\varphi \equiv A(\lambda x.\varphi)$
- $ae_1e_2 \equiv e_1 \wedge e_2$
- $n\ e \equiv \neg e$

Paradoksalny kombinator punktu stałego:

- $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$
- $Ye = e(Ye)$

Chcemy wprowadzić pewną relację równości, która będzie mówiła, kiedy dwa wyrażenia są równe.

$$e_1 = e_2$$

Przyjmujemy sobie, że aplikacja wiąże w lewo

- $e_1e_2...e_n \equiv ((e_1e_2)...e_n)$
- $\lambda x.e_1...e_n \equiv \lambda x.(e_1...e_n)$
- $\lambda x_1...x_n.e \equiv \lambda x_1.(\lambda x_2....(\lambda x.e)...)$

Definicja. Dowody indukcyjne: niech S będzie zbiorem Λ-termów takim, że

- $x \in S$ dla każdej zmiennej,
- jeżeli $r, s \in S$, to $r\ s \in S$,
- jeżeli $r \in S$, to $\lambda x.r \in S$ dla każdej zmiennej x

to S zawiera wszystkie termy.

Jednoznaczność:
Jeżeli t jest λ-wyrażeniem, to

- albo jest zmienną,
- albo jest aplikacją jakiegoś termu r_1 do jakiegoś termu r_2,
- albo jest abstrakcją w postaci $\lambda x.r_1$,

Definicja. Funkcje definiowane przez rekursję strukturalną:

- $F(x) = F_x(x)$,
- $F(r\ s) = F_{app}(F(r), F(s))$,

- $F(\lambda x.r) = F_{abs}(x, F(r))$,

Definicja. Zbiór zmiennych wolnych

- $FV(x) = \{x\}$
- $FV(r\ s) = FV(r) \cup FV(s)$,
- $FV(\lambda x.r) = FV(r) \backslash \{x\}$,

Definicja. Zbiór zmiennych związanych

- $BV(x) = \varnothing$
- $BV(r\ s) = BV(r) \cup BV(s)$,
- $BV(\lambda x.r) = BV(r) \cup \{x\}$,

Definicja. Zbiór podtermów

- $sub(x) = \{x\}$
- $sub(r\ s) = \{r\ s\} \cup sub(r) \cup sub(s)$,
- $sub(\lambda x.r) = \{\lambda x.r\} \cup \{r\}$,

Definicja. Ścieżki $\lambda x.x(\lambda z.z)$

- paths(x) = $\{\epsilon\}$
- paths(r s) = $\{\epsilon\} \cup \{L\, p : p \in paths(r)\} \cup \{R\, p : p \in paths(s)\}$
- paths $(\lambda x.r) = \{\epsilon\} \cup \{*\, p : p \in paths(r)\}$

Definicja. $occ(\lambda x.x(\lambda z.x), *R) = \lambda z.x)$

- $occ(t, \epsilon) = t$,
- $occ(rs, L\, p) = occ(r, p)$
- $occ(rs, R\, p) = occ(s, p)$
- $occ(\lambda x.r, *p) = occ(r, p)$,

Definicja. Alfakonwersja t $\equiv_\alpha$ wtw

1. paths (t) = paths (s)
2. $\{p \in paths(t) \mid occ(t, p) \in FV(t)\} = \{p \in paths(s) \mid occ(s, p) \in FV(s)\}$
   $\{p \in paths(t) \mid occ(t, p) \in BV(t)\} = \{p \in paths(s) \mid occ(s, p) \in BV(s)\}$
3. $p \in paths(t)$ i $occ(t, p) \in FV(t)$ to $occ(t, p) = occ(s, p)$ i to samo dla s,
4. Jeżeli p jest wystąpieniem zmiennej związanej w t to jest też wystąpieniem zmiennej związanej w s i ich wystąpienia wiążące są równe.

19 Wykład 13.05.2008

19.1 Lambda, lambda, lambda...

Definicja. t $\equiv_\alpha$ s wtw gdy:

1. zbiory ścieżek w t i s są równe,
2. zbiory wystąpień zmiennych wolnych w t i s są równe,
3. zbiory wystąpień zmiennych związanych w t i s są równe,
4. jeżeli ϱ jest wolnym wystąpieniem zmiennej, to $occ(t, \varrho) = occ(s, \varrho)$,
5. jeżeli ϱ jest wystąpieniem zmiennej związanej, to $binder(t, \varrho) = binder(s, \varrho)$,

Definicja. Term jest regularny, jeżeli $FV(t) \cap BV(t) = \varnothing$, oraz każda zmienna związana ma w t dokładnie jedno wystąpienie wiążące.

Fakt. *Jeżeli $S \subseteq V$ jest taki, że $|V \backslash S| = \infty$, to dla każdego termu t istnieje t' taki że $t' \equiv_\alpha t$ i $BV(t) \cap S = \varnothing$.*

Wniosek: w każdej klasie abstrakcji $[t]_{\equiv_\alpha}$ istnieją reprezentanci „omijający" zbiór S. Inaczej: dla wszelkich klas abstrakcji $[t]_{\equiv_\alpha}$ i $[s]_{\equiv_\alpha}$ istnieją reprezentanci t' i s' tacy, że $t'[x/s']$ jest wykonalne.

Definicja. Podstawowa relacja dedukcji: $(\lambda x.t)r \ \to_\beta \ t'[x/r]$, gdzie $t' \equiv_\alpha t$ i takie, że $t'[x/r]$ jest wykonalne.

- $\dfrac{t \to_\beta s}{t \to_{\beta_1} s}$,
- $\dfrac{t \to_{\beta_1} s}{tr \to_{\beta_1} sr}$,
- $\dfrac{t \to_{\beta_1} s}{rt \to_{\beta_1} rs}$,
- $\dfrac{t \to_{\beta_1} s}{\lambda x.t \to_{\beta_1} \lambda x.s}$,

$\to_\beta$ – zwrotne i przechodnie domknięcie $\to_{\beta_1}$,
$\equiv_\beta$ – najmniejsza relacja równoważności zawierająca $\to_{\beta_1}$,

Definicja. Redukt termu t – taki term s, że $t \to_\beta s$, V – redukt t $(t, s) \in E$ wtw. $t \to_{\beta_1} s$.

Definicja. Term jest (słabo) normalizowalny, jeżeli istnieje s w postaci normalnej t. że $t \to_\beta s$.

Twierdzenie. Church – Rosser[18] Niezależnie od tego jak bardzo się rozejdziemy, to zawsze możemy się jeszcze zejść...

Twierdzenie. R – słabo konfluentna

- jeśli xRy i xRz to istnieje r takie że yR^*r i zR^*r

[18]Romantyczne twierdzenie...

20 Wykład 29.05.2008

20.1 Rachunek lambda z typami

$$t ::= x \mid y \mid ts \mid \lambda x.t \mid n \mid + \mid true \mid false \mid \text{if } t \text{ then } t \text{ else } t$$

$$\sigma ::= int \mid bool$$

$$\tau ::= \sigma \to \tau \mid \sigma$$

- $\dfrac{}{x : int} \quad x \in V_{int}$
- $\dfrac{}{y : bool} \quad y \in V_{bool}$
- $\dfrac{t : \sigma \to \tau \quad s : \sigma}{ts : \tau}$
- $\dfrac{t : bool \quad s : \sigma \quad r : \sigma}{\text{if } t \text{ then } s \text{ else } r : \sigma}$
- $\dfrac{t : \tau}{\lambda x.t : int \to \tau} \quad x \in V_{int}$
- $\dfrac{}{n : int}$
- $\dfrac{t : int \quad s : int}{t + s : int}$
- $\dfrac{}{true, false : bool}$
- $\dfrac{t : \tau}{\lambda y.t : bool \to \tau} \quad y \in V_{bool}$

Powyżej był sobie fortran, a teraz jest sobie pascal... (czy coś na kształt).

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

$$\sigma ::= o \mid \sigma_1 \to \sigma_2$$

- $\dfrac{}{\Gamma, x : \sigma \vdash x : \sigma}$
- $\dfrac{\Gamma \vdash t : \sigma \to \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash ts : \tau}$
- $\dfrac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x : \sigma \to \tau}$
- $\Gamma = \{x_1 : \sigma_1, ..., x_n : \sigma_n\}$

20.2 Kontrola typów

$$\Gamma \vdash t : ...$$

Typowanie a la Church (explicite):

- $\text{type}(\Gamma, t) = \sigma$
- $\text{type}(\Gamma, x) = \Gamma(x)$
- **type**(Γ, `ts`) =
  let
  - σ = **type**(Γ, `t`)
  - τ = **type**(Γ, `s`)

  in
  if σ `jest postaci` $\tau \to \sigma'$
  then **return** σ'
  else **undefined**

Typowanie a la Curry (implicite):

- $$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}$$
- $$\frac{\Gamma \vdash t : \sigma \to \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash ts : \tau}$$
- $$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

Term = preterm dla którego istnieje typowanie.

20.3 Rekonstrukcja typów – błędna, ale za to jaka ładna!

1. $t = x$ $\quad \Gamma \vdash x : \beta \quad \dfrac{\Gamma, x : \gamma \vdash t : \delta}{\Gamma \vdash \lambda x.t : \beta}$
2. $t = t_1 t_2 \quad \dfrac{\Gamma \vdash t_1 : \gamma \quad \Gamma \vdash t_2 : \delta}{\Gamma \vdash t_1 t_2 : \beta}$

- $\text{type}(\Gamma, t) = (\Gamma', \sigma)$
- $\text{type}(\Gamma, x) = \Gamma(x)$
- **type**(Γ, `ts`) =
  let
  - (Γ', σ) = **type**(Γ, `t`)
  - (Γ'', τ) = **type**(Γ′, `s`)

  in
  if σ `jest postaci` $\sigma_1 \to \sigma_2'$
  then $\Theta = mgu(\sigma_1, \tau)$
  return $(\Gamma''\Theta, \sigma_2\Theta)$
  else **undefined**

- **type**$(\Gamma,\ \lambda x.t)\ =$
  let
  $\gamma\ =$ `fresh variable`
  $\Gamma'\ = \Gamma \cup \{x:\gamma\}$
  $(\Gamma'' \cup \{x:\sigma\}, \tau)\ =$ **type**$(\Gamma',\ t)$
  in
  return $(\Gamma'', \sigma \to \tau)$

21 Wykład 03.06.2008

21.1 Lambda c.d.

$$\Lambda\text{-termy:} \quad t \ ::= \ x \mid t_1 t_2 \mid \lambda x.t$$
$$\text{typy:} \quad \sigma \ ::= \ \alpha \mid \sigma_1 \to \sigma_2$$

Mamy typowania **Church-style – explicite** oraz **Curry-style – implicite**. W 1978 Robin Milner oraz jego doktorant Luis Damas pracowali nad językiem ML. Programista, gdy program jest poprawny, nie musi widzieć typów – program jest wtedy prostszy. Pisanie typów explicite jest potrzebne, gdy chcemy np. zawęzić typy, lub gdy program z jakichś przyczyn jest nietypowalny. **Mads Tofte – Anno Domini** – odrobaczanie millienium bug'a.

- jak wyliczyć najogólniejszy unifikator dwóch typów?
  - $\mathtt{mgu}(\sigma_1 \to \sigma_2, \tau_1 \to \tau_2)\ =$
    let
    $\Theta_1\ = \mathtt{mgu}(\sigma_1, \tau_1)$
    $\Theta_2\ = \mathtt{mgu}(\sigma_2\Theta_1, \tau_2\Theta_1)$
    in
    $\Theta_1\Theta_2$
  - $\mathrm{mgu}(\alpha,\ \alpha) = []$
  - $\mathrm{mgu}(\alpha,\ \sigma) = \begin{cases} [\alpha/\sigma], & \text{gdy } \alpha \notin FV(\sigma) \\ \bot, & \text{w p.p.} \end{cases}$
  - $\mathrm{mgu}(\sigma, \alpha) = \mathrm{mgu}(\alpha, \sigma)$
- do tych λ-termów chcemy dodać typy:
  - $\Gamma = \{x_i : \sigma_i\}_{i=1}^{n}$,
  - $\Gamma \vdash t : \sigma$,
  - $\Gamma, x : \sigma \equiv \Gamma \cup \{x : \sigma\}$ przy czym $x \notin Dom(\Gamma)$,
  - $\dfrac{}{\Gamma, x : \sigma \vdash x : \sigma}$,
  - $\dfrac{\Gamma \vdash t : \sigma \to \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash ts : \tau}$,
  - $\dfrac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$,
- rachunek intuicjonistyczny...
- 1968 – Logika/Hindley, Języki Programowania/Strachey-Milner...
- Izomorfizm Curry'ego-Howard'a,

21.2 Rekonstrukcja typów

- $\text{type}(\Gamma, t) = (\Theta, \sigma)$
- $\text{type}(\Gamma, x) = \begin{cases} \text{„nie zadeklarowana zmienna”, gdy } x \notin Dom(\Gamma) \\ ([], \Gamma(x)) \end{cases}$
- $\underline{\textbf{type}}(\Gamma,\ \texttt{ts}) =$

  $\underline{\textbf{let}}$

  $(\Theta_1, \sigma_1) = \underline{\textbf{type}}(\Gamma,\ \texttt{t})$

  $(\Theta_2, \sigma_2) = \underline{\textbf{type}}(\Gamma\Theta_1,\ \texttt{s})$

  $\beta = \texttt{fresh variable}$

  $\Theta_3 = \texttt{mgue}(\sigma_1\Theta_2,\ \sigma_2 \to \beta)$

  $\underline{\textbf{in}}$

  $(\Theta_1\Theta_2\Theta_3, \beta\Theta_3)$
- $\underline{\textbf{type}}(\Gamma,\ \lambda x.t) =$

  $\underline{\textbf{let}}$

  $\beta = \texttt{fresh variable}$

  $(\Theta_1, \sigma_1) = \underline{\textbf{type}}(\Gamma \cup \{x : \beta\}, t)$

  $\underline{\textbf{in}}$

  $(\Theta_1, \beta\Theta_1 \to \sigma_1)$

22 Wykład 03.06.2008

22.1 ...

...

Spis treści

1 Wykład 04.03.2008 **1**

2 Wykład 06.03.2008 **1**

3 Wykład 11.03.2008 **2**
3.1 Dwa fajne sposoby na generowanie liczb pierwszych 2
3.2 Fajny sposób na ciąg fibonacciego 2
3.3 Monoid (półgrupa z jedynką) 2
3.4 Klasy typów 2
3.5 O monadach 3

4 Wykład 13.03.2008 **3**
4.1 Monoid raz jeszcze 3
4.2 O ciągu fibonacciego 3
4.3 Monady 5

5 Wykład 18.03.2008 **7**

6 Wykład 20.03.2008 **7**

7 Wykład 27.03.2008 **7**
7.1 Semantyka Denotacyjna 7
7.2 Rodzaje semantyki formalnej (TWi) 8

8 Wykład 01.04.2008 **9**
8.1 Przypomnienie 9
8.2 Teoria Dowodu 10
8.3 Mały wstęp do składni 10

9 Wykład 03.04.2008 **11**
9.1 Składnia 11
9.2 Semantyka Denotacyjna 11
9.2.1 Funkcja semantyczna 12
9.2.2 Powtórka z logiki 13

10 Wykład 08.04.2008 **13**
10.1 Wstęp 13
10.2 Semantyka operacyjna 14
10.2.1 Semantyka Dużych Kroków 14
10.2.2 Semantyka Małych Kroków 16

11 Wykład 10.04.2008 **17**
11.1 Wstęp 17
11.2 Semantyka aksjomatyczna – Floyd 17

12 Wykład 15.04.2008 **19**
12.1 Wprowadzenie 19
12.2 Asercje 19

13 Wykład 17.04.2008 **21**
13.1 Ciąg dalszy semantyk . . . 21
13.2 O algorytmie unifikacji . . . 22
13.3 Najsłabsze warunki wstępne . . . 22
13.4 Termy algebraiczne . . . 23
13.5 Algebry . . . 24

14 Wykład 22.04.2008 **24**
14.1 Algebra abstrakcyjna . . . 24
14.1.1 Słowo wstępne (TWi) . . . 24
14.1.2 Składnia . . . 24
14.1.3 Algebra, interpretacje zmiennych, interpretacje termów . . . 25
14.1.4 Semantyka denotacyjna algebraicznie . . . 26
14.1.5 Sygnatury i termy (TWi) . . . 27
14.1.6 Termy nad pojedynczym gatunkiem (TWi) . . . 28

15 Wykład 24.04.2008 **29**
15.1 O unifikacji . . . 29
15.2 Podstawienie (TWi) . . . 30
15.3 Algorytm unifikacji (TWi) . . . 31

16 Wykład 29.04.2008 **33**
16.1 Teorie syntaktyczne (TWi) . . . 33
16.2 Algebry (TWi) . . . 34
16.3 Homomorfizmy algebr (TWi) . . . 35
16.4 Podalgebry i algebry generowane (TWi) . . . 36
16.5 Zasada indukcji (TWi) . . . 36
16.6 Konstruktory (TWi) . . . 37
16.7 Teorie semantyczne (TWi) . . . 37

17 Wykład 06.05.2008 **38**
17.1 Mini-Haskell . . . 38
17.2 39
17.3 Semantyka algebraiczna . . . 39

18 Wykład 08.05.2008 **39**
18.1 Smutek i nostalgia . . . 39
18.2 Rachunek Lambda . . . 40

19 Wykład 13.05.2008 **42**
19.1 Lambda, lambda, lambda... . . . 42

20 Wykład 29.05.2008 **43**
20.1 Rachunek lambda z typami . . . 43
20.2 Kontrola typów . . . 44
20.3 Rekonstrukcja typów – błędna, ale za to jaka ładna! . . . 44

21 Wykład 03.06.2008 **45**
21.1 Lambda c.d. . . . 45
21.2 Rekonstrukcja typów . . . 46

22 Wykład 03.06.2008 46
22.1 46