Denotační sémantika programovacího jazyka

doc. Dr. Ing. Miroslav Beneš
katedra informatiky, A-1007
☎ 59 732 4213

Obsah přednášky

- Princip denotační sémantiky
- Sémantické funkce
- Výrazy
- Příkazy
- Vstup a výstup
- Kontinuace
- Program

Motivace

- Specifikace programovacího jazyka
  - **syntaxe** – struktura jazykových konstrukcí
  - **sémantika** – význam jazykových konstrukcí
- Metody popisu sémantiky
  - slovní popis – nepřesný
  - formální definice
    - axiomatická sémantika
    - operační sémantika
    - *denotační sémantika*

Denotační sémantika

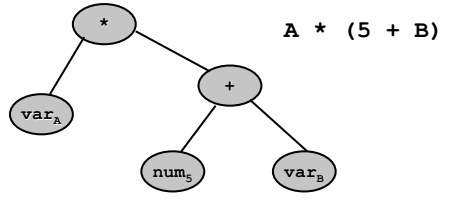
- Popis významu programu pomocí funkcí
  - program :: Input → Output
  - využití λ-kalkulu
  - popis prostředky funkcionálního jazyka
- Princip kompozicionality
  - význam složené konstrukce je dán kombinací významů jednotlivých složek
    $[[E_1+E_2]] = [[E_1]] + [[E_2]]$
  - není vždy splněno – konstrukce pro paralelní výpočty

Abstraktní syntaxe

- Zjednodušená struktura programu
  - ano - operátory, operandy
  - ne – závorky, priorita, asociativita, oddělovače
  - E → E+E | E-E | E*E | E/E | num | var
- Abstraktní syntaktický strom (AST)
  - vnitřní uzly jsou tvořeny operátory
  - listy obsahují operandy

Příklad AST

`E → E+E | E-E | E*E | E/E | num | var`



A * (5 + B)

Sémantické funkce

- Přiřazují význam jednotlivým syntaktickým konstrukcím
  - výraz → hodnota
  - příkaz → akce
- Syntaktické domény
  - představují pojmenované třídy syntaktických konstrukcí
  - výraz: **Exp**, příkaz: **Com**, deklarace: **Dec**, program: **Prog** apod.
  - Sémantické funkce jsou definovány pro každou doménu samostatně

Příklady sémantických funkcí

- e :: Exp -> Int
  - e [[1 + 1]] = e[[1]] + e[[2]] = 2
  - významem výrazu je číslo
- p :: Prog -> ([Int] -> [Int])
  - p [[i = read; write 2*i]] = λ (x:_) . [2 * x]
  - p [[write 40+2]] = λ _ . [42]
  - významem programu je funkce, která transformuje seznam vstupních hodnot na seznam hodnot výstupních

Výrazy s konstantami

```
data Exp = Add Exp Exp      -- x+y
         | Mul Exp Exp      -- x*y
         | Neg Exp          -- -x
         | Num Int          -- num

e :: Exp -> Int
e (Add e1 e2) = (e e1) + (e e2)
e (Mul e1 e2) = (e e1) * (e e2)
e (Neg e1)    = - (e e1)
e (Num x)     = x
```


Výrazy s proměnnými

- Význam (hodnota) výrazu závisí na hodnotách proměnných
  - e [[x + 3]] {x → 7} = 10
- Hodnota proměnných se během výpočtu nemění (*tento předpoklad brzy opustíme!)*
- Valuační funkce
  - přiřazuje hodnotu proměnným – model paměti
  - `type Store = String -> Int`
  - musí být parametrem sémantické funkce *e*

Výrazy s proměnnými

```
data Exp =  ...
         |  Var String

type Store = String -> Int

e :: Exp -> Store -> Int
e (Add e1 e2) s = (e e1 s) + (e e2 s)
e (Mul e1 e2) s = (e e1 s) * (e e2 s)
e (Neg e1)    s = - (e e1 s)
e (Num x)     _ = x
e (Var v)     s = s v
```


Výrazy s přiřazením

- Hodnoty proměnných (valuační funkce) se mohou během vyhodnocení výrazu změnit

  a + 2 * (a = c + 3) + 5 * a
- Sémantická funkce *e* musí vrátit i novou valuační funkci

  e :: Exp -> Store -> (Int, Store)

Výrazy s přiřazením

```
data Exp = ...
         | Asgn String Exp

e :: Exp -> Store -> (Int, Store)
e (Asgn v e1) s =
    let (v1,s1) = e e1 s
        s2 v'  = if v'=v then v1 else s1 v'
    in (v1, s2)

e (Add e1 e2) s =
    let (v1, s')  = e e1 s
        (v2, s'') = e e2 s'
    in (v1+v2, s'')

e (Num x)     s = (x, s)
```


Příkazy

- Neprodukují hodnotu
- Významem je vedlejší efekt příkazu
  - změna stavu programu – např. hodnot proměnných, vstup/výstup apod.

```
data Com = Eval Exp       -- e;
         | If Exp Com     -- if e then c
         | While Exp Com  -- while e do c
         | Seq Com Com    -- c1; c2

c :: Com -> Store -> Store
```


Příkazy

```
c :: Com -> Store -> Store
c (Eval e1) s = let (_,s') = e e1 s in s'

c (If e c1) s = let (v1,s') = e e1 s
                in if v1 == 0 then s'
                   else c c1 s'

c p@(While s1 c1) =
          let (v1, s') = e e1 s
          in if v1 == 0 then s'
             else c p (c c1 s')

c (Seq c1 c2) s   = c c2 (c c1 s)
```


Vstup a výstup

- Stav programu zahrnuje
  - okamžité hodnoty proměnných – Store
  - nezpracovaný vstup – seznam čísel
  - (vytvořený výstup – seznam čísel)

```
type Input = [Int]
type Output = [Int]
type State = (Store, Input, Output)
```

Čtení hodnoty ze vstupu

```
data Exp = …
         | Read

e :: Exp -> State -> (Int, State)

e Read (s, (x:xs), os)
             = (x, (s, xs, os))

e Read (_, [], _)
             = error "Konec vstupu"
```


Zápis hodnoty na výstup

```
data Com = …
           | Write Exp

c :: Com -> State -> State
c (Write e1) s =
    let (v1, (s', is, os)) = e e1 s
    in (s', is, os++[v1]))
```

- **vyhodnotí se výraz v operandu**
- **hodnota se přidá do výstupu**

Kontinuace

- **`write (3 + 5) * 2`**
  - vypočtu 3+5
  - hodnotu 8 a nový stav předám zbytku programu
  - zbytek programu vypočte hodnotu 8*2, zobrazí hodnotu 16 a skončí
- *kontinuace výrazu* je funkce, která dostane hodnotu, nový stav výpočtu a vrátí výsledek zbývající části programu

```
type ECont = Int -> State -> Output
```


Kontinuace

- write 3; write 5;
  - zobrazím 3
  - nový stav předám zbytku programu
  - zbytek programu zobrazí ještě hodnotu 5 a skončí
- *kontinuace příkazu* je funkce, která dostane nový stav výpočtu a vrátí výsledek zbývající části programu

```
type CCont = State -> Output
```

Kontinuace

- Sémantické funkce
  - obdrží jednu nebo více kontinuací
  - mohou (ale nemusí) jim předat řízení
- Výraz
  - `type ECont = Int -> State -> Output`
  - `e :: Exp -> State -> ECont -> Output`
- Příkaz
  - `type CCont = State -> Output`
  - `c :: Com -> State -> CCont -> Output`

Výrazy s kontinuacemi

```
type ECont = Int -> State -> Output
e :: Exp -> State -> ECont -> Output

e (Add e1 e2) s ec =
    e e1 s (\v1 s' ->
        e e2 s' (\v2 s'' ->
            ec (v1+v2) s''))
```


Příkazy s kontinuacemi

```
type CCont = State -> Output
c :: Com -> State -> CCont -> Output

c (Seq c1 c2) s cc =
    c c1 s (\s' ->
        c c2 s' (\s'' -> cc s'))
c (If e1 c1) s cc =
    e e1 s (\v1 s' ->
        if v1 == 0 then cc s'
                   else c c1 s' cc
```


Použití kontinuací

- Ošetření chyb
  - žádná kontinuace se nevyvolá, ukončí se výpočet
  - výjimky – v případě chyby se předá řízení kontinuaci svázané s ošetřením výjimky
- Sémantika příkazů skoku (break, goto, ...)
  - s cílem skoku je svázána kontinuace
  - ta se použije při realizaci skoku
- Volání podprogramů
  - kontinuace příkazu volání podprogramu představuje „návratovou adresu“ pro příkaz return

Modely výpočtu programu

- Zobrazení výsledku až na konci výpočtu
  - výsledkem je seznam čísel nebo chyba

```
data Output =    OK [Int]
             |   Err String
```

- Průběžné zobrazování výsledků
  - výsledkem je seznam čísel ukončený příznakem OK nebo chybou

```
data Value = I Int | OK | Err String
type Output = [Value]
```


Počáteční valuační funkce

- Všechny proměnné mají nulovou hodnotu

```
type Store = String -> Int
emptyStore :: Store
emptyStore id = 0
```

- Všechny proměnné jsou nedefinované

```
type Store = String -> Maybe Int
emptyStore :: Store
emptyStore id = Nothing
```


Počáteční stav programu

- Proměnné
  - počáteční valuační funkce
- Vstup
  - celý vstupní seznam
- Výstup
  - není součástí stavu

```
data State = State store :: Store,
                   input :: Input
initialState :: Input -> State
initialState inp = State store = inp,
                         input = emptyStore
```


Vyhodnocení programu

- Program je tvořen příkazem

```
type Prog = Com
```

- Sémantická funkce pro program

```
p :: Prog -> Input -> Output
p body inp =
  c body (initialState inp) (\_->[OK])
```

Úkoly pro cvičení

- Seznamte se s ukázkovou implementací interpretu jazyka tiny
- Rozšiřte interpret o následující konstrukce:
  - logické operátory And, Or, Not
  - příkaz For
  - funkci Eof
- Pro testování využijte dodaného překladače jazyka tiny do datových struktur pro Haskell

Domácí úkoly

- Definujte abstraktní syntaxi a sémantiku podmíněného příkazu if-then-else
  - bez použití kontinuací
  - s kontinuacemi
- Definujte abstraktní syntaxi a sémantiku příkazu repeat-until (do-while)
  - bez použití kontinuací
  - s kontinuacemi