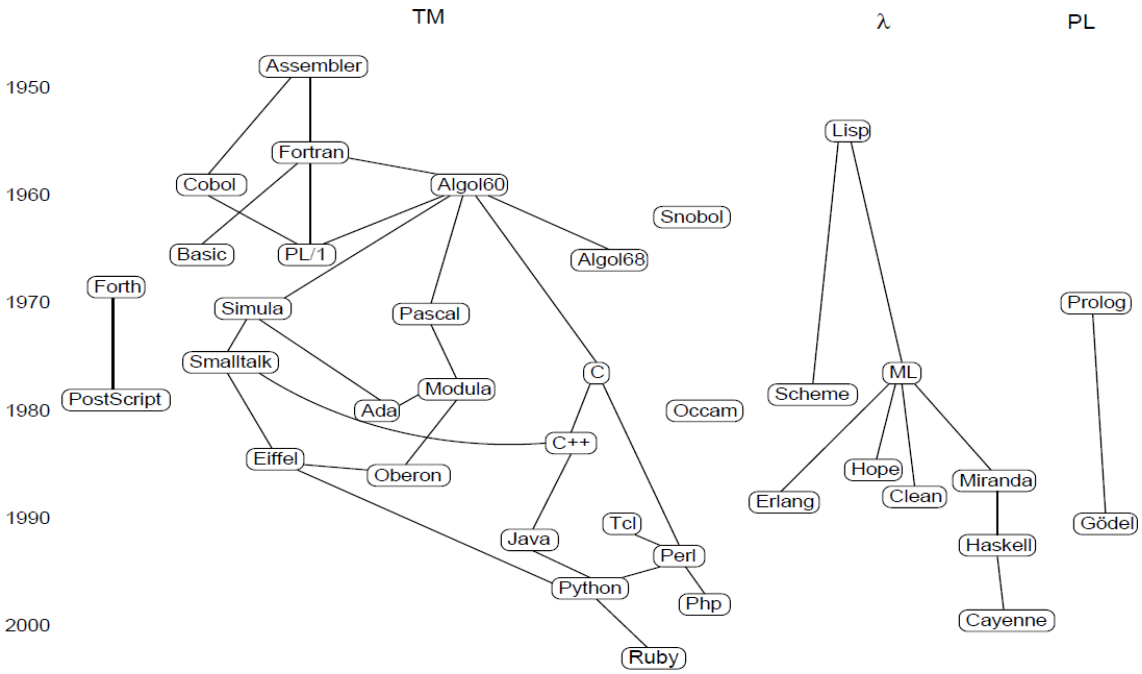
PB006 – Principy programovacích jazyků – poznámky

- Výpisky ze slidů pana Škarvady. V podstatě nic není navíc, spíš se jedná o zkrácení a přeformátování. Za bezchybnost neručím!



1 Úvod

1.1 Aspekty programovacího jazyka

- Pragmatika
  - vztah k výpočtu
  - implementace kompilátoru, interpretu, …
  - oblast nasazení
- Syntax
  - struktura a forma programů v jazyce
  - popsána formální gramatikou – ve třech úrovních (lexikální, bezkontextová, kontext omezení)
- Sémantika
  - význam
  - vztah mezi programem a výpočetním modelem

2 Syntax

- Popsána formální gramatikou $G = (N, \Sigma, \mathrm{P}, \mathrm{S})$,
- V případě programovacích jazyků bývá bezkontextová, tj.
  - $P \subseteq (N \times (N \cup \Sigma)^+) \cup \{(S, \epsilon)\}$.

2.1 Konkrétní syntax

- definuje vlastní jazyk
- dostatečně mnoho terminálů

2.1.1 Lexikální syntax (mikrosyntax)

- k vymezení množiny *lexikálních atomů* (tvoří ‚slovník')
- popsána – regulární nebo jednoduchou bezkontextovou gramatikou
- terminály – znaky

2.1.2 Frázová syntax (makrosyntax)

- popisuje úplnou strukturu programu (tvoří množinu správně vytvořených vět)
- popsána – bezkontextovou gramatikou
- terminály – lexikální atomy

2.1.3 Kontextová omezení (statická sémantika)

- podmínky vymezující podmnožinu jazyka (pravidla vylučující syntakticky správné, ale nesmyslné věty)

2.2 Abstraktní syntax

- popisuje strukturu
- nezávislá na konkrétní reprezentaci
- obsahuje hlavně neterminály

3 Typy

3.1 Pojetí typů

- množiny hodnot
- heterogenní algebry => množiny hodnot spolu s operacemi na nich
- variety heterogenních algeber => množiny s operacemi a axiomy

3.2 Typy a typové konstruktory

3.2.1 Kartézský součin ($A \times B$)

- Množina všech uspořádaných dvojic (a, b) takových, že $a : A, b : B$.
- Projekční funkce (selektory):
  - $fst : A \times B \to A$
  - $fst\ (x, y) = x$
  - $snd : A \times B \to B$
  - $snd\ (x, y) = y$

3.2.2 Disjunktní sjednocení ($A + B$)

- Sjednocení množiny všech dvojic (L, a) s množinou všech dvojic (R, b), kde $a : A, b : B$. L, R jsou příznaky původu prvku a, b.
- Inserční funkce (konstruktory):
  - $inl : A \to A + B$
  - $inl\ x = (L, x)$
  - $inr : B \to A + B$
  - $inr\ y = (R, y)$

3.2.3 Pole

- Zobrazení (konečného) ordinálního typu I do typu T.
  - Konečný typ je ordinální ⇔ existuje bijekce $ord : I \to \{1, \dots, n\}$, kde $n = |I|$.
- Zápis: Array $I\ T$

3.2.4 Funkce

- Zápis: $A \to B$
- $f(x, y) \cong g\ x\ y$

3.2.5 Potenční množiny

- Set A typ všech podmnožin A
- $A \to \text{Bool}$ typ všech predikátů nad A
- Každé hodnotě $a : \text{Set}\ A$ odpovídá *charakteristická funkce* $\chi_a : A \to \text{Bool}$ taková, že pro každé $x : A$ platí $x \in a \ \Leftrightarrow \chi_a(x) = true$.

3.2.6 Prázdný typ

- disjunktní sjednocení nulového počtu typů
- Void, ⊥
- Neobsahuje žádnou hodnotu.

3.2.7 Jednotkový typ

- kartézský součin nulového počtu prvků
- Unit, ⊤
- Má jedinou hodnotu – uspořádanou nultici ().

3.2.8 Výčtové typy

- Případ disjunktního sjednocení – všechny inserce jsou typu $\text{Unit} \to T$.

3.2.9 Rekursivní (induktivní) typy

- Je-li $F(t)$ zápis typů obsahující typovou proměnnou t, pak $\mu\ t.F(t)$ je (množinově *nejmenší* typ vyhovující rovnici $t = F(t)$.
- $T = \mu\ t.F(t) \sim T = F(T)$
- např. konečné seznamy

3.2.10 Korekursivní (koinduktivní) typy

- Je-li $F(t)$ zápis typů obsahující typovou proměnnou t, pak $\nu\ t.F(t)$ je (množinově *největší* typ vyhovující rovnici $t = F(t)$.
- $T = \nu\ t.F(t) \sim T = F(T)$
- např. nekonečné seznamy

3.3 Monomorfismus a polymorfismus

3.3.1 Monomorfní typu

- a) základní typy
- nebo
- b) Pomocí typových konstruktorů složené z monomorfních typů.

3.3.2 Polymorfní typy

- Zastupují celou množinu monomorfních typů.

3.3.2.1 Parametrický polymorfismus

- V typových výrazech se vyskytují typové proměnné, za něž lze dosadit libovolný typ.
- př. $\forall b.\,\mathrm{Array}\ (0..9)b$

3.3.2.2 Podtypový (inklusivní) polymorfismus

- Na množině typů je zavedeno uspořádání $\preccurlyeq:$ a každá hodnota má kromě svého nejmenšího typu i všechny jeho nadtypy.
- $A \preccurlyeq: B$ – „A je podtypem B“
- **Pravidlo subsumpce**
  - Nechť $A \preccurlyeq: B$, a je typu A, potom také a je typu B.
  - Zápis:

$$\frac{a : A \qquad A \preccurlyeq: B}{a : B}$$

- **Kovariace**
  - Typový konstruktor S je *kovariantní*, když

$$\frac{A \preccurlyeq: A'}{S\ A \preccurlyeq : S\ A'}$$

  - Příklad – typový konstruktor *Set.*

$$\sigma \preccurlyeq: \sigma' \Rightarrow \mathrm{Set}\ \sigma \preccurlyeq: \mathrm{Set}\ \sigma'$$

    - Přirozené číslo je celým číslem, tedy i množina přirozených čísel je množinou celých čísel.
- **Kontravariace**
  - Typový konstruktor T je *kontravariantní*, když

$$\frac{A \preccurlyeq: A'}{T\ A' \preccurlyeq : T\ A}$$

  - Příklad – typový konstruktor *Predicate.* $\mathrm{Predicate}\ \alpha = \alpha \to \mathrm{Bool}.$

$$\sigma \preccurlyeq: \sigma' \Rightarrow \mathrm{Predicate}\ \sigma' \preccurlyeq: \mathrm{Predicate}\ \sigma$$

    - Přirozené číslo je celým číslem, tedy predikát nad celými čísly je predikátem nad přirozenými čísly.
- **Kovariace a kontravariace u typových konstruktorů větší arity**
  - se zavádí zvlášť pro každý parametr.
  - n-ární typový konstruktor S je ve svém i-tém typovém parametru $(1 \leq i \leq n)$ *kovariantní*, když

$$\frac{A_i \preccurlyeq: A'}{S\ A_1 \dots A_n \preccurlyeq : S\ A_1 \dots A_{i-1}\ A'\ A_{i+1} \dots A_n}$$

  - n-ární typový konstruktor T je ve svém i-tém $A_1 \dots A_{i-1}$typovém parametru $(1 \leq i \leq n)$ *kontravariantní*, když

$$\frac{A \preccurlyeq: A'}{T\ A_1 \dots A_{i-1}\ A' A_{i+1} \dots A_n \preccurlyeq : T\ A_1 \dots A_n}$$

  - Příklad
    - Typový konstruktor → je v prvním parametru *kontravariantní* a ve druhém parametru *kovariantní.*

3.4 Přetížení

- Symbol je přetížený, označuje-li více hodnot různých typů.

- **Kontextově nezávislé**
  - Funkce lze vyřešit jen z typu argumentů.
- **Kontextově závislé**
  - Typy argumentů nestačí, bere se širší kontext.
- Symbol g je přetížen dvěma různými typy $\sigma_1 \to \tau_1,\ \sigma_2 \to \tau_2$.
  - *Kontextově nezávislé* musí být — $\sigma_1 \neq \sigma_2$.
  - *Kontextově závislé* musí být — $\tau_1 \neq \tau_2$.

3.5 Koerce

- implicitní zobrazení hodnot jednoho typu do hodnot jiného typu

3.6 Druhy

- *Druhy* slouží ke klasifikaci typů, typových konstruktorů, typových funkcí apod.
- Symbol $*$ značí druh všech typů.

3.7 Hodnotově závislé typy

- Typové konstruktory parametrizovány typy + *hodnotami.*
- Příklad vektor
  - Typ $\mathrm{Vec}\ n$ všech n-složkových vektorů reálných čísel.
  - $\mathrm{Vec} : Nat \to *$
- Používáno hlavně akademickými jazyky:
  - Agda, Cauenne, …

4 Sémantika

- Přiřazuje významy programům nebo jejím částem.

4.1 Formální sémantika

4.1.1 Operační sémantika

- Množinou pravidel popisujících výpočet abstraktního počítače — výsledek výpočtu je významem programu.
- Rozlišujeme:
  - *strukturní (sos)*
  - *přirozená (bos)*

4.1.2 Axiomatická sémantika

- Množina tvrzení (tzv. teorie) o nějakém systému, v nemž probíhá výpočet.
- Zejména u imperativních jazyků — tvrzení se vyjadřují o stavech.

4.1.3 Denotační sémantika

- Definována na programech a komponentách explicitně — množina funkcí přiřazujících částem programů význam.
- $\Rightarrow$ Přiřazení prvků sémantických domén $\to$ programovým konstrukcím.

- **Příklad:**
  - P množina konstrukcí
  - $MVar$ množina (přepisovatelných) proměnných
  - V množina hodnot
  - $S = MVar \longrightarrow V$ množina stavů
  - $[\![\_]\!]\_ : P \times S \longrightarrow S$ stavový transformátor
  - $\mathcal{M}[\![\_]\!]\_ : P \times S \longrightarrow V$ významová funkce
  - $[\![x++]\!]\sigma = \sigma'$, kde
    - $\sigma'(x) = \sigma(x) + 1$
    - $\forall y, y \neq x . \sigma'(y) = \sigma(y)$
  - $\mathcal{M}[\![x++]\!]\sigma = \sigma(x)$

4.1.3.1 Nedeterministická sémantika

- …

4.1.3.2 Deterministická sémantika

- Významem výrazu e je jeho hodnota (prvek sémantické domény).
- V imperativních jazycích závisí sémantika výrazu na stavu $\sigma \in S$, takže
  - $\mathcal{M}[\![e]\!] : S \longrightarrow Val$
- Význam výrazu e ve stavu σ je hodnota $\mathcal{M}[\![e]\!]\sigma \in Val$.

- V jazycích bez vedlejších efektů, ale s proměnnými závisí sémantika výrazů na tzv. hodnotovém kontextu $\epsilon \in Env$. Potom
  - $\mathcal{M}[\![e]\!] : Env \longrightarrow Val$
- Význam výrazu e ve stavu ϵ je hodnota $\mathcal{M}[\![e]\!]\epsilon \in Val$.
- **Příklad:**
  - $\mathcal{M}[\![0]\!]\sigma = 0$
  - $\mathcal{M}[\![1]\!]\sigma = 1$
  - ⋮
  - $\mathcal{M}[\![e_1 + e_2]\!]\sigma = \mathcal{M}[\![e_1]\!]\sigma + \mathcal{M}[\![e_2]\!]\sigma$
  - $\mathcal{M}[\![e_1 * e_2]\!]\sigma = \mathcal{M}[\![e_1]\!]\sigma \cdot \mathcal{M}[\![e_2]\!]\sigma$
  - $\mathcal{M}[\![\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3]\!]\sigma = \begin{cases} \mathcal{M}[\![e_2]\!]\sigma, & \text{pokud } \mathcal{M}[\![e_1]\!]\sigma = \text{tt} \\ \mathcal{M}[\![e_3]\!]\sigma, & \text{pokud } \mathcal{M}[\![e_1]\!]\sigma = \text{ff} \\ \bot, & \text{jinak} \end{cases}$

4.2 Výrazy

- **Literály** – označují hodnoty
- **Jména** – konstanty, parametry fcí, proměnné
- **Výrazy** popisující hodnoty **složených typů**
- **Podmíněné výrazy** -- if…then…else, case…of…
- **Aplikace** funkce na argumenty (volání funkce)
- **Abstrakce** („uzávěry“)
- **Příkazy**

4.3 Funkcionální a procedurální abstrakce

- *Funkcionální abstrakce* se provádí nad **výrazem** → některé jeho (volné) proměnné se prohlásí za parametry výsledné funkce.
- Počet těchto parametrů určuje tzv. *aritu*.
- *Procedurální abstrakce* se provádí podobně nad **příkazem**.
- Příklad:
  - float sqr (float x) { **return** $x * x$; }

4.4 Proměnné

4.4.1 Funkcionální, logické

- *Čisté proměnné* (v matematickém smyslu),
- označují hodnotu,
- slouží i k označení parametrů fcí.

4.4.2 Imperativní

- *Přepisovatelné proměnné*,
- označují paměťové místo.
- Paměťové místo je úložiště hodnoty.
- Ve většině jazyku se však jménem proměnné označuje i daná hodnota (⇒ automatické dereferencování).

4.5 Příkazy

- Významem příkazu je *stavový transformátor* $s : S \longrightarrow S$.
- $MVar$ množina program. proměnných reprezentujících paměťová místa
- Val množina hodnot, která mohou být do těchto míst uloženy
- $\sigma : MVar \longrightarrow Val$ stav
- S množina všech stavů
- Je-li p příkaz, pak se jeho *stavový transformátor* značí $[\![p]\!] : S \longrightarrow S$.

4.5.1 Příkazy obsahující výrazy BEZ VEDLEJŠÍCH EFEKTŮ

Prázdný příkaz **skip**

$$[\![\textbf{skip}]\!] = id, \quad [\![\textbf{skip}]\!]\sigma = \sigma \text{ pro všechny stavy } \sigma \in S$$

Přiřazovací příkaz $v := e$

$$[\![v := e]\!]\sigma = \sigma', \quad \text{kde } \sigma'(v) = \mathcal{M}[\![e]\!]\sigma,$$
$$\forall x \in MVar - \{v\}.\ \sigma'(x) = \sigma(x)$$
$$\mathcal{M}[\![e]\!]\sigma \text{ je hodnota výrazu } e \text{ ve stavu } \sigma$$

Sekvence **begin** $p_1; \ldots; p_k$ **end**

$$[\![\textbf{begin } p_1; \ldots; p_k \textbf{ end}]\!] = [\![p_k]\!] \circ \cdots \circ [\![p_1]\!]$$

tj. pro všechna $\sigma \in S$

$$[\![\textbf{begin } p_1; \ldots; p_k \textbf{ end}]\!]\sigma = [\![p_k]\!]([\![p_{k-1}]\!](\ldots([\![p_2]\!]([\![p_1]\!]\sigma))\ldots))$$

Podmíněný příkaz **if** e **then** p_1 **else** p_2

$$[\![\textbf{if } e \textbf{ then } p_1 \textbf{ else } p_2]\!]\sigma = \begin{cases} [\![p_1]\!]\sigma, \text{ pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{tt} \\ [\![p_2]\!]\sigma, \text{ pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{ff} \\ \bot \text{ jinak} \end{cases}$$

Příkaz cyklu **while** e **do** p

$$[\![\textbf{while } e \textbf{ do } p]\!]\sigma = \begin{cases} \sigma, \text{ pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{ff} \\ [\![\textbf{while } e \textbf{ do } p]\!]([\![p]\!]\sigma), \text{ pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{tt} \\ \bot \text{ jinak} \end{cases}$$

Tvrzení: Pro každý výraz e a příkaz p platí

$$[\![\textbf{while } e \textbf{ do } p]\!] = [\![\textbf{if } e \textbf{ then begin } p; \textbf{while } e \textbf{ do } p \textbf{ end}]\!]$$

Násobné větvení

$$\begin{array}{c} \textbf{case } e \textbf{ of} \\ v_1 \to p_1 \\ v_2 \to p_2 \\ \vdots \\ v_k \to p_k \end{array}$$

$$[\![\textbf{case } e \textbf{ of } v_1 \to p_1, \ldots, v_k \to p_k]\!]\sigma = [\![p_i]\!]\sigma, \text{ kde}$$
$$i = \min\{j | 1 \leq j \leq k, \mathcal{M}[\![v_j]\!]\sigma = \mathcal{M}[\![e]\!]\sigma\},$$
existuje-li toto minimum, jinak σ

4.5.2 Příkazy obsahující výrazy S VEDLEJŠÍMI EFEKTY

- Připustíme, aby příkazy (výrazy typu Command) byly podvýrazy výrazů jiného typu.
- Pak jsou všechny výrazy stavovými transformátory $[\![e]\!] : S \to S$, ale kromě toho vracejí hodnotu $\mathcal{M}[\![e]\!] : S \to Val$.
- Například v C ve stavu σ, $\sigma(x) = 5$, je
  - o $[\![x++]\!]\,\sigma = \sigma',\ \sigma'(x) = 6, \sigma'(y) = \sigma(y)$ pro $y \neq x$
  - o $\mathcal{M}[\![x++]\!]\,\sigma = 5$

$$[\![\textbf{skip}]\!] = id$$

$$[\![x := e]\!]\sigma = \sigma', \quad x \in \mathit{MVar}$$
$$\sigma'(x) = \mathcal{M}[\![e]\!]\sigma$$
$$\sigma'(y) = ([\![e]\!]\sigma)(y) \ \text{ pro } y \in \mathit{MVar} - \{x\}$$

$$[\![e_l := e_r]\!]\sigma = \sigma', \quad v = \mathcal{M}[\![e_l]\!]([\![e_r]\!]\sigma)$$
$$\sigma'(v) = \mathcal{M}[\![e_r]\!]\sigma$$
$$\sigma'(y) = ([\![e_l]\!]([\![e_r]\!]\sigma))(y) \ \text{ pro } y \in \mathit{MVar} - \{v\}$$

$$[\![\textbf{begin } p_1; \ldots; p_k \textbf{ end}]\!] = [\![p_k]\!] \circ \cdots \circ [\![p_1]\!]$$

$$[\![\textbf{if } e \textbf{ then } p_1 \textbf{ else } p_2]\!]\sigma = \begin{cases} [\![p_1]\!]([\![e]\!]\sigma), & \text{pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{tt} \\ [\![p_2]\!]([\![e]\!]\sigma), & \text{pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{ff} \\ \bot & \text{jinak} \end{cases}$$

$$[\![\textbf{while } e \textbf{ do } p]\!]\sigma = \begin{cases} [\![e]\!]\sigma, & \text{pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{ff} \\ [\![\textbf{while } e \textbf{ do } p]\!]([\![p]\!]([\![e]\!]\sigma)), & \text{pokud } \mathcal{M}[\![e]\!]\sigma = \mathit{tt} \\ \bot & \text{jinak} \end{cases}$$

4.6 Řadící příkazy

- Skoky
  - Explicitní přenesení řízení výopčtu do jiné části programu.
- Úniky
  - Ukončují provádění složeného příkazu, který unikový příkaz obsahuje.
- Vyjímky

4.7 Volání funkce a předávání parametrů

- Aplikace funkce na argumenty = tzv. volání funkce.

4.7.1 Volání hodnotou

Nechť definice funkce (procedury) je $f\ (x_1, \ldots, x_n) \ = \ b$

a její volání je $f\ (a_1, \ldots, a_n)$, kde $a_1, \ldots, a_n$ jsou výrazy.

Pak

$$\begin{aligned} [\![f(a_1, \ldots, a_n)]\!]\sigma &= [\![b]\!]\sigma' \\ \mathcal{M}[\![f(a_1, \ldots, a_n)]\!]\sigma &= \mathcal{M}[\![b]\!]\sigma' \end{aligned}$$

kde pro všechna $i,\ 1 \leq i \leq n$

$$\begin{aligned} \sigma_0 &= \sigma \\ \sigma_i &= [\![a_i]\!]\sigma_{i-1} \\ \sigma'(x_i) &= \mathcal{M}[\![a_i]\!]\sigma_{i-1} \\ \sigma'(y) &= \sigma_n(y) \text{ pro každé } y \notin \{x_1, \ldots, x_n\} \end{aligned}$$

4.7.2 Volání jménem

Nechť definice funkce f je

$$f(x_1, \ldots, x_n) = b$$

Pak

$$\begin{aligned} [\![f(a_1, \ldots, a_n)]\!]\sigma &= [\![b']\!]\sigma \\ \mathcal{M}[\![f(a_1, \ldots, a_n)]\!]\sigma &= \mathcal{M}[\![b']\!]\sigma \end{aligned}$$

kde b' vznikne z b současnou substitucí výrazů $a_1, \ldots, a_n$ za všechny (volné) výskyty formálních parametrů $x_1, \ldots, x_n$, tj.

$$b' = [a_1/x_1, \ldots, a_n/x_n]b$$

- Implementuje se pomocí nulárních (bezparametrických) funkcí pro každý skutečný parametr předávaný jménem -- tzv. *thunks*.

```
function sum (i : Ref Int, m : Int, n : Int, name x : Real) : Real
  = begin s : Ref Real;
          s := 0;
          for i := m to n do s := (s) + x;
          return (s)
    end;
k : Ref Int;
  ⋮
sum (k, 1, 10, 1/(k * (k + 1)))
```

```
double sum ( int *i, int m, int n, double (*x)() )
  {
    double s;
    s = 0;
    for (*i = m; *i <= n; (*i)++) { s = s + (*x)(); }
    return s;
  }
...
double thunk1 (void) { return 1/(k*(k+1)); }
...
sum (&k, 1, 10, &thunk1);
```

4.7.3 Volání „dle potřeby“ – líná aplikace

- Varianta volání jménem, argumenty se ale vyhodnocují nejvýše jednou.
- U referenčně transparentních jazyků ⇒ jazyky bez vedlejších efektů.

- Počet vyhodnocování argumentů:
  - volání hodnotou — 1
  - volání jménem — $0 \ldots n$
  - volání dle potřeby — $0 \ldots 1$

4.7.4 Volání odkazem

- Skutečné parametry smějí být jen (adresovatelná) paměťová místa (Ref T).
- **Neprovádí** se jejich implicitní dereferencování.

4.7.5 Volání výsledkem

- Skutečné parametry jsou typu Ref a, tj. adresovatelná paměťová místa.

Je-li definice funkce (procedury)

$$f(\textbf{out}\ y_1, \ldots, y_n) = b$$

Pak

$$\begin{aligned} [\![f(w_1, \ldots, w_n)]\!]\sigma &= \sigma'' \\ \mathcal{M}[\![f(w_1, \ldots, w_n)]\!]\sigma &= \mathcal{M}[\![b]\!]\sigma_n \end{aligned}$$

kde pro všechna $i,\ 1 \leq i \leq n$

$$\begin{aligned} \sigma_i &= [\![w_i]\!]\sigma_{i-1}, \quad \sigma_0 = \sigma \\ v_i &= \mathcal{M}[\![w_i]\!]\sigma_{i-1} \\ \sigma' &= [\![b]\!]\sigma_n \\ \sigma''(v_i) &= \sigma'(y_i) \\ \sigma''(u) &= \sigma'(u) \ \text{pro}\ u \notin \{v_1, \ldots, v_n\} \end{aligned}$$

³B006

4.7.6 Volání hodnotou-výsledkem

- Skutečné parametry jsou typu Ref a, tj. adresovatelná paměťová místa.

Je-li definice procedury (funkce) $f(\textbf{inout}\ z_1, \ldots, z_n) = b$

Pak

$$\begin{aligned} [\![f(w_1, \ldots, w_n)]\!]\sigma &= \sigma''' \\ \mathcal{M}[\![f(w_1, \ldots, w_n)]\!]\sigma &= \mathcal{M}[\![b]\!]\sigma' \end{aligned}$$

kde pro všechna $i,\ 1 \leq i \leq n$

$$\begin{aligned} \sigma_i &= [\![w_i]\!]\sigma_{i-1}, \quad \sigma_0 = \sigma \\ v_i &= \mathcal{M}[\![w_i]\!]\sigma_{i-1} \\ \sigma'(z_i) &= \sigma_n(v_i) \\ \sigma'(u) &= \sigma_n(u) \ \text{pro}\ u \notin \{z_1, \ldots, z_n\} \\ \sigma'' &= [\![b]\!]\sigma' \\ \sigma'''(v_i) &= \sigma''(z_i) \\ \sigma'''(u) &= \sigma''(u) \ \text{pro}\ u \notin \{v_1, \ldots, v_n\} \end{aligned}$$

4.7.7 Smíšené volání – hodnotou, výsledkem, hodnotou-výsledkem

- Víceparametrická funkce své parametry vyhodnocuje různými způsoby, podle předpisu v hlavičce funkce.

$$f(\textbf{in}\ x_1, \ldots, x_m, \textbf{out}\ y_1, \ldots, y_n, \textbf{inout}\ z_1, \ldots, z_r) = b$$

4.7.8 Shrnutí typů volání (není ze slidů, nezaručuji 100% správnost)

- **Volání jménem:**
  - Argument není vyhodnocen před voláním funkce.
  - *Je substituován přímo do funkce, tzn. je **pokaždé vyhodnocován** pouze v místech, kde se ve funkci objevuje.*

    - Použití v C:
        - Pomocí ukazatele na unární funkci (viz příklad výše).
- **Volání hodnotou:**
    - Argument je vyhodnocen a výsledná hodnota je zkopírována do odpovídající proměnné ve funkci.
    - *Změny proměnné ve funkci se mimo ní **neprojeví**.*
    - Použití v C:
        - defaultní volání
- **Volání odkazem:**
    - Změny obsahu přepisovatelné proměnné určené formálním parametrem, mění *současně* obsah proměnné, která je skutečným argumentem při volání.
    - *Změny proměnné ve funkci se **okamžitě projevují** i mimo ní.*
    - Použití v C:
        - Předání odkazem na proměnnou. (Popřípadě v C++ předáním referencí.)
- **Volání hodnotou-výsledkem:**
    - Změny obsahu přepisovatelné proměnné určené formálním parametrem, se dějí pouze *lokálně* a obsah skutečného argumentu se změní až v okamžiku návratu.
    - *Změny proměnné ve funkci se mimo funkci **projeví až po návratu**.*
- **Volání výsledkem:**
    - Argument je neinicializovaný, resp. může být inicializovaný, ale jeho hodnota není v těle funkce brána v potaz (například tím, že hodnota formální proměnné je inicializována v těle funkce na určitou hodnotu, která není závislá na hodnotě skutečné proměnné).
    - *Není důležitá hodnota proměnné při volání funkce + hodnota se do proměnné **zapíše až po návratu** z funkce.*
- *Pozor na rozdíl mezi voláním odkazem a voláním hodnotou-výsledkem — může mít vliv na výsledný stav.*

5 Viditelnost

5.1 Viditelnost jazykových entit

- **Statická**
    - Pro každou definici jazykové entity (konstanty, proměnné, …) je statickou sémantikou určena oblast platnosti definice.
- **Dynamická**
    - Viditelnost jazykových entit závisí na momentálně aktivních programových jednotkách (blocích, funkcích, …) v době běhu.
    - Téměř nepoužívaná (Lisp, Snobol).

5.2 Třídy a objekty

- **Objekt**
    - Modul s privátními proměnnými (atributy) a veřejnými operacemi (metodami).
- **Třída**
    - („generický objekt“) popisuje typ objektu.
    - Vlastní objekty se nazývají *instance* třídy.

5.2.1 Dědičnost

- Třída A je *podtřídou* třídy B (značíme $A \leq: B$), když dědí metody třídy B a případně přidává další.
- **Jednoduchá** $\Rightarrow$ každá třída má nejvýše jednu bezprostřední nadtřídu.
- **Násobná** (jinak).

5.3 Vlastnosti OO jazyků

- Třídy a objekty (statické nebo dynamické).
- Dědičnost a inkluzní polymorfismus
  - typový systém s podtypy — $a : A, A \leq: A' \Rightarrow a : A'$
- Viditelnost omezená na objekty
  - Řízená pomocí public/protecte/private.
- V (dynamických) objektech mohou být atributy i metody
  - statické — definované ve třídách
  - dynamické — definované v objektech
- Jazyky:
  - **Simula 67**
    - Považován za nejstarší jazyk, v němž se objevily *některé principy OO* programování.
  - **Smalltalk**
    - První čistě objektový OO jazyk; netypovaný.
  - **Eiffel**
    - Čistě objektový, typovaný.
  - **C++**
    - Není čistě objektové.
  - **Java**
    - Není čistě objektová (má i primitivní typy).
  - **Python, Ruby**
    - Interpretovaný bytový kód.
  - **Ada**
    - Staticky typovaná, na zakázku MO USA v 70. letech.
  - **OCaML**
    - Objektová verze funkcionálního ML.

6 Správa paměti

6.1 Paměťové třídy dat

6.1.1 Persistentní data

- Existují nezávisle na výpočtu.
- Obvykle na vnějších paměťových médiích (soubory, databáze).

6.1.2 Transientní data

- Existují pouze po dobu výpočtu.
- Obvykle v procesem adresovatelné části operační paměti (hodnoty proměnných, …), ale i vně (dočasné soubory, …).

- **Statická**
- **Automatická (zásobníková)**
- **Dynamická (haldová)**

6.2 Správa paměti

6.2.1 Statická

- Prováděná překladačem.
- *[Instrukce programu, systémová data, statická data, vyrovnávací paměti, …]*

6.2.2 Zásobníková

- V době výpočtu při zajájení/ukončení aktivace procedury, bloku, …
- *[lokální data v blocích, parametry funkcí a procedur, pomocné datové struktury ve funkcích, návratové adresy, …]*

6.2.3 Dynamická

- V době výpočtu, prováděna speciálními procedurami pro alokaci, dealokaci, přesouvání, scelování,…
- 1**. řízena explicitně programem** *[malloc, free, …]*
- 2. **spouštěná výhradně „run-time podporou“ výpočtu** *[garbage collectiong, setřásání]*

6.3 Fáze správy paměti

- alokace
- dealokace
- obnovení volné paměti
- scelování (změna polohy obsazených a volných bloků)

6.4 Správa dynamické paměti

- **Inicializace** – vytvoření seznamu volných bloků
- **Alokace bloku** (požadované velikosti)
- **Uvolnění bloku** – může způsobit vznik *slepého odkazu* nebo *smetí*
- **Úklid smetí (garbage collection)**
    - bloky mají příznak dostupnosti (1 bit)
    - nastaví se na nedostupné
    - postupuje se od známých ukazatelů a všechny navštívené bloky se označí „dostupné“
    - nedostupné bloku se dají do seznamu volných
- **Scelování (spojování sousedních volných bloků)**
- **Setřásání (úplné scelování)** – způsobí změny ukazatelů

7 Paradigmata

- **Deklarativní paradigma**
  - *Program popisuje, co je výsledkem.*
  - ***Funkcionální*** – program je výraz a výpočet je jeho úprava.
  - ***Logické*** – program je teorie a výpočet je odvozen z ní.
- **Imperativní paradigma**
  - *Program popisuje, jak se dospěje k výsledku.*
  - ***Procedurální*** – výpočet je provádění procedur nad daty.
  - ***Objektové*** – výpočet je předávání zpráv mezi objekty.

7.1 Logické paradigma

- Vztahy mezi hodnotami pomocí tzv. *Hornových klausulí*.
- Program se skládá ze seznamu klausulí (teorie) a z formule (cíle, dotazu).
- Abstraktní syntax:
  - Program ::== Klausule* Formule
  - Klausule ::== Formule :- Formule*
  - Formule ::== Pred (Term*)
  - Term ::== Var | Fun (Term*)
- Kontextová omezení (Statická sémantika)
  - Predikátové a funkční symboly mají konsistentní arity.
  - Levá strana (tj. závěr) implikace se nazývá *hlava klausule*.
  - Předpoklady implikace se nazývají *podcíle*.
  - Klausule s nulovým počtem podcílů je tzv. *fakt*.
  - Proměnné v klausuli, které jsou na levé straně – *univerzálně kvantifikovány*.
  - Ostatní proměnné, které jsou jen na pravé straně – *existenčně kvantifikovány*.

$$\forall x\, \forall y\ .\ \mathit{Potomek}(x, y) \Leftarrow \exists z\ .\ \mathit{Rodič}(y, z) \wedge \mathit{Potomek}(x, z)$$

7.1.1 Termy

- $Zero$ – nulární funkční symbol (konstanta).
- $Succ$ – unární funkční symbol.
- **Uzavřené termy:**
  - $Zero$
  - $Succ(Zero)$
  - $Succ(Succ(Zero))$
  - …
- **Termy s proměnými**
  - $Succ(x)$
  - $Succ(Succ(Succ(y)))$
  - …

7.1.2 Klauzule

- $LessThan(Zero, Succ(y))$.
- $LessThan(Succ(x), Succ(y)) :- LessThan(x, y)$.

7.1.3 Dotaz

- $LessThan(x, Succ(Succ(Zero)))$.

7.2 Sémantika logického programu

7.2.1 Substituce

- *Subst* je množina všech substitucí, tj. konečných zobrazení
  - $\sigma : \mathrm{Var} \rightarrow_f \mathrm{GTerm}$,
- kde $\mathrm{GTerm} = \{\tau \in \mathrm{Term} \mid \tau \text{ neobsahuje proměnné}\}$ je množina všech tzv. *uzavřených* termů.
- Substituci σ lze rozšířit na termy $\Rightarrow$ rozšířená substituce $\bar{\sigma} : \mathrm{Term} \rightarrow \mathrm{GTerm}$.

7.2.2 Logické jazyky

- **Prolog**
  - 70. léta
  - netypovaný jazyk
  - plochá struktura a viditelnost:
    - všechny predikáty jsou globální
    - všechny proměnné jsou lokální (v klauzuli)
  - není čistě logický
- **Gödel**
  - 90. léta
  - typovaný jazyk
  - modulární struktura
  - parametrický polymorfismus
- **Mercury**
  - polovina 90. let
  - multiparadigmatický jazyk (logický + funkcionální)

7.2.3 Aplikace logického programování

- zpracování přirozeného jazyka
- simulace
- symbolická manipulace s výrazy, řešení rovnic
- Výhody
  - vyšší úroveň
  - logika programu oddělena od řízení výpočtu
- Nevýhody
  - efektivita
  - nevýhodné pro aplikace s intenzivním I/O

7.3 Funkcionální paradigma

- Nerozlišuje stavy, výpočet je jednostavový.
  - Nemá přepisovatelné proměnné.
- parametrický polymorfismus, typové a konstruktorové třídy
- uživatelské typy
- líné vyhodnocování, nekonečné datové struktury

7.3.1 Sémantika

- Hodnotový kontext (prostředí)
  - $\epsilon : \text{Var} \longrightarrow \text{Val}$
- Env … množina všech hodnotových kontextů
- Sémantická funkce
  - $\mathcal{M} : \text{Term} \times \text{Env} \longrightarrow \text{Val}$
- Val … sémantická doména

7.3.2 CPO

- CPO je uspořádaná množina, v níž má každý nejvýše spočetný řetězec supremum.

monotonní funkce

f je monotonní, právě když pro každé dva prvky x, y takové, že $x \sqsubseteq y$, platí $f(x) \sqsubseteq f(y)$

spojité funkce

f je spojitá, právě když pro každý spočetný řetězec $x_1, x_2, \ldots$ platí $f\left(\bigsqcup(x_1, x_2, \ldots)\right) = \bigsqcup(f(x_1), f(x_2), \ldots)$

- Každá spojitá funkce je monotónní. (Obrácená věta neplatí.)
- Každá monotónní funkce na *konečné* doméně je spojitá.

7.3.3 Sémantické domény

- Primitivní, tzv. *ploché*:
  - Unit, Bool, Nat, Integer, Char

7.3.4 Sémantika konstant

- nezávisí na hodnotovém kontextu ϵ.

Pro libovolné $\varepsilon \in Env$

$$\begin{array}{rcl} \mathcal{M}[\![\mathit{True}]\!]\varepsilon & = & tt \\ \mathcal{M}[\![\mathit{False}]\!]\varepsilon & = & ff \\ \mathcal{M}[\![\mathit{0}]\!]\varepsilon & = & 0 \\ & \vdots & \end{array} \quad \left| \quad \begin{array}{rcl} \mathcal{M}[\![\mathit{succ}]\!]\varepsilon & : & \mathit{Val} \to \mathit{Val}, \ldots \\ \mathcal{M}[\![\mathit{iszero}]\!]\varepsilon & & \ldots \\ \mathcal{M}[\![\mathit{not}]\!]\varepsilon & & \ldots \\ & \vdots & \end{array} \right.$$

7.3.5 Sémantika výrazů

$$\mathcal{M}[\![x]\!]\varepsilon = \varepsilon(x)$$

$$\mathcal{M}[\![f\ e]\!]\varepsilon = (\mathcal{M}[\![f]\!]\varepsilon)(\mathcal{M}[\![e]\!]\varepsilon) \qquad \text{[líná aplikace]}$$

$$\mathcal{M}[\![g\ e]\!]\varepsilon = \begin{cases} \bot, & \text{když } \mathcal{M}[\![e]\!]\varepsilon = \bot \\ (\mathcal{M}[\![g]\!]\varepsilon)(\mathcal{M}[\![e]\!]\varepsilon) & \text{jinak} \end{cases} \qquad \text{[striktní aplikace]}$$

$$\mathcal{M}[\![\lambda x.\, e]\!]\varepsilon = f \ \text{ taková funkce z } \mathit{Val} \to \mathit{Val},$$
$$\text{že } \forall t \in \mathit{Val}.\ f(t) = \mathcal{M}[\![e]\!](\varepsilon[x \mapsto t])$$

$$\mathcal{M}[\![\textbf{if } e \textbf{ then } e' \textbf{ else } e'']\!]\varepsilon = \begin{cases} \mathcal{M}[\![e']\!]\varepsilon, & \text{když } \mathcal{M}[\![e]\!]\varepsilon = tt \\ \mathcal{M}[\![e'']\!]\varepsilon, & \text{když } \mathcal{M}[\![e]\!]\varepsilon = ff \\ \bot & \text{jinak} \end{cases}$$

7.3.6 Funkcionální jazyky

- **LISP**
  - konec 50. let
  - jednoduchá syntax, stejná pro data i algoritmy
  - není čistě funkcionální
  - netypovaný
  - dynamická viditelnost
  - dialekty: AutoLisp, eLisp, …
- **Scheme**
  - konec 50. let
  - netypovaný jazyk
  - statická viditelnost
  - striktní vyhodnocování
  - streamy
- **ML**
  - konec 70. let
  - typovaný jazyk
  - striktní vyhodnocování
  - dialekty: CaML, OCaML
- **Erlang**
  - dynamicky typovaný jazyk
  - striktní vyhodnocování
  - konstrukce pro podporu paralelního vyhodnocování

- **Hope, Clean, Orwell, Miranda**
    - konec 80. let
    - vesměs líně vyhodnocované
    - používané ve výuce
- **Opal**
    - 90. léta
    - čistě funkcionální
    - modulární
    - striktně vyhodnocovaný
- **Haskell**
    - čistě funkcionální
    - modulární
    - líně vyhodnocovaný
    - čisté začlenění imperativních konstrukcí do referenčně transparentního jazyka pomocí monadických typů a operací
- **Cayenne, Agda, Epigram**
    - experimentální jazyky
    - velmi silný (tedy nerozhodnutelný) typový systém