1) Sémantika: operační sémantika, denotační sémantika, pevný bod funkce, vázání jmen, stav a data programu.

Operační sémantika – sémantika malého kroku a velkého kroku. [TPJ přednáška 1]

Sémantika malého kroku (SSOS)

S = <CF, ➔ FC, IF, OF>

- Postupně se aplikují přepisovací pravidla

CF – doména konfigurací (obor hodnot)
➔ – přepisovací relace (transformuje konfigurace)
FC – nezjednodušitelné konfigurace (zapisuje se přeškrtnutou šipkou)
IF – vstupní funkce
OF – výstupné funkce

(Možné konce – konečný stav, uváznutí – např. něco, na co nemáme pravidlo, nekonečná smyčka)

CF = množina konfigurací

- Konfigurace = stav paměti + stav programu (paměť / program může být konstantní i nulový)
  - Konfigurace = přepisovací pravidla,
  - stav paměti = zobrazení „jméno proměnné“ → hodnota,
  - stav programu = seznam zbývajících přepisovacích pravidel
  - (VarName -> (Number U Boolean)) × Program

FC = množina finálních konfigurací

- Musíme říct, s čím skončíme
- Vybrat nějaké konfigurace, a když dojdu k nějaké z nich, tak jsem skončil
- Imperativní jazyk = FC jsou takové, které mají prázdný program

=> = přepisovací/přechodová relace

Konfigurace => konfigurace = přepisovací pravidla (o konfiguracích se mluví, je-li nějaká sémantika, jinak se říká stav)

(paměť, program) => (jiná/stejná paměť, jiný/stejný program)

Pokud už prvek nelze dále přepsat podle žádného pravidla, je v **normální formě. NF** jsou většinou finální konfigurace. ***obecně:*** NF = výsledek U zaseklý stav

- Silně normalizující: tvoří normální formu v každé cestě
- Slabě normalizující: Alespoň v jedné cestě tvoří normální formu.

IF = vstupní funkce

- Konverze dat od uživatele do počátečního stavu sémantiky
- Program × Input → CF
- IF(program) = ({}, program); {} = prázdná paměť
- Jiná syntaxe zápisu funkce:
  - IF = λ P **.** ({}, P)

OF = Výstupní funkce

- Konverze mezi finální konfigurací a výstupem programu

1.2 Small-Step Operational Semantics

Convention: $e, e', e_1, e_2, \ldots \in Expr$ and $n, n', n_1, n_2, \ldots \in Num$.

$$\frac{}{\triangle n \Rightarrow -n} \tag{3}$$

$$\frac{}{n \odot n' \Rightarrow n + n'} \tag{4}$$

$$\frac{e \Rightarrow e'}{\triangle e \Rightarrow \triangle e'} \tag{5}$$

$$\frac{e_1 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e' \odot e_2} \tag{6}$$

$$\frac{e_2 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e_1 \odot e'} \tag{7}$$

Operační sémantika velkého kroku (Big-Step Operational Semantics)

- Ze vstupů přepíše hned (co nejrychleji) na výsledek

1.3 Big-Step Operational Semantics

$$\frac{}{n \Longrightarrow n} \tag{8}$$

$$\frac{e \Longrightarrow n}{\triangle e \Longrightarrow -n} \tag{9}$$

$$\frac{e \Longrightarrow n \quad e' \Longrightarrow n'}{e \odot e' \Longrightarrow n + n'} \tag{10}$$

Please note that $\Rightarrow$ is relation on expressions (i.e. $\Rightarrow \in Expr \times Expr$), whereas $\Longrightarrow$ is relation between expressions and numbers (i.e. $\Longrightarrow \in Expr \times Num$).

Denotační sémantika

- Oproti operační sémantice se nezajímá o syntaxi (jak je to napsané – {,},(,),if, …), ale o sémantiku = jaký to má význam.

[[syntaktická algebra]] = sémantická algebra

[[3]] (dvojitá hran. závorka) = 3 (číslo se přepíše na číslo)

[[/\ ]] = λx.-x (unární mínus)

[[/\ E]] překlápí se /\ a E zvláště: → [[/\ ]] ([[E]]) – denotační sémantika trojúhelníka použitá na denotační sémantiku E. [[/\ ]] je vlastně funkce, definice viz předchozí

Př.:

[[/\ 5]] = [[/\ ]] [[5]] = λx.-x (5) = -5

- přepíše se to na sémantiku trojúhelníku aplikovanou na sémantiku podvýrazu (= e)

Pevný bod funkce

bod X, ve kterém platí F(X) = X.

- Využívá se pro rekurzivní funkce
- Y kombinátor v lambda kalkulu: Y = λf (λx . f(x x))(λx . f(x x))
- Př.: Generující funkce faktoriálu **fact = λF. λX.if x==0 then 1 else F(decrement(X))**
- Generující funkci dám do Y kombinátoru:
    - Y fact =
      = λf (λx . f(x x))(λx . f(x x)) fact =
      = (λx . fact(x x))(λx . fact(x x)) =
      = fact ((λx . fact(x x) (λx . fact(x x)) =
      = fact (Y fact)
    - Y fact je pevný bod funkce, která počítá faktoriál

Vázání jmen

- Lambda kalkulus
- λX.λY.A B C X Y = λX.(λY.(A B C)) X Y

- X a Y jsou vázané proměnné (je před nima lambda), A, B, C jsou volné proměnné = globální proměnné
- Pokud vázanou proměnnou přejmenuji, tak se nic nestane.
- Funkce, co mají jen vázané proměnné, vrátí při každém zavolání stejný výsledek. Funkce s volnými proměnnými jsou závislé na globálním kontextu.
    - Funkce, co mají jen vázané proměnné, se v λkalkulu nazývají **kombinátory.**

Stav a data programu

Stav = proměnné v prostředí, proměnné mají typ, prostředí = množina všech proměnných, kontext (v handoutech **Γ** (Gama))

Čistě funkční jazyky (a matematika) jsou bezestavové, stavové výpočty mohou být reprezentovány jako iterace skrz stavy.

Př.: Funkce na nalezení maxima z pole:

$$max : N^* \to N$$
$$max(\langle a_1, \ldots, a_n \rangle) = loop(\langle a_1, \ldots, a_n \rangle, 1, 0)$$
$$loop : N^* \times N \times N \to N$$
$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = m \quad \text{if } c > n$$
$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = loop(\langle a_1, \ldots, a_n \rangle, c+1, m) \quad \text{if } c \leq n \wedge a_c \leq m$$
$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = loop(\langle a_1, \ldots, a_n \rangle, c+1, a_c) \quad \text{otherwise}$$

Monády – struktury, co reprezentují výpočet jako sekvenci kroků.

Př.: Nalezení maxima pomocí monád:

$$State = N^* \times N \times N \tag{3}$$
$$Action = State \to State \tag{4}$$
$$Condition = State \to Boolean \tag{5}$$

$$updateMax : Action$$
$$updateMax(\langle a_1, \ldots, a_n \rangle, c, m) = (\langle a_1, \ldots, a_n \rangle, c, a_c) \tag{6}$$

$$updateNeeded : Condition$$
$$updateNeeded(\langle a_1, \ldots, a_n \rangle, c, m) = true \quad \text{if } a_c > m \tag{7}$$
$$updateNeeded(\langle a_1, \ldots, a_n \rangle, c, m) = false \quad \text{otherwise}$$

$$increaseIndex : Action$$
$$increaseIndex(\langle a_1, \ldots, a_n \rangle, c, m) = (\langle a_1, \ldots, a_n \rangle, c + 1, m) \tag{8}$$

$$finished : Condition$$
$$finished(\langle a_1, \ldots, a_n \rangle, c, m) = true \quad \text{if } c > n \tag{9}$$
$$finished(\langle a_1, \ldots, a_n \rangle, c, m) = false \quad \text{otherwise}$$

$$ifStatement : Condition \times Action \to Action$$
$$ifStatement(cond, body) = \lambda s.body(s) \quad \text{if } cond(s) = true \tag{10}$$
$$ifStatement(cond, body) = \lambda s.s \quad \text{otherwise}$$

$$forLoop : Condition \times Action \times Action \to Action$$
$$forLoop(cond, iter, body) = \lambda s.ifStatement(cond, forLoop(cond, iter, body)(iter(body(s)))) \tag{11}$$

$$max : N^* \to N$$
$$max(\langle a_1, \ldots, a_n \rangle) = \pi_3(forLoop(finished, increaseIndex, ifStatement(updateNeeded, updateMax)) (\langle a_1, \ldots, a_n \rangle, 1, 0)) \tag{12}$$

Vyhodnování proměnných

- call by name, call by value, call by denotation

Call by value

- parametr funkce se vyhodnotí při zavolání funkce, vkládá se hodnota.

Call by name

- parametr se v okamžiku volání vloží jako nevyhodnocený výraz na místa, kde je použit.

Call by denotation

- parametr se dosadí až při vyhodnocení výrazu

Data programu

- dělí se na součiny, sumy a sumy součinů

Součiny:

Positional data = N-tice, každý prvek může mít jiný typ

Sequence, List = pole

Named = třída

Nonstrict, stream = data, která se získají/vypočítají v okamžiku, kdy je potřeba (např. InputStream, odněkud se to vezme)

Sumy:

Union v C, nadtypy v Javě

Sumy součinů:

Binární a ternální operátory, double dispatch

2) Statická sémantika: typy, polymorfní typy, typy vyššího řádu, rekonstrukce (inference) typů, abstraktní typy.

Statická sémantika je řešena při překladu programu, zde jsou definovány a deklarovány jednotlivá pravidla a prvky programovacího jazyka. V těchto prvcích je zahrnuta jazyková konstrukce, její typy parametrů, význam příkazů a další prvky. Statická sémantika dále kontroluje statické typy a práci s tabulkou definovaných programových symbolů.

- Staticky tipované jazyky požadují uvedení datového typu u každé deklarace. Zde nelze deklarovat proměnnou, či funkci nebo objekt bez zadání datového typu.
- Všechny typové kontroly jsou prováděny staticky při překladu. Už při překladu má být každé proměnné přiřazen datový typ.
- Je možné daný datový typ přímo přetypovat. Přetypování především slouží k obcházení typových kontrol.
- Výhodou statického typování je lepší možnost odhalení typových chyb.
- Hlavní nevýhodou této metody je větší složitost programových konstrukcí, délka zdrojového kódu a tím i menší pružnost programovacího jazyka.
- Mezi neznámější zástupce staticky typovaných jazyků patří Java, Ada a jazyk C.

Typy:

- Základní typy (např. Number, Boolean, String)
- Typ má každá proměnná, výraz, funkce
- Typ = množina přípustných hodnot a operací s nimi
- TopType
- **Type preservation** = zachování typů během přepisovací relace
- **Progress** = když mám nějaký term v konfiguraci, tak pak to je buď finální konfigurace, nebo lze ještě nejméně jednou přepsat
    - Dk. Analýzou pravidel
- Type preservation + progress = **soundness**. Soundness je dle Píšeho a Buka velmi důležitá vlastnost typů.
- **Terminace** = nadefinuji „energii“, nadefinuji, že při každém přepsání se musí snížit.
- **Determinismus** = pokud se výraz přepíše na hodnotu *v* a tentýž výraz jinou cestou na hodnotu *v'*, pak *v = v'*. Platí to, protože konfulence relace.
- **Konfluence** = „strom vyhodnocování relace se rozdělí a pak se zase spojí“

Polymorfní typy

- Recursive type

$$\frac{\Gamma \cup \{X\} \vdash A}{\Gamma \vdash \mu X.A}$$

$$\frac{\Gamma \vdash e : \mu X.A}{\Gamma \vdash \texttt{unfold e} : A[X \mapsto \mu X.A]}$$

$$\frac{\Gamma \vdash e : A[X \mapsto \mu X.A]}{\Gamma \vdash \texttt{fold e} : \mu X.A}$$

- Universal type

$$\frac{\Gamma \cup \{X\} \vdash A}{\Gamma \vdash \forall X.A}$$

$$\frac{\Gamma \cup \{X\} \vdash e : A}{\Gamma \vdash \lambda X.e : \forall X.A}$$

$$\frac{\Gamma \vdash e : \forall X.A \quad \Gamma \vdash B}{\Gamma \vdash e(B) : A[X \mapsto B]}$$

Typy vyššího řádu

- Product type

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash A_1 \times A_2}$$

$$\frac{\Gamma \vdash e_1 : A_1 \quad \Gamma \vdash e_2 : A_2}{\Gamma \vdash (e_1, e_2) : A_1 \times A_2}$$

$$\frac{\Gamma \vdash e : A_1 \times A_2}{\Gamma \vdash \texttt{first}\ e : A_1}$$

$$\frac{\Gamma \vdash e : A_1 \times A_2}{\Gamma \vdash \texttt{second}\ e : A_2}$$

- Union type

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash A_1 + A_2} \tag{12}$$

$$\frac{\Gamma \vdash e : A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash \texttt{inLeft}_{A_2} e : A_1 + A_2} \tag{13}$$

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash e : A_2}{\Gamma \vdash \texttt{inRight}_{A_1} e : A_1 + A_2} \tag{14}$$

$$\frac{\Gamma \vdash e : A_1 + A_2}{\Gamma \vdash \texttt{isLeft}\ e : \mathit{Boolean}} \tag{15}$$

$$\frac{\Gamma \vdash e : A_1 + A_2}{\Gamma \vdash \texttt{isRight}\ e : \mathit{Boolean}} \tag{16}$$

$$\frac{\Gamma \vdash e : A_1 + A_2}{\Gamma \vdash \texttt{asLeft}\ e : A_1} \tag{17}$$

$$\frac{\Gamma \vdash e : A_1 + A_2}{\Gamma \vdash \texttt{asRight}\ e : A_2} \tag{18}$$

- Function type

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$$

$$\frac{\Gamma \cup \{(x, A)\} \vdash e : B}{\Gamma \vdash (\lambda x : A.e) : A \to B}$$

$$\frac{\Gamma \vdash e : A \to B \quad \Gamma \vdash p : A}{\Gamma \vdash e(p) : B}$$

- Record type

$$\frac{\Gamma \vdash A_1 \quad \ldots \quad \Gamma \vdash A_n}{\Gamma \vdash \texttt{Record}(l_1 : A_1, \ldots, l_n : A_n)} \tag{19}$$

$$\frac{\Gamma \vdash e_1 : A_1 \quad \ldots \quad \Gamma \vdash e_n : A_n}{\Gamma \vdash \texttt{record}(l_1{=}e_1, \ldots, l_n{=}e_n) : \texttt{Record}(l_1 : A_1, \ldots, l_n : A_n)} \tag{20}$$

$$\frac{\Gamma \vdash e : \texttt{Record}(l_1 : A_1, \ldots, l_j : A_j, \ldots, l_n : A_n)}{e.l_i : A_i} \tag{21}$$

Abstraktní typy

Fronta, HashMapa, Množina (Set), Seznam (List), Zásobník (Stack), graf, …

Rekonstrukce (inference) typů

- Z proměnné vyvodit její typ, není to ve staticky typovaných jazycích (C++)