Sémantika programovacích jazykov

William Steingartner

william.steingartner@tuke.sk

Katedra počítačov a informatiky FEI TU v Košiciach

Množiny

Množina je úplne definovaná svojimi prvkami. Napríklad:

$$A = \{a_1, a_2, \ldots, a_n\} \qquad B = \{b \mid P(b)\}$$

Konvencie:

- $\mathbb{N}$ pre množinu prirodzených čísel,
- $\mathbb{N}_0$ pre množinu prirodzených čísel obsahujúcu aj číslo 0,
- $\mathbb{Z}$ pre množinu celých čísel.

Množinové operácie:

$A \cup B = \{c \mid (c \in A) \vee (c \in B)\}$	zjednotenie
$A \cap B = \{c \mid (c \in A) \wedge (c \in B)\}$	prienik
$A \times B = \{(a, b) \mid (a \in A) \wedge (b \in B)\}$	binárny karteziánsky súčin

Relácie

Binárna relácia R z množiny A do množiny B je podmnožinou kartediánskeho súčinu

$$R \subseteq A \times B.$$

Ak prvky $a \in A$ a $b \in B$ sú v relácii R, zapisujeme to aRb.

Relácia $R \subseteq A \times A$ sa nazýva **predusporiadanou** (*preorder*) **na** A, ak pre všetky $a_1, a_2, a_3 \in A$ platí:

- $a_1 R a_1$ (reflexívnosť),
- ak $a_1 R a_2$ a $a_2 R a_3$, potom $a_1 R a_3$ (tranzitívnosť).

Relácia $R \subseteq A \times A$ sa nazýva **čiastočne usporiadanou** (*poset*) **na** A, ak pre všetky $a_1, a_2, a_3 \in A$ platí:

- $a_1 R a_1$ (reflexívnosť),
- ak $a_1 R a_2$ a $a_2 R a_1$, potom $a_1 = a_2$ (antisymetria),
- ak $a_1 R a_2$ a $a_2 R a_3$, potom $a_1 R a_3$ (tranzitívnosť).

Funkcie

Úplne definovaná funkcia z A do B

$$f : A \to B$$

je relácia, pre ktorú platí: pre každé $a \in A$ existuje **práve jedno** $b \in B$ také, že afb, čo zapisujeme:

$$f(a) = b,$$

pričom A je **definičný obor**, B je **obor hodnôt** funkcie f.

Ak chceme zapísať, že funkcia f priradí konkrétnemu prvku $a \in A$ určitý prvok $b \in B$, použijeme **iný druh šípky**:

$$a \mapsto b.$$

Čiastočne definovaná funkcia nie je definovaná pre všetky prvky definičného oboru. Zapisujeme ju:

$$f : A \rightharpoonup B.$$

Znamená to, že existuje $a \in A$ také, že $f(a) = \bot$.

Zloženie, currying

Zloženie (kompozícia) funkcií $f : A \to B$, $g : B \to C$, je funkcia z A do C:

$$g \circ f : A \to C \quad \text{taká, že pre každé } a \in A$$
$$(g \circ f)(a) = g(f(a))$$

Currying funkcií. Funkciu:

$$f : A \times B \to C$$

môžeme zapísať:

$$f : A \to B \to C.$$

Najprv sa f aplikuje na argument $a \in A$ a dostaneme **funkciu**:

$$f(a) : B \to C.$$

V druhom kroku sa aplikuje argument $b \in B$ a dostaneme **hodnotu funkcie**:

$$f(a, b) \in C.$$

Currying funkcií je implicitne asociatívny **sprava**!

Sémantika programov

Sémantika sa zaoberá významom programov písaných v programovacích jazykoch.

Programovacie jazyky obsahujú rôzne druhy **konštrukcií**, napr. výrazy, deklarácie, príkazy, a pod.

Napríklad:

- **čisté funkcionálne jazyky** majú deklarácie, typy, vzory, zhody, . . .
- **imperatívne jazyky** majú deklarácie, typy, bloky, formálne parametre, príkazy, . . .

Tieto rozdiely nie sú len na úrovni syntaxe, ale odrážajú podstatné rozdiely **vo význame** rôznych druhov konštrukcií.

Každá konštrukcia v sebe môže zahŕňať:

- **tok riadenia**,
- **tok informácií**.

Tok riadenia a informácií

Vykonávanie programu pozostáva z výpočtu na určitom počítači.

Výpočet na počítači je kombinácia podvýpočtov konštrukcií v programe.

Riadenie:

- **vtečie** do konštrukcie keď sa začne jej podvýpočet,
- **vytečie**, keď sa tento podvýpočet skončí.

Spracovanie informácií počas výpočtu odpovedá **toku informácií** medzi jednotlivými konštrukciami programu.

Konštrukcie sa môžu líšiť najmä v tom

- aký tok riadenia a
- aký tok informácií

spôsobia *z* a *do* svojich podkonštrukcií.

Tok riadenia

Medzi základné druhy toku riadenia patria:

- **Zoradenie do postupnosti** (*sequencing*) – riadenie vteká postupne do podkonštrukcií zhora nadol (zľava doprava), pokiaľ každý podvýpočet terminuje normálne.
- **Prekladanie** (*interleaving*) – riadenie sa presúva striedavo vpred a nazad medzi dvoma alebo viacerými podkonštrukciami.
- **Výber medzi alternatívami** (*choice*) – riadenie vtečie len do jednej alternatívy. Ak vykonanie zvolenej alternatívy zlyhá, môže sa vrátiť (*backtracking*) a skúsiť inú alternatívu.
- **Signalizovanie výnimočnej situácie** (*exception raising*) – preruší normálny tok riadenia, pokračuje uzatvárajúcou podkonštrukciou (*handler*), ktorá ju spracuje. Potom môže tok riadenia pokračovať nasledujúcou podkonštrukciou.
- **Iterácia** (*iteration*) – tok riadenia opakovane vteká do tej istej podkonštrukcie.
- **Volanie procedurálnej abstrakcie** (*procedural abstraction*) – tok riadenia vtečie do tejto konštrukcie, ktorá sa nachádza na inom mieste programu a po ukončení výpočtu sa pokračuje od bodu aktivácie.
- **Súbežné procesy** (*concurrent processes*) – delia tok riadenia do separátnych vlákien, ktoré môžu vyvolať potrebu synchronizácie.

Tok informácií

Medzi základné druhy toku informácií patria:

- **Výpočet hodnôt** – obvykle sa hodnoty počítajú vo výrazoch, ale napr. deklarácie počítajú prostredie, príkazy prázdne (*dummy*) hodnoty, a pod.
- **Použitie vypočítaných hodnôt** – informácie vytekajú z konštrukcií a vtekajú do iných. Výpočet hodnoty vždy vyžaduje termináciu výpočtu. Signalizovanie výnimočnej situácie možno chápať ako kombináciu toku riadenia (na iné miesto programu) a výpočet hodnoty, ktorá identifikuje príslušnú výnimočnú situáciu.
- **Účinok na pamäť** – môže sa alokovať nová bunka v pamäti, alebo sa do bunky uloží nová hodnota, alebo sa bunka dealokuje (uvoľní).
- **Viditeľnosť (*scope*)** – väzby vypočítané jednou podkonštrukciou môžu vtiecť do inej podkonštrukcie, ktorá je v oblasti viditeľnosti tejto väzby.
- **Odovzdávanie správ** (*message passing*) – konštrukcia získa informáciu, že jej kontext akceptuje správu a kontext získa informáciu uloženú v samotnej správe.

Paradigmy programovania

Programovacie jazyky možno klasifikovať podľa toho, ktorú **paradigmu programovania** podporujú najviac.

Niektoré jazyky určené pre jednu paradigmu programovania sa (po určitom úsilí) dajú použiť aj pre iné paradigmy.

Rozoznávame nasledujúce paradigmy programovania:

- imperatívne programovanie,
- funkcionálne programovanie,
- súbežné programovanie,
- objektovo orientované programovanie,
- logické programovanie.

Imperatívne programovanie

Imperatívne programovanie – kladie dôraz na zoradenie do postupnosti, iteráciu, procedurálnu abstrakciu a jej aktiváciu (volanie) a účinok na pamäť.

Charakteristiky imperatívneho programovania:

- početné zmeny hodnôt uložených v pamäti,
- samotné výpočty sú jednoduché,
- tok riadenia je vyjadrený postupnosťou príkazov, podmieňovacími príkazmi a príkazmi cyklu.

Jazyky podporujúce túto paradigmu sú napr.: C, Pascal, Algol60 a desiatky iných.

Funkcionálne programovanie

Funkcionálne programovanie – kladie dôraz na výpočet hodnôt, oblasti viditeľnosti väzieb, procedurálnu abstrakciu a jej aktiváciu.

Charakteristiky funkcionálneho programovania:

- programy obsahujú rozsiahle a komplexné výpočty hodnôt,
- programy obsahujú len málo alebo žiadne priradenia,
- tok riadenia je vyjadrený aktiváciou funkcií, funkcie počítajú hodnoty z hodnôt argumentov a odovzdávajú ich iným funkciám ako argumenty,
- oblasť viditeľnosti väzieb v deklaráciách je často rekurzívna.

Jazyky podporujúce túto paradigmu sú napr.: Lisp, Scheme, Miranda, Gofer, Standard ML, CAML, Haskell a mnoho iných.

Súbežné programovanie

Súbežné programovanie – kladie dôraz na súbežné procesy, výber medzi alternatívami a odovzdávanie správ.

Charakteristiky súbežného programovania:

- každý proces spravidla vykonáva malé množstvo výpočtov medzi prijímaním a odosielaním správ,
- výber medzi alternatívami sa robí podľa druhu prijímanej správy alebo na základe synchronizácie procesov.

Jazyky podporujúce túto paradigmu sú napr.: PL/1, Algol68, Modula, Ada, Occam, Concurrent ML a iné.

Objektovo orientované programovanie

Objektovo orientované programovanie sa často chápe ako imperatívne programovanie s pravidlami pre oblasti viditeľnosti väzieb.

Charakteristiky objektovo orientovaného programovania:

- používa väzby medzi objektmi a metódami,
- objekty sú fragmenty pamäte vytvorené buď inštanciovaním tried alebo klonovaním existujúcich objektov,
- metódy sú procedurálne abstrakcie, ktoré majú často priamy účinok na objekty, ku ktorým sú viazané.

Jazyky podporujúce túto paradigmu sú napr.: Simula67, SmallTalk, Beta, Self, C++, Java, OCaml a iné.

Logické programovanie

Logické programovanie – kladie dôraz na výber medzi alternatívami a na procedurálnu abstrakciu.

Charakteristiky logického programovania:

- táto paradigma programovania sa úplne odlišuje od predošlých,
- program pozostáva z množiny alternatívnych procedurálnych abstrakcií pre každý identifikátor,
- výber medzi alternatívami sa robí pomocou unifikácie vzorov (*pattern unification*) medzi parametrami abstrakcií a argumentami aktivácií. Táto unifikácia viaže identifikátory k termom. Táto väzba zaniká v prípade, keď zlyhanie zapríčiní návrat späť (*backtracking*).

Základným logickým programovacím jazykom je Prolog.

Formálny popis programovacieho jazyka

Popis (alebo **definícia**) **programovacieho jazyka** sa nazýva formálna, ak je zapísaná v pojmoch a symboloch s presným významom.

Referenčné manuály a príručky programovacích jazykov obsahujú formálny popis syntaxe jazyka, ale popis významu konštrukcií je neformálny, v prirodzenom jazyku. To často vedie k nejednoznačnému pochopeniu a mnohým chybám v programoch.

Naším cieľom je zaviesť vyjadrenie významu konštrukcií jazykov explicitne a jednoznačne použitím **formálnych sémantických metód**.

Formálna definícia jazyka

Formálna definícia programovacieho jazyka pozostáva z:

- formálnej definície syntaxe vyjadrenej:
  - Backusovou-Naurovou formou,
  - induktívnou definíciou,
  - odvodzovacími pravidlami, . . .
- formálnej definície sémantiky jazyka vhodnou sémantickou metódou.

Syntax

Syntax programovacieho jazyka určuje tvar a štruktúru programov písaných v tomto jazyku.

Rozoznávame niekoľko druhov syntaxe:

- konkrétnu syntax,
- abstraktnú syntax,
- regulárnu syntax,
- bezkontextovú syntax,
- kontextovú syntax.

Z hľadiska formálnej definície programovacieho jazyka nás budú zaujímať len **konkrétna** a **abstraktná syntax**.

Konkrétna syntax

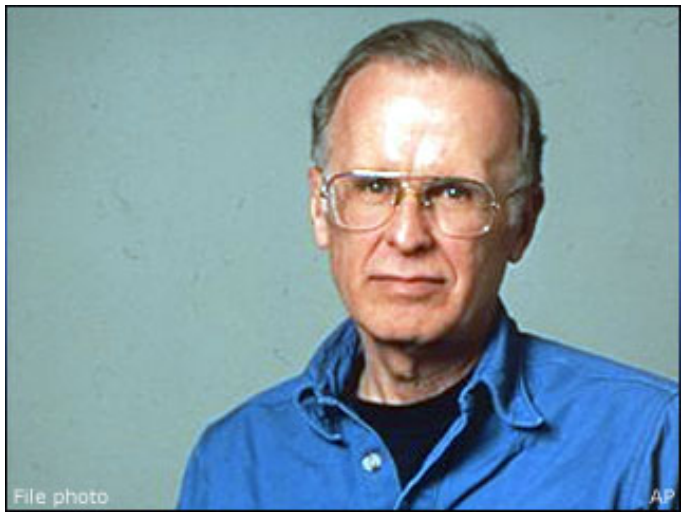
Konkrétna syntax sa zaoberá **textom** programu a slúži pre lexikálnu a syntaktickú kontrolu programov.

Konkrétna syntax:

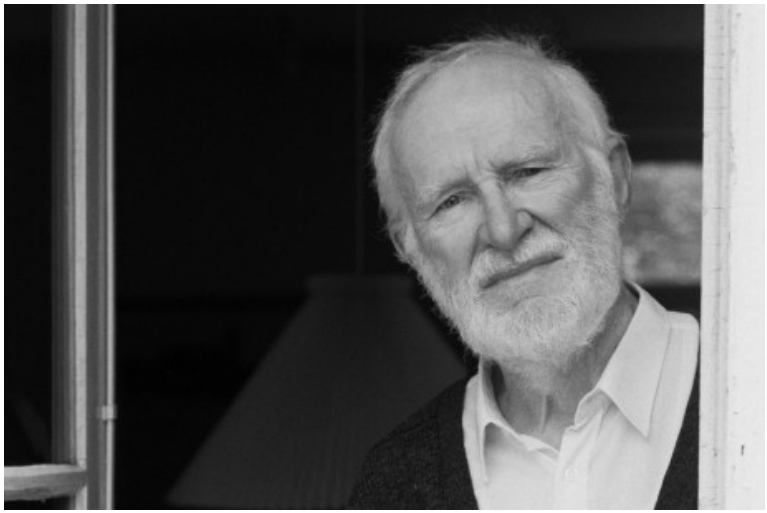
- určuje, či reťazce znakov patria do programovacieho jazyka,
- poskytuje **syntaktický strom** (*parse tree*) pre každý akceptovaný program,
- je špecifikovaná **formálnymi gramatikami** a zapísaná v BNF produkčnými pravidlami pre každý neterminálny symbol.
- Gramatika pre konkrétnu syntax by mala byť jednoznačná.
- Ak je gramatika nejednoznačná, musia byť pridané pravidlá precedencie pre výber jediného syntaktického stromu.

Konkrétna syntax

BNF – Backus-Naur Form – je formálny zápis, pomocou ktorého sa vyjadruje gramatika jazyka v tvare produkčných pravidiel (John Backus, Peter Naur pre Algol60).



John Backus



Peter Naur

Abstraktná syntax

Abstraktná syntax sa zaoberá len **štruktúrou programov** a slúži pre účely formálnej sémantiky.

Kým konkrétna syntax sa zaoberá reálnymi reťazcami znakov, abstraktná syntax sa venuje len štruktúre, ktorú reprezentuje ako **strom abstraktnej syntaxe** (*abstract syntax tree*), **AST**.

Definícia abstraktnej syntaxe pozostáva:

- zo syntaktických oblastí (domén),
- zo stromu abstraktnej syntaxe.

Strom abstraktnej syntaxe je definovaný nasledovne:

- v uzloch stromu sú prvky syntaktických oblastí (konštruktory),
- v listoch sú atomické entity, napr. pravdivostné hodnoty, čísla, niekedy aj operácie,
- z každého uzla vedú hrany ku argumentom konštruktora.

Strom abstraktnej syntaxe môžeme zapísať aj lineárne, v tvare produkčných pravidiel BNF.

Abstraktná syntax

V našom predmete budeme používať nasledujúcu definíciu abstraktnej syntaxe.

Abstraktná syntax pozostáva:

- zo syntaktických oblastí,
- z jedného produkčného pravidla pre každú syntaktickú oblasť.

Každé produkčné pravidlo má tvar:

$$prvok\_syntaktickej\_oblasti ::= tvar\_1 \mid tvar\_2 \mid \ldots$$

Príklad abstraktnej syntaxe

Pre jazyk netypovaných aritmetických výrazov definujeme abstraktnú syntax nasledujúcim spôsobom:

1. Syntaktické oblasti:

$n \in \mathbf{Num}$	reťazce číslic
$x \in \mathbf{Var}$	premenné
$e \in \mathbf{Expr}$	aritmetické výrazy

   - n je atomická entita, pre abstraktnú syntax výrazov nie je dôležitý tvar čísel
   - x je atomická entita, pre abstraktnú syntax výrazov nie je dôležitý tvar identifikátorov

2. Potrebujeme preto len jedno produkčné pravidlo pre syntaktickú oblasť $\mathbf{Expr}$ aritmetických výrazov:

$$e ::= n \mid x \mid e + e \mid e - e \mid e * e \mid (e).$$

Sémantika

Formálna sémantika programovacích jazykov:

- poskytuje abstraktné jednotky, ktoré reprezentujú len podstatné črty všetkých možných vykonávaní programov písaných v danom jazyku,
- ignoruje detaily, ktoré nie sú z hľadiska významu programu podstatné.

Obvykle sa za **podstatné črty** považuje

- vzťah medzi vstupom a výstupom,
- terminovanie alebo neterminovanie vykonávania programu.

Implementačné detaily, napr.

- skutočné adresy v pamäti počítača,
- skutočný čas vykonávania programu,
- architektúra počítača,

sa z hľadiska sémantiky považujú za **nepodstatné** a ignorujú sa.

Sémantické metódy

Poznáme viacero sémantických metód:

- operačná sémantika:
  - prirodzená sémantika,
  - štrukturálna operačná sémantika,
- denotačná sémantika,
- axiomatická sémantika,
- akčná sémantika,
- atribútové gramatiky,
- algebraická sémantika,
- kategorická sémantika,
- sémantika hier.

Použitie formálnych sémantických metód

Všeobecne sa formálne metódy sémantiky používajú:

- pri návrhu programovacieho jazyka,
- pri hľadaní skrytých chýb navrhnutého jazyka,
- pre generovanie prekladačov a interpreterov,
- pre zápis štandardov programovacích jazykov,
- pre stanovenie všeobecných vlastností jazykov,
- pre vyučovanie a používanie programovacích jazykov.

Sémantické metódy

Atribútové gramatiky

- definujú systémy, ktoré systematicky počítajú *metadáta* (atribúty) pre rôzne konštrukcie jazyka,
- súvisia s denotačnou sémantikou, cieľový jazyk je pôvodný jazyk obohatený o anotácie atribútov,
- používa sa pre generovanie kódu v prekladačoch.

Algebraická sémantika definuje model špecifikácie (ADT) ako mnohodruhovú algebru.

Kategorická sémantika používa ako základný formalizmus teóriu kategórií a model definuje ako funktor medzi kategóriami.

Sémantika hier je inšpirovaná teóriou hier, je vhodná najmä pre nedeterministické procesy a rozhodovacie procesy.