

Prednáška 8: Algebraický prístup k špecifikácii softvéru

Metódy a prostriedky špecifikácie 2011/12

Valentino Vranič

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

22. november 2011

Obsah prednášky

- 1 Úvod
- 2 Typy a funkcie
- 3 Axiómy a predpoklady
- 4 Predpoklady
- 5 Dôsledky a invarianty
- 6 Iný príklad: objednávka
- 7 Zásobník v jazyku Z
- 8 K implementácii

Úvod

Problém prešpecifikácie

- Zmena vnútornej štruktúry
- Zapuzdrenie
- Rozhranie – definuje signatúry operácií
- Pod rozhraním si predstavujeme určité správanie
- Ale ani programovacie jazyky, ani UML nám neumožňuje toto správanie vyjadriť abstraktne

Príklad: zásobník

- Zásobník – angl. stack = stoh (napr. sena)
- LIFO – Last In, First Out
- Rôzne reprezentácie (štruktúry), napr.
 - zreťazený zoznam
 - pole
 - postupnosť (v jazyku Z)
- Ktorá je správna?
- Ako definovať zásobník bez určenia jeho reprezentácie?

Definícia zásobníka bez reprezentácie

- Prostredníctvom *operácií* alebo *funkcií* sa vyhneme reprezentácii
 - Po vložení prvku na zásobník bude tento prvok viditeľný ako prvý
 - Po vložení prvku na zásobník bude tento prvok operáciou výberu prvý odstránený
 - Po vytvorení zásobník bude prázdny
- Funkcionálne/aplikatívne vyjadrenie

Abstraktné typy údajov

- Abstract data type (ADT) – abstraktný typ údajov
- Abstrahujeme od štruktúry typu
- Typ definujeme správaním
- Správanie je vyjadrené prostredníctvom funkcií v aplikatívnom (matematickom) zmysle
- Funkcie nemenia žiadne hodnoty – nemajú vedľajšie účinky (side effects)

Štruktúra abstraktného typu údajov

- Typy
- Funkcie
- Axiómy
- Predpoklady

Typy a funkcie

Typy

- Zásobník – príklad vychádza zo špecifikácie Bertranda Meyera¹

Stack[*E*]

¹B. Meyer. Object-Oriented Software Construction. Prentice Hall, 2nd edition, 2000.

Funkcie

$$\text{new} : \text{Stack}[E]$$
$$\text{empty} : \text{Stack}[E] \rightarrow \text{Boolean}$$
$$\text{push} : \text{Stack}[E] \times E \rightarrow \text{Stack}[E]$$
$$\text{pop} : \text{Stack}[E] \rightarrow \text{Stack}[E]$$
$$\text{top} : \text{Stack}[E] \rightarrow E$$

Parciálne a totálne funkcie

- Funkcia je zobrazenie prvkov definičného oboru na prvky oboru hodnôt (relácia), pričom každému prvku definičného oboru zodpovedá najviac jeden prvok oboru hodnôt
- Definičný obor – doména (domain)
- Obor hodnôt – kodoména (range)
- Funkcia je parciálna, ak nie je definovaná pre všetky prvky domény
- Funkcia *top* napr. nie je definovaná pre prázdny zásobník

$$top : Stack[E] \rightarrow E$$

- Funkcia je totálna, ak nie je parciálna
- Príklad: funkcia *empty*

$$empty : Stack[E] \rightarrow Boolean$$

Axiómy a predpoklady

Axiómy

$\forall e : E, s : Stack[E]$

$A1 : top(push(s, e)) = e$

$A2 : pop(push(s, e)) = s$

$A3 : empty(new)$

$A4 : \neg empty(push(s, e))$

Predpoklady

Predpoklady

$pop(s : Stack[E])$ requires $\neg empty(s)$

$top(s : Stack[E])$ requires $\neg empty(s)$

Dôsledky a invarianty

Dôsledky (1)

- Dôsledky funkcií bývajú vyjadrené axiómami, ktoré sledujú vlastnosti výsledku uplatnenia týchto funkcií
- Inak povedané, dôsledok je definovaný uplatnením dopytovej funkcie nad skúmanou funkciou
- Klasifikácia funkcií v ADT (označenie: Meyer – obvyklé):
 - creator function – constructor
 - query function – accessor
 - command function – modifier
- Invarianty nie sú vyjadrené priamo axiómami, lebo súvisia so štruktúrou, a tú v ADT nevyjadrujeme

Dôsledky (2)

$\forall e : E, s : Stack[E]$

$A1 : top(push(s, e)) = e$

$A2 : pop(push(s, e)) = s$

$A3 : empty(new)$

$A4 : \neg empty(push(s, e))$

- A1 – po aplikovaní funkcie `push()`, na vrch zásobníka bude pridaný príslušný prvok
- A2 – po aplikovaní funkcie `pop()`, vrchný prvok už nebude na zásobníku
- A3 – zásobník je po vytvorení prázdny
- A4 – po aplikovaní funkcie `push()`, zásobník nebude prázdny

Iný príklad: objednávka

Požiadavky

- Objednávka tovaru
- Objednávka je po vytvorení prázdna
- Položky je možné pridávať a odoberať
- Po expedovaní už nie je možné objednávku meniť
- Objednávku je možné zrušiť (aj neprázdnu)

Prvý pokus

Typy

Objednavka, Polozka

Funkcie

nova : *Objednavka*

pridajPolozku : *Objednavka* \times *Polozka* \rightarrow *Objednavka*

odoberPolozku : *Objednavka* \times *Polozka* \rightarrow *Objednavka*

expeduj : *Objednavka* \rightarrow *Objednavka*

zrus : *Objednavka* \rightarrow *Empty*

prazdna : *Objednavka* \rightarrow *Boolean*

Axiómy

$\forall p$: *Polozka*, o : *Objednavka*

A1 : *odoberPolozku*(*pridajPolozku*(o , p), p) = o

A2 : *prazdna*(*nova*)

A3 : \neg *prazdna*(*pridajPolozku*(o , p))

Predpoklady

odoberPolozku(o : *Objednavka*, p : *Polozka*) requires \neg *prazdna*(o)

Čo chýba?

- Nevieme kedy je objednávka expedovaná
- Potrebujeme však vyjadriť nemennosť expedovanej objednávky
- Problém by vyriešilo pridanie príslušného príznaku alebo atribútu, ale tento prístup to neumožňuje
- Mohli by sme rozlišovať medzi typmi expedovanej a neexpedovanej objednávky, ale pri väčšom počte atribútov budeme mať explóziu typov
- Aké je teda riešenie?

Atribúty sledujeme pomocou funkcií

- Doplníme funkciu:

expedovana : *Objednavka* \rightarrow *Boolean*

- Upravíme predpoklady:

pridajPolozku(*o* : *Objednavka*, *p* : *Polozka*) requires \neg *expedovana*(*o*)

odoberPolozku(*o* : *Objednavka*, *p* : *Polozka*) requires \neg *prazdna*(*o*) \wedge \neg *expedovana*(*o*)

zrus(*o* : *Objednavka*) requires \neg *expedovana*(*o*)

Zásobník v jazyku Z

Zásobník bez definície štruktúry (1)

- Musíme simulovať typ Boolean, lebo jazyk Z ho nepozná:

$[Stack]$
 $Boolean ::= t \mid f$

Zásobník bez definície štruktúry (2)

- Funkcie reprezentujeme vo forme generickej definície

 $[E]$ $push : Stack \times E \rightarrow Stack$ $pop : Stack \rightarrow Stack$ $top : Stack \rightarrow E$ $empty : Stack \rightarrow Boolean$ $new : Stack$ $\forall s : Stack; e : E \bullet$ $top(push(s, e)) = e$ $\wedge pop(push(s, e)) = s$ $\wedge empty(new) = t$ $\wedge empty(push(s, e)) = f$

Zásobník bez definície štruktúry (3)

- Problém je, že takto nevieme vyjadriť predpoklady, lebo takto vlastne definujeme len konštanty – v tomto zmysle akoby makrá
- Nepoužívame adekvátny prostriedok: schémy
- Funkcie pre nás budú operácie
- Operácie teda vyjadríme schémami
- V schémach vyjadríme predpoklady a dôsledky

Zásobník ako postupnosť (1)

- Ak chceme použiť schémy, musíme predpokladať nejakú štruktúru zásobníka
- Zoberme postupnosť ako vhodnú matematickú abstrakciu (definuje poradie prvkov)

$$\begin{array}{l} \text{Stack}[E] \text{ —————} \\ \text{stack} : \text{seq } E \end{array}$$

- Po inicializácii je zásobník prázdny

$$\begin{array}{l} \text{StackInit}[E] \text{ —————} \\ \text{Stack}'[E] \\ \hline \text{stack}' = \langle \rangle \end{array}$$

- Dôsledok vyjadrený v schéme je uplatnením axiómu

empty(new)

Zásobník ako postupnosť (2)

- Funkcia push

$ \begin{array}{l} \text{StackPush}[E] \\ \Delta\text{Stack}[E] \\ e? : E \end{array} $	$ \text{stack}' = \langle e? \rangle \hat{\ } \text{stack} $
---	--

- Dôsledky vyjadrené v schéme sú uplatnením axiémov

$$\begin{aligned}
 \text{top}(\text{push}(s, e)) &= e \\
 \neg \text{empty}(\text{push}(s, e))
 \end{aligned}$$

Zásobník ako postupnosť (3)

- Funkcia top

$$\text{StackTop}[E]$$
$$\exists \text{Stack}[E]$$
$$e! : E$$
$$\text{stack} \neq \langle \rangle$$
$$e! = \text{head stack}$$

Zásobník ako postupnosť (4)

- Funkcia pop

$$\text{StackPop}[E]$$
$$\Delta\text{Stack}[E]$$
$$\text{stack} \neq \langle \rangle$$
$$\text{stack}' = \text{tail stack}$$

K implementácii

ADT a OOP

- Možno povedať, že trieda je implementácia abstraktného typu údajov
- Presnejšie, trieda predstavuje abstraktný typ údajov s čiastočnou implementáciou
- Najčastejšie nemáme prostriedky na vyjadrenie predpokladov a dôsledkov funkcií
- Úplne bez implementácie: plne abstraktné triedy a rozhrania
- Dedenie – cesta k rôznym verziám ADT

Iné možnosti

- Pre implementáciu ADT nie je nevyhnutné OOP
- Funkcionalita ADT sa dá implementovať aj procedurálne alebo funkcionálne (aplikatívne)
- V závislosti od programovacieho jazyka môžeme byť viac alebo menej obmedzení v možnosti vyjadriť typ ako taký

Sumarizácia

Sumarizácia

- Abstraktné typy údajov ako prostriedok na vyjadrenie správania bez viazania sa na určitú štruktúru
- Možnosti vyjadrenia v jazyku Z – predsa musíme určiť štruktúru aj keď na vyššej úrovni abstrakcie
- Možnosti implementácie – trieda ako implementácia ADT

Literatúra

- Kapitola 6 z knihy
B. Meyer. Object-Oriented Software Construction. Prentice Hall, 2nd edition, 2000.