Formálna špecifikácia softvéru: jazyk Z a algebraická špecifikácia

Poznámky k prednáškam z predmetu Modelovanie softvéru

Valentino Vranić
http://fiit.sk/~vranic/, vranic@stuba.sk
Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

15. november 2016

Obsah

1 Úvod

- Softvérové systémy sa špecifikujú väčšinou neformálnymi prostriedkami
- Takýto prístup je zdrojom chýb
- Zároveň nie je možné formálnym spôsobom overiť vlastnosti špecifikácie
- Podnet pre uplatnenie matematického formalizmu v špecifikovaní softvéru
- Pozrieme sa na jazyk Z ako príklad formálneho prístupu k špecifikácii

2 Jazyk Z

- Špecifikácia operácií prostredníctvom matematického modelu podmienok založeného na teórii množín a predikátovej logike
- Jean-Raymond Abrial, Steve Schuman a Bertrand Meyer, 1977
- *Z*ermelova–Fraenkelova teória množín
- Matematické základy
  - Predikátová logika
  - Teória množín
- Štruktúrovanosť
  - Schémy
  - Typy
- Použitie prirodzeného jazyka
  - Pomenovanie identifikátorov
  - Komentár
- Spresňovanie (refinement)

3 Štruktúra špecifikácie v jazyku Z

Typická organizácia špecifikácie v jazyku Z

1. Typy
2. Stav a inicializácia systému
3. Základné operácie
4. Robustné operácie

Príklad: Systém správy balíkov

- Softvérové balíky (knižnice, pomocné programy a pod.) sú potenciálne spoločné pre viac aplikácií – stačí ich inštalovať iba raz[1]
- Každá aplikácia pri inštalácii pridáva nové balíky
- Pri odinštalovaní aplikácie sa nesmú odinštalovať balíky, ktoré sú používané zo strany iných aplikácií

Typy

- V jazyku Z typ je množina
- Základný je len typ celých čísel $\mathbb{Z}$
- Ostatné typy sa musia definovať
- Daný typ – typ s neznámou štruktúrou (nezaujímavou z hľadiska špecifikácie):

  $[Balik]$

- Typ definovaný vymenovaním:

  $Sprava ::= OK$
  $\mid$ ‘*Vazba uz existuje*‘ $\mid$ ‘*Vazba neexistuje*‘
  $\mid$ ‘*Balik nie je k dispozicii*‘ $\mid$ ‘*Balik uz je k dispozicii*‘
  $\mid$ ‘*Balik nie je nainstalovany*‘ $\mid$ ‘*Balik uz je nainstalovany*‘
  $\mid$ ‘*Vazba sa este pouziva*‘
  $\mid$ ‘*Nie su nainstalovane potrebne baliky*‘
  $\mid$ ‘*Balik sa nemouze odinstalovat*‘

- Typ pomocou potenčnej množíny

  $\mathbb{P}\, Balik$

- *Balik* je množina všetkých balíkov – jeho *potenčná množina* je množina všetkých podmnožín tejto množiny
- Príklad potenčnej množiny (power set):

  $$\mathbb{P}\{1,2,3\} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}\, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

- Štruktúrované typy sa vytvárajú pomocou kartézskeho súčinu:

  $Balik \times Balik$

- Typy možno pomenovať definíciou skratky:

  $Vazba == Balik \times Balik$

- Iný spôsob vytvorenia štruktúrovaného typu je schémou

[1] M. Kiselkov. Systém správy balíkov. Správa k projektu, Metódy a prostriedky špecifkácie 2001/02, FEI STU.

4 Špecifikácia operácií

Schéma

Schémy – prostriedok modularizácie špecifikácie

- Schéma pozostáva z deklarácie premenných a predikátu, ktorý ohraničuje ich hodnoty

$$\begin{array}{|l} \textit{Schema} \\ \hline Deklaracia \\ \hline Predikat \end{array}$$

- Rovnocenné je horizontálne vyjadrenie:

$$Schema \mathrel{\widehat=} [Deklaracia \mid Predikat]$$

Stav a inicializácia systému

$$\begin{array}{|l} \textit{System} \\ \hline DispBalik : \mathbb{P}\, Balik \\ InstBalik : \mathbb{P}\, Balik \\ ZzVazba : \mathbb{P}\, Vazba \\ \hline ZzVazba(\!|\, InstBalik \,|\!) \subseteq InstBalik \end{array}$$

- Stav systému je daný hodnotami premenných, z ktorých pozostáva
- Dvojbodka znamená, že premenná nadobúda hodnoty z danej množiny

$$\begin{array}{|l} \textit{Inicializacia} \\ \hline System' \\ \hline ZzVazba' = \emptyset \\ DispBalik' = \emptyset \\ InstBalik' = \emptyset \end{array}$$

- Apostrof vyjadruje stav po operácii
- Takto môžeme vyjadriť *dôsledok* (postcondition)

Operácie a konvencie Ξ a Δ

- Operácie pracujú nad stavom systému – preto v nich zahŕňame schému stavu systému
- Či operácia mení stav systému alebo nie, určuje spôsob jeho zahrnutia: konvencia Ξ alebo Δ

$\Delta State$

$State$
$State'$

$\Xi State$

$State$
$State'$

$\Theta State = \Theta State'$

Operácie bez zmeny stavu

- Zistenie, ktoré balíky sú potrebné, nemení stav systému
- Vstupné premenné sú označené otáznikom, výstupné výkričníkom

PotrebneBaliky

$\Xi System$
$balik? : Balik$
$potrebne! : \mathbb{P}\, Balik$

$potrebne! = ZzVazba(\!| \{balik?\} |\!) \setminus InstBalik$

- Konvencia Ξ určuje, že premenné zo schémy *System* zachovajú svoje hodnoty aj po operácií, čím sú vyjadrené *invarianty*
- Akákoľvek podmienka stanovená v schéme *System*, ktorá sa nevzťahuje na vstupné ani výstupné premenné, je tiež invariantom
- Ani zistenie, ktoré balíky sú potrebné, ale nie sú k dispozícii (tiež nemení stav systému):

PotrebneBalikyNieKDispozicii

$\Xi System$
$balik? : Balik$
$potrebne! : \mathbb{P}\, Balik$

$potrebne! = (ZzVazba(\!| \{balik?\} |\!) \setminus InstBalik) \setminus DispBalik$

Operácie so zmenou stavu

- Pridanie balíka do množiny balíkov, ktoré sú k dispozícii, stav systému mení
- Pri zmene stavu sa uvádzajú aj *predpoklady* (preconditions) – obsahujú len premenné bez apostrofu

RegistrujBalik

$$\Delta System$$
$$balik? : Balik$$
$$balik? \notin DispBalik$$
$$DispBalik' = DispBalik \cup \{balik?\}$$

- Odober balík z množiny balíkov, ktoré sú k dispozícii

VymazBalik

$$\Delta System$$
$$balik? : Balik$$
$$balik? \in DispBalik$$
$$DispBalik' = DispBalik \setminus \{balik?\}$$

- Pridaj väzbu do množiny väzieb

RegistrujVazbu

$$\Delta System$$
$$vazba? : Vazba$$
$$vazba? \notin ZzVazba$$
$$ZzVazba' = ZzVazba \cup \{vazba?\}$$

- Odober väzbu z množiny väzieb

VymazVazbu

$$\Delta System$$
$$vazba? : Vazba$$
$$vazba? \in ZzVazba$$
$$first\ vazba? \notin DispBalik \land first\ vazba? \notin InstBalik$$
$$ZzVazba' = ZzVazba \setminus \{vazba?\}$$

- Nainštaluj balík – inštalácia sa nesmie vykonať, pokiaľ nie sú nainštalované všetky balíky, od ktorých je balík závislý

Instalacia

$$\Delta System$$
$$balik? : Balik$$
$$balik? \notin InstBalik$$
$$balik? \in DispBalik$$
$$ZzVazba(\!| InstBalik \cup \{balik?\} |\!) \setminus InstBalik = \emptyset$$
$$InstBalik' = InstBalik \cup \{balik?\}$$

- Odinštaluj balík – balík sa nesmie odinštalovať, pokiaľ je nainštalovaný iný od neho závislý balík

Odinstalovanie

$\Delta System$
$balik? : Balik$

$balik? \in InstBalik$
$\text{dom}(ZzVazba \rhd \{balik?\}) \cap InstBalik = \emptyset$
$InstBalik' = InstBalik \setminus \{balik?\}$

Predpoklady, dôsledky a invarianty

- Predpoklady sa vyjadrujú podmienkami
  - uvedenými v súvislosti so schémami zahrnutými podľa konvencie Δ, pričom tieto podmienky obsahujú nedekorované premenné (bez apostrofu, otáznika a výkričníka)
  - ktoré sa vzťahujú na vstupné premenné a neobsahujú výstupné premenné ani premenné s apostrofom
- Dôsledky sa vyjadrujú podmienkami, ktoré sa vzťahujú na premenné s apostrofom a/alebo na výstupné premenné
- Invarianty sa vyjadrujú podmienkami uvedenými v súvislosti so schémami zahrnutými podľa konvencie Ξ, pričom tieto podmienky neobsahujú vstupné ani výstupné premenné

5 Spájanie operácií

Spájanie operácií

- Uvedené operácie nedávajú používateľovi najavo nič o výsledku
- V jazyku Z výstup modelujeme pomocou výstupných premenných
- Tieto operácie sú určené na spájanie so základnými operáciami

Operácie zabezpečujúce hlásenia

- Niektoré operácie zabezpečujúce hlásenia sú úplne generické

Uspech

$sp! : Sprava$

$sp! = \text{`}OK\text{`}$

- Kontrola či väzba už existuje alebo neexistuje

VazbaExistuje

$\Xi System$
$vazba? : Vazba$
$sp! : Sprava$

$vazba? \in ZzVazba$
$sp! = \text{`}Vazba\ uz\ existuje\text{`}$

VazbaNeexistuje

$\Xi System$
$vazba? : Vazba$
$sp! : Sprava$

$vazba? \notin ZzVazba$
$sp! = \text{`}Vazba\ neexistuje\text{`}$

- Kontrola, či je balík registrovaný (je v množine balíkov ktoré sú k dispozícií) alebo nie je registrovaný

BalikJeReg

$\Xi System$
$balik? : Balik$
$sp! : Sprava$

$balik? \in DispBalik$
$sp! = \text{`}Balik\ uz\ je\ k\ dispozicii\text{`}$

BalikNieJeReg

$\Xi System$
$balik? : Balik$
$sp! : Sprava$

$balik? \notin DispBalik$
$sp! = \text{`}Balik\ nie\ je\ k\ dispozicii\text{`}$

- Kontrola, či je balík nainštalovaný, resp. nie je nainštalovaný

BalikJeInst

$\Xi System$
$balik? : Balik$
$sp! : Sprava$

$balik? \in InstBalik$
$sp! = \text{`}Balik\ uz\ je\ nainstalovany\text{`}$

BalikNieJeInst

$\Xi System$
$balik? : Balik$
$sp! : Sprava$

$balik? \notin InstBalik$
$sp! = \text{`}Balik\ nie\ je\ nainstalovany\text{`}$

- Kontrola, či sa väzba ešte používa

VazbaSaPouziva

$\Xi System$
$vazba? : Vazba$
$sp! : Sprava$

$first\ vazba? \in DispBalik \lor first\ vazba? \in InstBalik$
$sp! = \text{`}Vazba\ sa\ este\ pouziva\text{`}$

- Kontrola, či sú nainštalované všetky potrebné balíky

ChybajuBaliky

$\Xi System$
$balik? : Balik$
$sp! : Sprava$

$ZzVazba(\!| InstBalik \cup \{balik?\} |\!) \setminus InstBalik \neq \emptyset$
$sp! = \text{`}Nie\ su\ nainstalovane\ potrebne\ baliky\text{`}$

- Kontrola, či sa balík môže odinštalovať

BalikSaPouziva

$\Xi System$
$balik? : Balik$
$sp! : Sprava$

$\mathrm{dom}(ZzVazba \rhd \{balik?\}) \cap InstBalik \neq \emptyset$
$sp! = \text{`}Balik\ sa\ nemouze\ odinstalovat\text{`}$

Spájanie špecifikácií operácií

- Špecifikácie operácií môžeme spájať logickými spojkami
- Výsledná schéma obsahuje všetky premenné v deklaračnej časti
- Predikáty sa spoja príslušnou logickou spojkou
- Toto je spôsob *oddelenia záležitostí* (separation of concerns)
  - Hlavná logika operácie je prezentovaná v samostatnej schéme
  - Každá ďalšia záležitosť, ako napr. výpisy a ošetrenia chybových stavov, je tiež prezentovaná v samostatnej schéme
  - Spojenie týchto záležitostí je zabezpečené spojením schém príslušnými logickými spojkami (samostatnou konštrukciou)
- Súvisí to s aspektovo-orientovaným prístupom

Robustné operácie

$RegistrujBalikRO \mathrel{\widehat{=}} (RegistrujBalik \land Uspech) \lor BalikJeReg$

$VymazBalikRO \mathrel{\widehat{=}} (VymazBalik \land Uspech) \lor BalikNieJeReg$

$RegistrujVazbuRO \mathrel{\widehat{=}} (RegistrujVazbu \land Uspech) \lor VazbaExistuje$

$VymazVazbuRO \mathrel{\widehat{=}} (VymazVazbu \land Uspech) \lor VazbaNeexistuje$

$PotrebneBalikyRO \mathrel{\widehat{=}} PotrebneBaliky \land Uspech$

$PotrebneBalikyNieKDispoziciiRO \mathrel{\widehat{=}}$
$PotrebneBalikyNieKDispozicii \land Uspech$

$InstalaciaRO \mathrel{\widehat{=}} (Instalacia \land Uspech) \lor BalikJeInst$
$\lor BalikNieJeReg \lor ChybajuBaliky$

$OdinstalovanieRO \mathrel{\widehat{=}} (Odinstalovanie \land Uspech)$
$\lor BalikNieJeInst \lor BalikSaPouziva$

6 Uplatnenie jazyka Z

Ďalšie mechanizmy jazyka Z

- V prednáške bol prezentovaný len zlomok mechanizmov jazyka Z
- Matematické dokazovanie vlastností špecifikácie
- Plné využitie predikátovej logiky
- Definovanie objektov generickou skratkou, axiomatickou definíciou a generickou definíciou
- Relácie a funkcie
- Postupnosti
- Multimnožiny
- Voľné typy
- Schéma ako typ
- Kompozícia schém

Nástroje na prácu v jazyku Z

- Jazyk Z od vzniku bol spojený s LaTeXom, čo umožnilo automatizovanú prácu so špecifikáciou v jazyku Z
- LaTeX štýl pre jazyk Z – `z-eves.sty`, upravená verzia `zed-csp.sty`
- LaTeX syntax pre jazyk Z: `http://staffwww.dcs.shef.ac.uk/people/A.Simons/z2sal/zdocs/Z%20of%20ZeTa.pdf`
- Nástroj Z/EVES na tvorbu a verifikáciu špecifikácií v jazyku Z
  - `http://fmt.cs.utwente.nl/courses/fmse/`
  - `http://www.uni-koblenz.de/~winter/Lehre/SS01/ZEves/ZEves.html`

Známe prípady uplatnenia jazyka Z

- Jazyk Z a formálne metódy určite neprevládajú, ale predsa sa používali a používajú
- CICS – Customer Information Control System, IBM
- D. Craigen et al. Industrial applications of formal methods to model, design and analyze computer systems: an international survey. Noyes Publications, 1995.

Implementácia špecifikácie

- Formalizmus jazyka Z pomáha zabezpečiť konzistentnosť špecifikácie
- Chyby však môžu vzniknúť pri jej implementácii
- Vzdialenosť jazyka Z od implementácie tomu žiaľ napomáha
- Metóda B je príklad formálneho prístupu bližšieho implementácii

7 Algebraická špecifikácia

Problém prešpecifikácie

- Pri špecifikácii máme tendenciu sa viazať na určitú reprezentáciu
- Ak napríklad pre položky, s ktorými pracujeme, nie je príznačné poradie a prvky sa vyskytujú iba raz, dá sa povedať, že tvoria množinu; povedať, že je to postupnosť je prešpecifikácia
- Príliš veľa detailov spôsobuje viac problémov než príliš málo detailov
- 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

8 Typy a funkcie

Typy

- Zásobník – príklad vychádza zo špecifikácie Bertranda Meyera[2]

$$Stack[E]$$

Funkcie

$$new : Stack[E]$$
$$empty : Stack[E] \to Boolean$$
$$push : Stack[E] \times E \to Stack[E]$$
$$pop : Stack[E] \nrightarrow Stack[E]$$
$$top : Stack[E] \nrightarrow E$$

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

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

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

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

9 Axiómy a predpoklady

Axiómy

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

$$A1 : top(push(s, e)) = e$$
$$A2 : pop(push(s, e)) = s$$
$$A3 : empty(new)$$
$$A4 : \neg\, empty(push(s, e))$$

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

Predpoklady

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

10 Dôsledky a invarianty

- 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

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

$A1 : top(push(s, e)) = e$
$A2 : pop(push(s, e)) = s$
$A3 : empty(new)$
$A4 : \neg\ empty(push(s, e))$

- 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

11 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

$nova : Objednavka$
$pridajPolozku : Objednavka \times Polozka \to Objednavka$
$odoberPolozku : Objednavka \times Polozka \to Objednavka$
$expeduj : Objednavka \to Objednavka$
$zrus : Objednavka \to Empty$
$prazdna : Objednavka \to Boolean$

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 $\neg\ prazdna(o)$
$expeduj(o : Objednavka)$ 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:

  $expedovana : Objednavka \to Boolean$

- Upravíme a rozšírime predpoklady:

  $odoberPolozku(o : Objednavka, p : Polozka)$ requires $\neg\ prazdna(o) \wedge \neg\ expedovana(o)$
  $expeduj(o : Objednavka)$ requires $\neg\ prazdna(o) \wedge \neg\ expedovana(o)$
  $zrus(o : Objednavka)$ requires $\neg\ expedovana(o)$
  $pridajPolozku(o : Objednavka, p : Polozka)$ requires $\neg\ expedovana(o)$

- Identifikujeme ďalšiu axiomu v súvislosti s expedovaním:

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

$A4 : expedovana(expeduj(o))$

12 Zásobník v jazyku Z

Zásobník bez definície štruktúry

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

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

- Funkcie reprezentujeme vo forme generickej definície

$[E]$

$$push : Stack \times E \to Stack$$
$$pop : Stack \nrightarrow Stack$$
$$top : Stack \nrightarrow E$$
$$empty : Stack \to Boolean$$
$$new : Stack$$

$$\forall s : Stack;\ e : E \bullet$$
$$top(push(s, e)) = e$$
$$\land\ pop(push(s, e)) = s$$
$$\land\ empty(new) = t$$
$$\land\ empty(push(s, e)) = f$$

- Problém je, že týmto spôsobom nemôžeme vyjadriť predpoklady, lebo takto vlastne definujeme len konštanty – akoby makrá
- Nepoužívame adekvátny prostriedok: schémy
- Funkcie pre nás budú operácie – operácie teda vyjadríme schémami
- Predpoklady a dôsledky potom budú vystupovať v schémach operácií – ako obvykle

Zásobník ako postupnosť

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

Stack$[E]$

$$stack : \mathrm{seq}\ E$$

- Po inicializácii je zásobník prázdny

StackInit$[E]$

$$Stack'[E]$$

$$stack' = \langle\ \rangle$$

- Dôsledok vyjadrený v schéme je uplatnením axiómy

$$empty(new)$$

- Funkcia push

StackPush$[E]$

$$\Delta Stack[E]$$
$$e? : E$$

$$stack' = \langle e? \rangle \frown stack$$

- Dôsledky vyjadrené v schéme sú uplatnením axióm

$$top(push(s, e)) = e$$
$$\neg\ empty(push(s, e))$$

- Funkcia top

$$
\begin{array}{|l}
\hline
StackTop[E] \\
\Xi Stack[E] \\
e! : E \\
\hline
stack \neq \langle\,\rangle \\
e! = head\ stack \\
\hline
\end{array}
$$

- Funkcia pop

$$
\begin{array}{|l}
\hline
StackPop[E] \\
\Delta Stack[E] \\
\hline
stack \neq \langle\,\rangle \\
stack' = tail\ stack \\
\hline
\end{array}
$$

13 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 možnostiach vyjadriť typ ako taký

14 Sumarizácia

- Jazyk Z ako príklad prostriedku na vyjadrenie formálnej špecifikácie
- Umožňuje štruktúrované vyjadrenie špecifikácie matematickým formalizmom
- Je možné formálne – matematickým dôkazom – overovať vlastnosti špecifikácie
- Operácie sú špecifikované prostredníctvom invariantov, predpokladov a dôsledkov – toto preniklo aj do programovania
- Podpora oddelenia záležitostí – aj pretínajúcich: výsledná operácia sa získa spojením parciálnych operácií
- Jestvujú aj ďalšie formálne prístupy, ktoré však boli ovplyvnené jazykom Z:
  - Object-Z
  - Metóda B (B-Method)
  - Alloy
- 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