Stromové štruktúry v logike

Šimon Horvát

3Ib, 2016 - 2017

Abstrakt. Práca sa zaoberá implementáciou aritmetického výrazu ale aj formálneho dôkazu, ktorý má logike stromovú štruktúru, v jazyku Java. Vzniknutá dátová štruktúra má za cieľ prijatý textový reťazec spracovať a následne vytvoriť z reťazca strom, ktorého každý uzol má požadované vlastnosti, charakteristiky. Implementácia umožní generovanie stromových výrazov do syntaxe TeX-u

Kľúčové slová: výraz, symbol, premenne, stromové štruktúry, TeX, Java.

1 Induktívne štruktúry okolo nás

V každej induktívnej štruktúre sú potrebné ako základné prvky, tak konštruktory. Ak by sme nemali základne stavebné jednotky, darmo by sme ovládali metódy konštrukcie, nič by sa nám zostrojiť nepodarilo. Rovnako bezradní by sme boli, keby sme síce základne prvky mali, ale nepoznali by sme konštruktory.

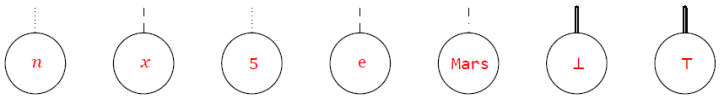
Pekné príklady induktívnych štruktúr poskytuje sama príroda, a to ako život jedinca (za základné prvky môžeme považovať dve rodičovské bunky a za konštruktory okrem iného predpisy zakódovanie v jeho DNA), tak život vo všeobecnosti (základnými prvkami sú prvé živé bunky a konštruktormi napríklad princípy evolučnej teórie), ale i svet ako celok (základným prvkom je situácia po veľkom tresku s dobre nastavenými konštantami, konštruktormi sú prírodné zákony).

1.1 Prostá induktívna štruktúra

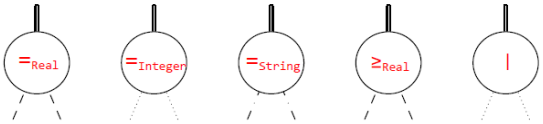
Každý prvok induktu môžeme vyjadriť aj vo forme stromu, t. j. konkrétneho grafického znázornenia orientovaného grafu, ktorého jeden vrchol – koreň – nemá žiadneho predka (t. j. ma vnútorný stupeň 0) a všetky ostatné majú práve jedného predka (t. j. majú vnútorný stupeň 1). V každom liste takéhoto stromu je základný prvok (tento uzol nemá potomkov), v uzle s aspoň jedným potomkom je konštruktor (jeho potomkovia sú (v príslušnom poradí) koreňmi stromov argumentov konštruktora).
Orientáciu pritom nebudeme vyjadrovať šípkami, ale polohou – koreň bude hore a vetvenie smerom nadol (horizontálne posuny sú povolené).

1.2 Konštrukcia potrebných prvkov logického jazyka

Každý symbol či premennú c a jeho charakteristiky ins(c) (skratka pre „**in**puts“ – vstupy) a out(c) (skratka pre „**out**puts“ – výstupy) môžeme znázorniť aj graficky. Dátové typy budú znázornené čiarami, pričom každému z nich bude prislúchať iný druh čiarky (napríklad **Boolean** bude dvojitá, **Integer** bodkovaná, **Real** čiarkovaná, **String** bodkočiarkovaná). Každý symbol či premenná bude mať podobu
akéhosi povedzme okrúhleho uzla obsahujúceho c, z ktorého vychádzajú akési synapsie – smerom nahor jedna čiara zodpovedajúca typu out(c) a smerom nadol zľava doprava čiary, ktoré zodpovedajú tici ins(c). Dostávame tak napríklad takéto reprezentácie príslušných symbolov:



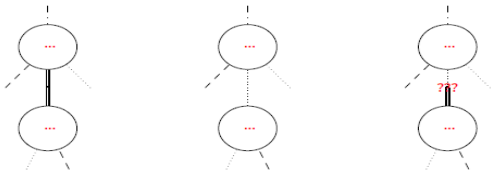
obr.1. Premenné a konštantové symboly nemajú žiadne synapsie smerom nadol.



obr.2. Pri reláciových symboloch je naopak synapsia nahor dvojitá.

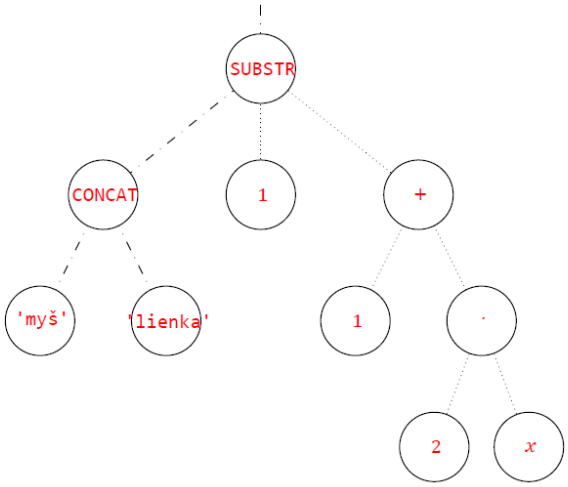
1.2 Výraz

Keď už teda máme k dispozícii logickú abecedu tvorenú symbolmi a premennými a nad každým z jej znakov máme typovú kontrolu jeho výstupu a prípadných vstupov. Vzniká teda prirodzená otázka, ako z týchto znakov vytvoriť „slová“, ktorými by sme dokázali zachytiť časti matematickej reality. Spomeňme si, že každý symbol vieme vyjadriť graficky vo forme uzlu a z neho vychádzajúcich synapsií.
Je asi prirodzené, že ak má k prepojeniu uzlov naozaj prísť, príslušné z nich vedúce synapsie sa zlúčia do jednej, pričom jedna ide zdola (je teda výstupom nižšieho uzla) a druhá zdola (je vstupom vyššieho). Aby sa tak
naozaj mohlo stať, zlučované synapsie musia byť rovnakého typu:



Napríklad prepojenie na prvom i druhom obrázku je v poriadku, pretože sú obe rovnaké (v prvom prípade dvojité, v druhom bodkované) na treťom však k nemu nedôjde, lebo jedna synapsia je bodkovaná a druhá dvojitá. Uvedomme si, že takýmto prepájaním nikdy neprepojíme všetky synapsie – výstupná synapsia najvyššieho uzla ostane vždy nezapojená. Všetky ostatné synapsie však zapojené byť môžu. Ak sa to podarí, vznikne ohodnotený strom, ktorý v tomto kontexte nazývame *výraz*. V každom z nich bude jeho jediná neobsadená synapsia vychádzať z jeho koreňa a bude určovať jeho typ.

Príkladom výrazu je nasledujúci ohodnotený strom. Každá jeho nadol smerujúca synapsia každého jeho uzla je správne prepojená s nahor smerujúcou synapsiou príslušného jeho dieťaťa:



2 Implementácia entít v jazyku Java

Hlavnou entitou v mojej práci je nepochybne trieda *Výraz*. Ale rovnako ako v predchádzajúcej kapitole bolo potrebne skonštruovať potrebne prvky logického jazyka, je rozumne komplexnú triedu Výraz rozobrať na menšie stavebné dieliky, čo znamená identifikovať menšie, menej komplexnejšie podentity.

2.1 Znak, Symbol a Typ ako podentity

Primárnou úlohou entity *Výraz* je premena textového reťazca (aritmetický výraz) na stromovú štruktúru, definovanú v bode **1.2**. A však, to nie je také jednoduché, keďže sa jedná o obyčajný *String*, ktorý v sebe neobsahuje žiadne informácie o v ňom použitých znakoch. No konštrukcia našej žiadanej štruktúry si žiada poznať o znaku napríklad jeho dátový typ, druh symbolu, aritu, ins a out a podobne.

O priradenie informácii k pravé načítanému znaku sa mi postará trieda *Znak.*

```
public class Znak {

      Informacia informacia;
      DatabazaZnakov db = new DatabazaZnakov();
      private List<Znak> bva = null;


      String oznacenie;

      public Znak(String znak) {
            this.oznacenie = znak;
            if(znak.length() != 0) this.priradInfo();
      }

...

  }
```

Samozrejme, informácia o znaku sa k načítanému znaku nepridá náhodne. Samotnému priradeniu, predchádza iterácia databázou, ktorá obsahuje informácie o všetkých *„stavebných prvkoch“* , o ktorých predpokladáme, že sa môžu v reťazci nachádzať. Následné sa porovnávaním vyberie prislúchajúci záznam z databázy a z nášho predtým „nič netušiaceho“ stringu sa stáva objekt triedy *Znak*, z ktorým možno ďalej pracovať.

Takto napríklad vyzerá záznam z databázy, pre symbol + :

```
  Informacia plus = new Informacia("+", ari:2, Ins(2,realReal),
out: Typ.REAL, Symbol.FUNK_SYM, bva: null);
```

Ako bolo spomenuté, objekt *Informacia* obsahuje zložku *Typ*. Je to dôležitá informácia, využívaná v samotnom procese konštruovania stromovej štruktúry

reprezentujúcej výraz. Ako sme už spomínali, zlučované synapsie musia byť rovnakého typu, aby vytvorený strom prešiel dátovou kontrolou.

Trieda *Typ* vyzerá takto:

```
public class Typ {

    private String oznacenie;

    public static final Typ INTEGER = new Typ("Integer");
    public static final Typ REAL = new Typ("Real");
    public static final Typ CHAR = new Typ("Char");
    public static final Typ STRING = new Typ("String");
    public static final Typ BOOLEAN = new Typ("Boolean");

...

}
```

2.2 entita Výraz

Keď už teda máme prvky(objekty triedy *Znak*) a konštruktory(predpisy, zákonitosti syntaxe aritmetického výrazu), nič nám nebráni pri konštrukcii našej induktívnej štruktúry.

```
public class Vyraz extends Znak {

    String znak;

    private Vyraz lavy;
    private Vyraz pravy;

  List<Vyraz> deti = new ArrayList<>();

....

}
```

Trieda *Vyraz* má stromovú štruktúru a teda algoritmy na konštrukciu výrazu majú rekurzívny charakter.

Keď už je proces konštrukcie hotový, a teda z textové reťazca je strom reprezentujúci výraz, ktorému už Java rozumie a ktorý má veľa vlastnosti, na ktoré sa vytvoreného objektu môžem pýtať, nič nebráni tomu aby sme z objektu výraz generovali textové

reťazce v ľubovoľnej podobe, do ľubovoľnej syntaxe, v ľubovoľnej postupnosti (preorder, inorder...). Stačí aby som namiesto postupnosti generovania reťazca vymenil poradie príkazov `aktulnyUzol.spracujSa, spracujLavy, spracujPravy,` a dostanem reťazce s rozličnými vlažnosťami. Alebo stačí aby som namiesto označenia uzla „/“ generoval reťazec „*\frac*“ a syntax vygenerovaného reťazca pre vyraz môže byť kľudne TeX-ovská.

3 Záver

3.1 Čo je hotové

Hotová je kompletná implementácia triedy *Vyraz*, ktorá je schopná načítať aritmeticky výraz, či logickú formulu, ktorú následne vie previesť do stromovej štruktúry. Objekt triedy *Vyraz,* teda vybudovaný strom sa vie typovo skontrolovať a ohodnotiť (počítanie rovníc, vyhodnotenie matematického výroku). Objekt vie ďalej generovať podľa požiadavky s vyššie popísanými vlastnosťami.

3.2 Na čom sa pracuje

Pracujem na entite dôkaz, ktorá by mala byť schopná priblížiť sa ku konštrukcii formálneho dôkazu pomocou odvodzovaniach pravidiel.
Zatiaľ program vie povedať, či je alebo nie je skúmaná formula axiómou.

Poďakovanie. Ďakujem vedúcemu mojej bakalárskej práce, doc. RNDr. Stanislavovi Krajčimu, PhD., za pomoc a množstvo inšpiratívnych myšlienok pri jej tvorbe.

Literatúra

1. http://ics.upjs.sk/~krajci/skola/vyucba/ucebneTexty/logika-stromy.pdf.

2. Goldstern Martin and Judah Haim, The incompleteness phenomenon, A new course in mathematical logic, A K Peters, Wellesley, Mass., 1995, xiii + 247 pp.