Vilniaus universitetas
Matematikos ir informatikos fakultetas
Programų sistemų katedra

Tiesioginio išvedimo produkcijų sistemoje demonstracinė sistema. Java kalba

Autorius: Dainius Jocas

Vilnius
2011

Turinys

Įvadas

Šio laboratorinio darbo užduotis yra suprogramuoti tiesioginį išvedimą produkcijų sistemoje Java programavimo kalba. Pateiksiu jo algoritmo pseudo kodą ir algoritmo realizaciją Java programavimo kalba. Taip pat pateiksiu tris tiesioginio išvedimo pavyzdžius, pademonstruosiu jų išvedimus ir nupiešiu semantinius medžius.

Tiesioginio išvedimo (Forward-Chaining) algoritmo pseudo kodas anglų kalba:

```
1.  repeat
2.     for every rule do
3.        if antecedents match assertions in the working memory and
          consequents would change the working memory then
4.           create triggered rule instance
5.           throw away triggered rule from list of rules
6.           break for loop
7.        end if
8.     end for
9.  until no change in working memory or no STOP signal
```

Tiesioginio išvedimo suprogramuoto algoritmo žodinis paaiškinimas

1. Dėl viso pikto pasitikriname ar neturime tikslo tarp pradinių faktų.
2. Suksime ciklą tol, kol rasime tiesioginio išvedimo sistemą arba matysime, jog jos rasti nepavyks;
3. Iš eilės tikriname visas turimas taisykles taip, kaip aprašyta sekančiuose žingsniuose:
4. Tikriname ar sąraše yra taisyklė, dėl kurios galėtume papildyti turimų faktų sąrašą. Jei radome tinkančią taisyklę:

1. Įrašome taisyklės ID į produkcijų sąrašą;
2. Išmetame taisyklę iš turimų taisyklių sąrašo;
3. Nutraukiame ciklą tam, kad patikrintume ar jau pasiekėme tikslą.

5. Išvedame programos einamąją būseną į terminalą.
6. Tikriname ciklo pabaigos sąlygą.

Algoritmo realizacija Java programavimo kalba

```
/**
 * This class is for building production system. The main method of the
 * class is doForwardChaining, which implement Forward Chaining principe.
 * @author Dainius Jocas, VU MIF, PS#3, 3rd year
 */
public class DerivationMachine {
    /* Data structures used in application */
    ListOfImplications listOfImplications;
    Facts facts;
    ArrayList <String> productionSystem;
    private String goal;

    /**
     * Constructor which loads data structures of the class.
     * @param listOfImplications object of ListOfImplications
     * @param facts list of facts
     * @param goal
     */
    public DerivationMachine(ListOfImplications listOfImplications, Facts facts,
            String goal) {
        this.listOfImplications = listOfImplications;
        this.facts = facts;
        this.goal = goal;
        this.productionSystem = new ArrayList();
    }

    /**
     * This method implements forward chaining rule.
     * @return true if goal was reached, otherwise false
     */
        public boolean doForwardChaining() {
        boolean changed = false;
        boolean goalReached = false; /* if true - the work is done */
        int step_index = 0;       /* counter of steps taken */
        if (!(goalReached = isGoalReached()) == true) {
            do {                                                          /* 1 */
                changed = false;
                for (Object im : this.listOfImplications.                 /* 2 */
                        getListOfImplications()) {
                  Implication imp = (Implication)im;
                  if (!(this.productionSystem.contains(imp.getDescriptor()))/* 3 */
                          && (isMemoryToBeChanged(imp))) {
                        this.facts.addFact(imp.getConsequence());
                        this.productionSystem.add(imp.getDescriptor());     /* 4 */
                        this.listOfImplications.removeImplication(imp);     /* 5 */
                        changed = true;
                        goalReached = isGoalReached();
```

```
                        break;                                                  /* 6 */
                    }                                                           /* 7 */
                }                                                               /* 8 */
                if (changed) {                                  /* log of application*/
                    showStepInfo(++step_index, this.facts.getListOfFacts(),
                        getCurrentProductionSystem());
                }
            } while ((changed == true) && (goalReached == false));              /* 9 */
        } else {
            this.productionSystem.add("No productions used");
        }
        return goalReached;
    }
}
```

Tekstinio pradinių duomenų failo struktūros reikalavimai

1. Failo pradžioje ir pabaigoje esančios tuščios arba komentarų eilutės yra ignoruojamos.
2. Terminaliniai žodeliai ({Productions, Facts, Goal}) yra rašomi lotyniškais rašmenimis ir yra nejautrūs registrui.
3. Tarp terminalinių žodelių ir duomenų gali būti tuščių eilučių.
4. Po duomenų sąrašo privalo būti palikta tuščia eilutė.
5. Simbolis ‘#’ yra komentaro simbolis ir viskas, kas eilutėje yra už šio simbolio nėra laikoma duomenimis programai ir programa niekaip jų neinterpretuoja.
6. Produkcijų sąraše rodykle yra laikoma dviejų simbolių seka ‘->’ išskirta tarpais, pvz. “A, B -> C”.
7. Faktai turi būti atskirti kablelio ir tarpo seka, pvz. “A, B, C”.
8. Pradinių duomenų tvarka nėra svarbi, pvz. tikslas “Goal” gali būti nurodytas ir failo pradžioje ir pabaigoje.
9. Viena produkcija vienoje eilutėje.

Tekstinio pradinių duomenų failo pavyzdys

1. # Author data
2. Productions:
3. # empty line
4. R1: A -> A
5. R2: B -> B
6.
7. Facts:
8. A, B
9.
10.
11. Goal:
12. A
13. #empty line

Pradiniai duomenys iš XML failo

Kadangi duomenys tarp programų neratai būna bendri, todėl reikia ir kažkokio bendro duomenų saugojimo primcipo. Pasaulinės tendencijos rodo, kad tas principas šiandien yra XML failai. Todėl ir

mūsų maža aplikacija turi galimybę pasikrauti duomenis iš XML failo. Reikalavimas XML failui yra vienas – jis turi atitikti šią DTD schemą:

```
<?xml version='1.0' encoding='UTF-8'?>
<!--
    An example how to use this DTD from your XML document:
    <?xml version="1.0"?>
    <!DOCTYPE data SYSTEM "[name_of_dtd_file].dtd">
    <data>
    ...
    </data>
-->
<!ELEMENT data (goal|facts|implications)*>
<!ELEMENT implications (implication)*>
<!ELEMENT implication (consequent|antecedent)*>
<!ATTLIST implication
    id CDATA #IMPLIED
  >
<!ELEMENT antecedent (#PCDATA)>
<!ELEMENT consequent (#PCDATA)>
<!ELEMENT facts (fact)*>
<!ELEMENT fact (#PCDATA)>
<!ELEMENT goal (#PCDATA)>
```

Pradinių duomenų failo pavyzdys:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : 1_simple.xml
    Created on : March 13, 2011, 11:43 AM
    Author     : Dainius Jocas, VU MIF, PS#3, 3rd years
    Description:
        Data input file for first excersice of the "Artificial Inteligence"
        course. In this file we will place simple example of input data for
        derivation.
-->

<!DOCTYPE data SYSTEM "InputDescription.dtd">
<data>
    <implications>
        <implication id="R1">
            <antecedent>A</antecedent>
            <consequent>B</consequent>
        </implication>
        <implication id="R2">
            <antecedent>B</antecedent>
            <consequent>Z</consequent>
        </implication>
    </implications>
    <facts>
        <fact>A</fact>
    </facts>
    <goal>Z</goal>
</data>
```

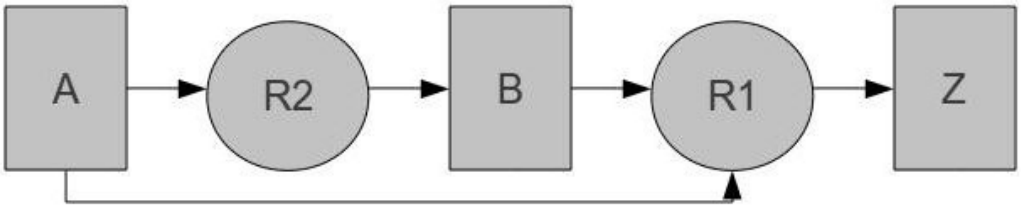
Kaip duomenų failą nurodžius failą su .xml išplėtimu, programa automatiškai prisitaikys prie failo formato.

Pavyzdys #1

Šio pavyzdžio tikslas patikrinti ar programa vykdo algoritmą pagal seką apibrėžtą reikalavimuose.

Failo turinys	Programos rezultatas

Productions:
r1: A, B -> Z
r2: A -> B
r3: B -> C
r4: A, C -> Z
 	
Facts:
A
 	
Goal:
Z
	`Program starts to work:`
	
	`Data for the program has been taken from src/InputData/input2.txt file.`
	
	`Initial data:`
	`  List of facts: A`
	`  Goal: Z`
	`  Implications:`
	`  r1: A, B -> Z`
	`  r2: A -> B`
	`  r3: B -> C`
	`  r4: A, C -> Z`
	
	`After step nr: 1`
	`  List of facts is: A, B`
	`  Implications used are: {r2}`
	
	`After step nr: 2`
	`  List of facts is: A, B, Z`
	`  Implications used are: {r2; r1}`
	
	`GOAL IS REACHED!`
	`Final list of facts is: A, B, Z`
	
	`End of the program!`



Paveikslas 1: Pirmo pavyzdžio semantinis grafas

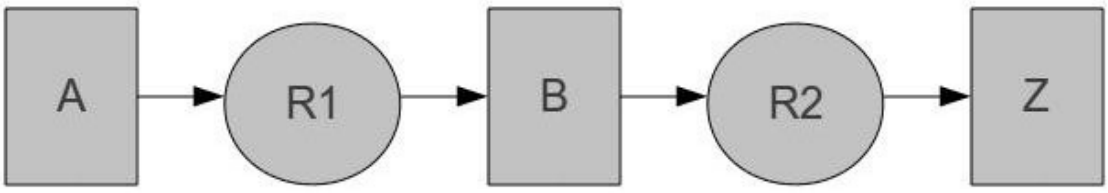
Matome, kad programa vykdo algoritmą taisyklingai.

Pavyzdys #2

Šio pavyzdžio tikslas patikrinti ar programa sugeba gauti teisingą rezultatą su labai paprastais duomenimis.

Failo turinys	Programos rezultatas

Productions:
R1: A -> B
R2: B -> Z
 	
Facts:
A
 	
Goal:
Z
	`Program starts to work:`
	
	`Data for the program has been taken from src/InputData/input1.txt file.`
	
	`Initial data:`
	`  List of facts: A`
	`  Goal: Z`
	`  Implications:`
	`  R1: A -> B`
	`  R2: B -> Z`
	
	`After step nr: 1`
	`  List of facts is: A, B`
	`  Implications used are: {R1}`
	
	`After step nr: 2`
	`  List of facts is: A, B, Z`
	`  Implications used are: {R1; R2}`
	
	`GOAL IS REACHED!`
	`Final list of facts is: A, B, Z`
	
	`End of the program!`



Paveikslas 2: Antro pavyzdžio semantinis grafas

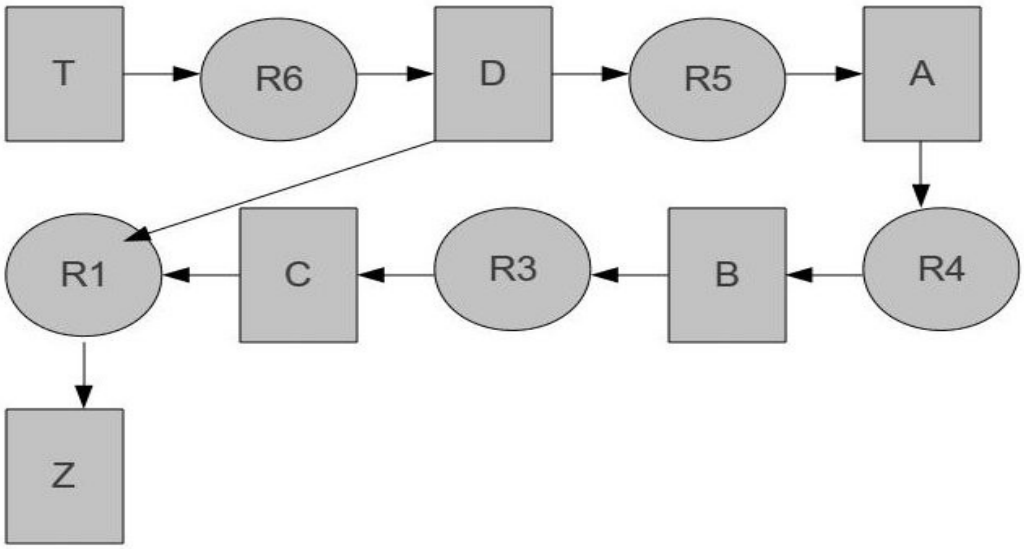
Matome, kad programa gauna teisingus rezultatus.

Pavyzdys #3

Šio pavyzdžio tikslas patikrinti ar programa sugeba dirbti su ilgu sąrašu pradinių duomenų

Failo turinys	Programos rezultatas

Productions:
R1: D, C -> Z
R2: C -> D
R3: B -> C
R4: A -> B
R5: D -> A
R6: T -> D
R7: G -> A
R8: H -> B
R9: J -> C
 	
Facts:
T
 	
Goal:
Z
	Program starts to work:
	
	Data for the program has been taken from src/InputData/input3.txt file.
	
	Initial data:
	  List of facts: T
	  Goal: Z
	  Implications:
	  R1: D, C -> Z
	  R2: C -> D
	  R3: B -> C
	  R4: A -> B
	  R5: D -> A
	  R6: T -> D
	  R7: G -> A
	  R8: H -> B
	  R9: J -> C
	
	After step nr: 1
	  List of facts is: T, D
	  Implications used are: {R6}
	
	After step nr: 2
	  List of facts is: T, D, A
	  Implications used are: {R6; R5}
	
	After step nr: 3
	  List of facts is: T, D, A, B
	  Implications used are: {R6; R5; R4}
	
	After step nr: 4
	  List of facts is: T, D, A, B, C
	  Implications used are: {R6; R5; R4; R3}
	
	After step nr: 5
	  List of facts is: T, D, A, B, C, Z
	  Implications used are: {R6; R5; R4; R3; R1}
	
	GOAL IS REACHED!
	Final list of facts is: T, D, A, B, C, Z
	
	End of the program!



Paveikslas 3: Trečio pavyzdžio semantinis grafas

Matome, kad programa tvarkingai pateikia savo darbinę būseną, o darbo rezultatai yra teisingi.

Išvados

Šiuo kurso "Dirbtinis intelektas" laboratoriniu darbu suprogramuota tiesioginio išvedimo algoritmo demostracinė programa Java programavimo kalba. Programos veikimas ištestuotas. Rezultatų teisingumas patikrintas. Tolesni darbai bus suprogramuoti atbulinio išvedimo (Backward-Chaining) algoritmą ir palyginti su tiesioginio išvedimo algoritmu.

Literatūra

[Čyr08] V. Čyras. Intelektualios sistemos, paskaitų konspektas. URL: http://uosis.mif.vu.lt/~cyras/AI/konspektas-intelektualios-sistemos.pdf.

[Nea86] R. E. Neapolitan. Forward-chaining versus a graph approach as the inference engine in expert systems. URL: http://uosis.mif.vu.lt/~cyras/AI/Neapolitan-1986-in-Principles_of_Expert_Systems.pdf

[Neg05] M. Negnevitsky. Artificial Intelligence: A Guide to Intelligent Systems.