Vilniaus universitetas  
Matematikos ir informatikos fakultetas  
Programų sistemų katedra

Atbulinio išvedimo produkcijų sistemoje demonstracinė sistema. Java kalba

Autorius: Dainius Jocas

Vilnius  
2011

Turinys

[Įvadas 2](#_Toc289205536)

[Žymėjimai 2](#_Toc289205537)

[Pseudokodas 3](#_Toc289205538)

[Duomenų failo reikalavimai 3](#_Toc289205539)

[Duomenų failo pavyzdys 3](#_Toc289205540)

[Relevantinis programos kodas 3](#_Toc289205541)

[Pavyzdys: devynios implikacijos 6](#_Toc289205542)

[Pavyzdys: modifikuotos devynios implikacijos 7](#_Toc289205543)

[Pavyzdys: neišvedamas tikslas 8](#_Toc289205544)

[Išvados 9](#_Toc289205545)

[Literatūra 9](#_Toc289205546)

# Įvadas

Šis dokumentas yra skirtas dokumentuoti ir apibendrinti antrąjį kurso „Dirbtinis inteletas“ laboratorinį darbą. Laboratorinio darbo užduotis yra suprogramuoti atbulinį išvedimą produkcijų sistemoje. Mano pasirinkta programvimo kalba yra Java. Šiame dokumente pateiksiu išsamų darbo aprašymą, atbulinio išvedimo pseudokodą, programos kodą, kuris yra relevantinis algoritmui, susiesiu pseudokodą ir programos kodą. Taip pat pateiksiu tris atbulio išvedimo pavyzdžius, pademonstruosiu jų išvedimo protokolą, kurį sugeneruoja demonstracinė sistema, nupiešiu semantinius išvedimo grafus.

# Žymėjimai

1. H – atbulinio išvedimo tikslas.
2. Faktai - pradiniai teisingi duomenys.
3. Implikacija – loginys reiškinys, turintis formą „A, B -> C“.
4. Sukcedentas – dešinioji implikacijos pusė.
5. Antecedentas – keirioji implikacijos pusė.
6. Antecedento narys – loginis kintamasis esantis antecedente.

# Pseudokodas

{Faktas; terminaliė būsena}

{ciklas}

T

Z

D

C

A

B

G

D

H

J

B

A

C

D

{Faktas}

{Aklavietė}

{Aklavietė}

{Aklavietė}

|  |  |
| --- | --- |
| Pseudokodas | Žingsnio nr. |
| atbulinis\_išvedimas(H)  if H yra\_tarp\_faktų then return true  if H nėra\_tarp\_sukcedentų then return false  if atbulinis\_isvedimas\_pakliuvo\_į\_ciklą then return false  for kiekvienai\_implikacijai\_kur\_H\_yra\_sukcedentas do  if visiems\_antecedentams  true == atbulinis\_išvedimas(antecedento\_narys) then return true  return false | {1}  {2}  {3}  {4}  {5}  {6}  {7}  {8} |

# Duomenų failo reikalavimai

1. Pirma failo eilutė gali būti arba tuščia arba su komentaru, kuris prasideda simboliu `#’.
2. Pradedant antra eilute privalo būti implikacijų sąrašas, kiekvieną implikaciją įrašant naujoje eilutėje. Implikacijos pavydys: ‚ABC‘ programa supras kaip ‚A, B -> C‘.
3. Po implikacijų sąrašo turi būti tuščia eilutė.
4. Faktai programos darbui. Pavyzdys: ‚ABC‘ programa supras kaip tris faktus ‚A, B, C‘.
5. Tuščia eilutė.
6. Atbulinio išvedimo tikslas.
7. Tuščia eilutė.

# Duomenų failo pavyzdys

|  |  |
| --- | --- |
| Eilutės numeris | Duomenų failo tekstas |
| 1:  2:  3:  4:  5:  6:  7:  8:  9: | #Dainius Jocas  FBZ  CDF  AD  ABC  Z |

# Relevantinis programos kodas

/\*\*

\* This class is implementation of the Backward-Chaining

\* @author Dainius Jocas

\*/

public class BackwardChaining {

private String facts;

private ArrayList<Implication> implications;

private String goal;

private ArrayList<String> productions = new ArrayList();

private String path = "";

/\*\*

\* Constructor

\* @param goal of the chaining

\* @param facts facts for the chaining

\* @param implications which will be used for derivation of the new facts

\*/

public BackwardChaining(String goal, String facts, ArrayList<String> implications) {

this.goal = goal;

this.facts = facts;

this.implications = new ArrayList();

for (String implication : implications) {

this.implications.add(new Implication(implication));

}

}

/\*\*

\* Method that checks if goal is derivable in production system.

\* @param goal

\* @return return true if goal is derivable, otherwise - false

\*/

/\* 1 Funkcijos deklaracija \*/

public boolean doBackwardChaining(String goal) {

System.out.println("Searching for: " + goal);

/\* 2 if H yra\_tarp\_faktų then return true \*/

if (isGoalInFacts(goal)) {

System.out.println(goal + " is in facts.");

return true;

}

/\* 3 if H nėra\_tarp\_sukcedentų then return false \*/

if (!isGoalSomewhereAsConsequent(goal)) {

System.out.println("DEADEND!");

removeLastStep();

return false;

}

/\* 4 if atbulinis\_isvedimas\_pakliuvo\_į\_ciklą then return false \*/

if (isChainingInTheLoop(goal)) {

System.out.println("LOOP!!!");

return false;

}

boolean result = true;

/\* 5 for kiekvienai\_implikacijai\_kur\_H\_yra\_sukcedentas do \*/

for (Implication implication : implications) {

if (implication.getConsequent().contains(goal)) {

addNewGoal(goal);

printDerivationPath();

/\* 6 if visiems\_antecedentams \*/

for (String newGoal : implication.getAntecedentAsList()) {

/\* 7 true == atbulinis\_išvedimas(antecedent\_narys) then return true \*/

result = result && doBackwardChaining(newGoal);

}

if (result) {

removeLastStep();

System.out.println(goal + " goes to list of facts!");

addNewFact(goal);

addImplicationToProductionSystem(implication);

return true;

} else {

result = true;

}

}

}

/\* 8 return false\*/

return false;

}

/\*\*

\* Method that checks if the goal is in facts

\* @param goal that is being searched

\* @return true if goal is in facts, otherwise - false

\*/

public boolean isGoalInFacts(String goal) {

if (this.facts.contains(goal)) {

return true;

} else {

return false;

}

}

/\*\*

\* Method that checks if there is a implications which has such a consequent

\* @param consequent

\* @return true if there is such an implication with a specific consequent,

\* otherwise - false

\*/

boolean isGoalSomewhereAsConsequent(String consequent) {

for (Implication implication : this.implications) {

if (implication.getConsequent().contains(consequent)) {

return true;

}

}

return false;

}

/\*\*

\* Method that ads a fact to the list of facts

\* @param fact

\*/

private void addNewFact(String fact) {

this.facts = this.facts + fact;

}

/\*\*

\* Method that add implication to the production system

\* @param implication

\*/

void addImplicationToProductionSystem(Implication implication) {

this.productions.add(implication.getDescriptor());

}

/\*\*

\* Method that checks if chaining is in loop

\* @param goal

\* @return true if chaining is in loop, otherwise - false.

\*/

boolean isChainingInTheLoop(String goal) {

return this.path.contains(goal);

}

}

# Pavyzdys: devynios implikacijos

|  |  |
| --- | --- |
| Duomenų failo turinys | Programos darbo protokolas |
| #Dainius Jocas  DCZ  CD  BC  AB  DA  TD  GA  HB  JC  T  Z | List of implications that was read from the data file:  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  Initial list of facts: {T}  The goal for the program: Z  Searching for: Z  Derivation path: <- Z  Searching for: D  Derivation path: <- D <- Z  Searching for: C  Derivation path: <- C <- D <- Z  Searching for: B  Derivation path: <- B <- C <- D <- Z  Searching for: A  Derivation path: <- A <- B <- C <- D <- Z  Searching for: D  LOOP!!!  Derivation path: <- A <- B <- C <- D <- Z  Searching for: G  DEADEND!  Derivation path: <- B <- C <- D <- Z  Searching for: H  DEADEND!  Derivation path: <- C <- D <- Z  Searching for: J  DEADEND!  Derivation path: <- D <- Z  Searching for: T  T is in facts.  D goes to list of facts!  Searching for: C  Derivation path: <- C <- Z  Searching for: B  Derivation path: <- B <- C <- Z  Searching for: A  Derivation path: <- A <- B <- C <- Z  Searching for: D  D is in facts.  A goes to list of facts!  B goes to list of facts!  C goes to list of facts!  Z goes to list of facts!  The goal is reached!  The answer is: {R6, R5, R4, R3, R1} |

{Faktas; terminaliė būsena}

{ciklas}

T

Z

D

C

A

B

G

D

H

J

B

A

C

D

{Faktas}

{Aklavietė}

{Aklavietė}

{Aklavietė}

Semantinis grafas 1: Devynios produkcijos

# Pavyzdys: modifikuotos devynios implikacijos

|  |  |
| --- | --- |
| Duomenų failo turinys | Programos darbo protokolas |
| #Dainius Jocas  CDZ  CD  BC  AB  DA  TD  GA  HB  JC  T  Z | List of implications that was read from the data file:  R1: C, D -> 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  Initial list of facts: {T}  The goal for the program: Z  Searching for: Z  Derivation path: <- Z  Searching for: C  Derivation path: <- C <- Z  Searching for: B  Derivation path: <- B <- C <- Z  Searching for: A  Derivation path: <- A <- B <- C <- Z  Searching for: D  Derivation path: <- D <- A <- B <- C <- Z  Searching for: C  LOOP!!!  Derivation path: <- D <- A <- B <- C <- Z  Searching for: T  T is in facts.  D goes to list of facts!  A goes to list of facts!  B goes to list of facts!  C goes to list of facts!  Searching for: D  D is in facts.  Z goes to list of facts!  The goal is reached!  The answer is: {R6, R5, R4, R3, R1} |

# Pavyzdys: neišvedamas tikslas

Semantinis grafas 2: Modifikuotos devynio produkcijos

{ciklas}

Z

C

B

D

A

T

C

D

{Faktas, terminalinė būsena}

{Faktas}

|  |  |
| --- | --- |
| Duomenų failo turinys | Programos darbo protokolas |
| # Dainius Jocas  AB  A  Z | List of implications that was read from the data file:  R1: A -> B  Initial list of facts: {A}  The goal for the program: Z  Searching for: Z  DEADEND!  Goal is not derivable in the production system |

# Išvados

Šiame kurso „Dirbtinis intelektas“ laboratoriniame darbe buvo suprogramuota atbulinio išvedimo produkcijų sistemoje demostracinė sistema Java programvimo kalba. Programos veikimas ištestuotas. Rezultatų teisingumas patikrintas juos lyginant su semantiniais grafais.

# 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.