Artificial intelligence - Project 2
- Propositional Logic and FOL -

Pop Ruxandra Maria, Zelenszky Bianca

01/12/2021

1 Introducere

Implementarea proiectului se bazează pe logica propozitională a predicatelor. Logica propozițională se ocupă de propoziții (care pot fi false sau adevărate) și de relațiile dintre propoziții.

Există 6 tipuri de simboluri în logica predicatelor:

- Predicate: p, q, r...
- Constante: a, b, c, mașina...
- Variabile: x, y, z...
- Conective: |, ->, <->, -
- Paranteze: (,)
- Cuantificatori: forall, exists...

O propoziție atomică este una al cărei adevăr sau falsitate nu depinde de adevărul sau falsitatea oricărei alte propoziții.

Următoarele propoziții sunt adevărate:

- Simbolurile True și False sunt propoziții atomice
- Simbolurile P1, P2 etc. sunt propoziții atomice
- Dacă S este o propoziție => ¬S este o propoziție (negation)
- Dacă S1 și S2 sunt propoziții => S1 S2 este o propoziție (conjunction)
- Dacă S1 și S2 sunt propoziții => S1 | S2 este o propoziție (disjunction)
- Dacă S1 și S2 sunt propoziții => S1 -> S2 este o propoziție (implication)
- Dacă S1 și S2 sunt propoziții => S1 <-> S2 este o propoziție (biconditional)

Tabele de adevăr:
1. AND

p	q	p and q
F	F	F
F	T	F
T	F	F
T	T	T

2.OR

p	q	p or q
F	F	F
F	T	T
T	F	T
T	T	T

3.If ... then

p	q	p -> q
F	F	T
F	T	T
T	F	F
T	T	T

3.Iff

p	q	p <-> q
F	F	T
F	T	F
T	F	F
T	T	T

2 Cerința problemei

2.1 Intersectie nedirijata

Avem patru mașini într-o intersecție nedirijată: M1, M2, M3, M4.
Intersecția e bazată pe desenul din cerință.
Cele două străzi sunt cu dublu sens.

- M1 nu semnalizează și merge în față.
- M2 semnalizează dreapta.
- M3 semnalizează stânga.
- M4 nu semnalizează și merge în față.

M1 se află în dreapta lui M4, M2 se află în dreapta lui M1, M3 se afla în dreapta lui M2, M4 se află în dreapta lui M3.

Vrem să determinăm dacă trece M2 primul, M1 al doilea, M4 al treilea și M3 ultimul.

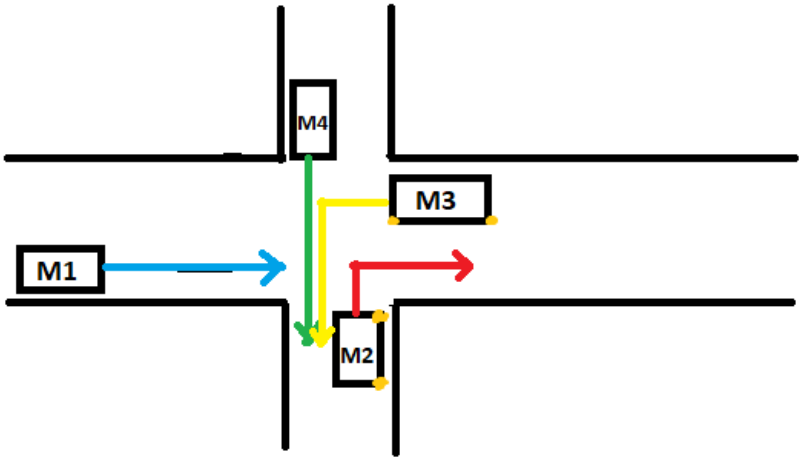

Figure 1: Intersectia nedirijata.

2.2 Intersectie dirijata

Avem patru mașini într-o intersecție dirijată: M1, M2, M3, M4.
Intersecția e bazată pe desenul din cerință.
Cele două străzi sunt cu dublu sens.

- M1 semnalizează stânga și se află pe strada cu semnul CEDEAZĂ.
- M2 nu semnalizează și merge în față și se află pe strada cu prioritate.
- M3 nu semnalizează și merge în față și se află pe strada cu semnul STOP.
- M4 semnalizează stânga și se află pe strada cu prioritate.

Vrem să determinăm dacă trece M2 primul, M4 al doilea, M3 al treilea și M1 ultimul.

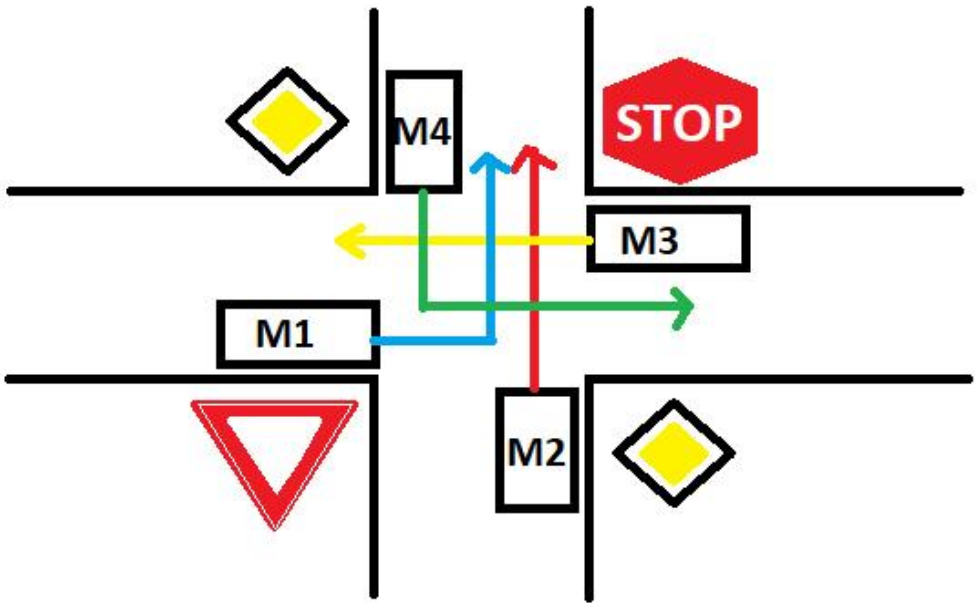

Figure 2: Intersectia dirijata.

3 Implementare

Prove9 este un tool folosit pentru a demonstra teoreme bazate pe logica propoziţională şi FOL(First Order Logic). Mace4 este folosit pentru a genera modele pe baza logicii propoziţionale sau FOL.

Pentru a reprezenta în FOL o problemă bazată pe logica propoziţională trebuie luat în considerare dimensiunea domeniului. Problema propusă de noi are un domeniu de dimensiune 4 (domain size: 4). Constantele problemei sunt: M1,M2,M3,M4; reprezentând cele 4 maşini aflate în intersecţie.

3.1 Intersecţie nedirijată

Predicatele problemei pentru intersectia nedirijată sunt:

- semnalizeaza(x): maşina x semnalizează
- semnalizeaza st(x): maşina x semnalizează stânga
- semnalizeaza dr(x): maşina x semnalizează dreapta
- fata(x): maşina x merge în faţă
- dreapta(x): maşina x merge în dreapta
- stanga(x): maşina x merge in stânga
- in dreapta(x, y): maşina y se află în dreapta lui x
- prioritate1(x): maşina x are prima prioritate
- prioritate2(x): maşina x are prioritate dupa ce trece prima maşină
- prioritate3(x): maşina x are prioritate după ce trece prima şi a doua maşină
- prioritate4(x): maşina x are prioritate după ce trece prima, a doua şi a treia maşină
- trece1(x): maşina x trece prima
- trece2(x): maşina x trece a doua
- trece3(x): maşina x trece a treia
- trece4(x): maşina x trece a patra

Explicaţii:

- Deoarece într-o intersecţie nedirijată se aplică prioritatea de dreapta, am ales să creez un predicat in dreapta(x, y) care o să arate care maşină se află în dreapta cărei maşini (de exemplu: in dreapta(M1, M2) înseamnă că maşina M2 este în dreapta lui M1).
- Dacă o maşină nu semnalizează, asta înseamnă că ea nu semnalizează la stânga sau la dreapta, adică merge în faţă.
- Predicatele prioritate1(x), prioritate2(x), prioritate3(x) şi prioritate4(x) arată ordinea de prioritate a maşinilor. De exemplu, prioritate3(M4) ar însemna că M4 ar fi a treia maşină care ar avea prioritate după ce au trecut primele două maşini.
- Liniile 49-52 asigură că o singură maşină poate avea prioritate la un moment dat.
- Predicatele trece1(x), trece2(x), trece3(x) şi trece4(x) indică ordinea de trecere a maşinilor. De exemplu, trece1(M2) ar însemna că maşina M2 trece prima în intersecţie.

Code:

```
1  % Saved by Prover9-Mace4 Version 0.5, December 2007.
2
3  set(ignore_option_dependencies). % GUI handles dependencies
4
5  if(Prover9). % Options for Prover9
6    assign(max_seconds, 60).
7  end_if.
```

```
8
9  if(Mace4).   % Options for Mace4
10   assign(max_seconds, 60).
11 end_if.
12
13 formulas(assumptions).
14
15 % Avem patru masini intr-o intersectie nedirijata: M1, M2, M3, M4.
16 % Intersectia e bazata pe desenul din cerinta.
17 % Cele doua strazi sunt cu dublu sens.
18 % M1 nu semnalizeaza si merge in fata.
19 % M2 semnalizeaza dreapta.
20 % M3 semnalizeaza stanga.
21 % M4 nu semnalizeaza si merge in fata.
22 % M1 se afla in dreapta lui M4, M2 se afla in dreapta lui M1, M3 se afla in dreapta lui M2, M4 se afl
23 % dreapta lui M3.
24
25 % Vrem sa determinam daca trece M2 primul, M1 al doilea, M4 al treilea si M3 ultimul.
26
27 % domain size 4 : {M1, M2, M3, M4}
28
29 % constant: M1, M2, M3, M4
30 % predicate: semnalizeaza(x): masina x semnalizeaza
31 % predicate: semnalizeaza_st(x): masina x semnalizeaza stanga
32 % predicate: semnalizeaza_dr(x): masina x semnalizeaza dreapta
33 % predicate: fata(x): masina x merge in fata
34 % predicate: dreapta(x): masina x merge in dreapta
35 % predicate: stanga(x): masina x merge in stanga
36 % predicate: in_dreapta(x, y): masina y se afla in dreapta lui x
37 % predicate: prioritate1(x): masina x are prima prioritate
38 % predicate: prioritate2(x): masina x are prioritate dupa ce trece prima masina
39 % predicate: prioritate3(x): masina x are prioritate dupa ce trece prima si a doua masina
40 % predicate: prioritate4(x): masina x are prioritate dupa ce trece prima, a doua si a treia masina
41 % predicate: trece1(x): masina x trece prima
42 % predicate: trece2(x): masina x trece a doua
43 % predicate: trece3(x): masina x trece a treia
44 % predicate: trece4(x): masina x trece a patra
45
46 -semnalizeaza_st(x) & -semnalizeaza_dr(x) -> fata(x).
47 semnalizeaza_dr(x) -> dreapta(x) & prioritate1(x).
48 semnalizeaza_st(x) -> stanga(x).
49 -semnalizeaza(x) -> -semnalizeaza_st(x) & -semnalizeaza_dr(x).
50
51 -semnalizeaza(M1).
52 semnalizeaza_dr(M2).
53 semnalizeaza_st(M3).
54 -semnalizeaza(M4).
55
56 in_dreapta(M1, M2).
57 in_dreapta(M2, M3).
58 in_dreapta(M3, M4).
59 in_dreapta(M4, M1).
60
61 prioritate1(x) -> -prioritate2(x) & -prioritate3(x) & -prioritate4(x).
62 prioritate2(x) -> -prioritate1(x) & -prioritate3(x) & -prioritate4(x).
63 prioritate3(x) -> -prioritate1(x) & -prioritate2(x) & -prioritate4(x).
```

```
64 prioritate4(x) -> -prioritate1(x) & -prioritate2(x) & -prioritate3(x).
65
66 % Masina M2 merge in dreapta si are prima prioritate
67
68 dreapta(x) & prioritate1(x) -> trece1(x) & -trece2(x) & -trece3(x) & -trece4(x).
69
70 % Masina M1 a avut in dreapta o masina care are prioritate si vrea s-o ia in fata
71
72 fata(x) & in_dreapta(x, y) & prioritate1(y) -> prioritate2(x) & -trece1(x) & trece2(x) & -trece3(x) &
73
74 % Masina M4 a avut in dreapta o masina care are prioritate si vrea s-o ia in fata
75
76 fata(x) & in_dreapta(x, y) & prioritate2(y) -> prioritate3(x) & -trece1(x) & -trece2(x) & trece3(x) &
77
78 % Masina M3 a avut in dreapta o masina care are prioritate si vrea s-o ia in stanga
79
80 stanga(x) & in_dreapta(x, y) & prioritate3(y) -> prioritate4(x) & -trece1(x) & -trece2(x) & -trece3(x
81
82 end_of_list.
83
84 formulas(goals).
85
86 trece3(M4).
87
88 end_of_list.
```

Comenzi:

Pentru rularea codului ne-am folosit de tool-ul Prover9 care demonstrează dacă o mașină trece în ordine sau nu. Astfel, se pot pune următoarele goal-uri pentru a testa codul:

- trece1(M2), trece2(M1), trece3(M4) sau trece4(M3) o să demonstreze ordinea trecerii mașinilor: M2 trece prima, M1 o să treacă a doua, M4 a treia și M3 ultima.
- trece1(M1), trece1(M3), trece1(M4), trece2(M2), trece2(M3), trece2(M4), trece3(M1), trece3(M2), trece3(M3), trece4(M1), trece4(M2) sau trece4(M4) vor da erori deoarece algoritmul nu o să reușească să găsească o demonstrație pentru ele.
- -trece1(M1), -trece1(M3), -trece1(M4), -trece2(M2), -trece2(M3), -trece2(M4), -trece3(M1), -trece3(M2), -trece3(M3), -trece4(M1), -trece4(M2) sau -trece4(M4) vor reuși să demonstreze faptul că mașinile nu vor trece în ordine în intersecție.
- -trece1(M2), -trece2(M1), -trece3(M4) sau -trece4(M3) vor da eroare deoarece algoritmul nu o să reușească să le demonstreze.

Comanda pentru rularea codului din terminal: prover9 -f ”aut.in”

3.2 Intersecție dirijată

Predicatele problemei pentru intersecția dirijată sunt:

- semnalizeaza(x): mașina x semnalizează
- semnalizeaza st(x): mașina x semnalizează stânga
- semnalizeaza dr(x): mașina x semnalizează dreapta
- fata(x): mașina x merge în față
- dreapta(x): mașina x merge în dreapta
- stanga(x): mașina x merge în stânga
- semn(x, cedeaza), semn(x, prioritate), semn(x, stop): mașina x are semnul de cedează/prioritate/stop
- peAceeasiStradă(x, y, prioritate), peAceeasiStrada(x, y, faraPrioritate): mașinile x si y se află pe o stradă cu prioritate/fără prioritate
- trece1(x): mașina x trece prima
- trece2(x): mașina x trece a doua
- trece3(x): mașina x trece a treia
- trece4(x): mașina x trece a patra

Explicații:

- Predicatele semn(x, cedeaza), semn(x, prioritate) și semn(x, stop) arată semnul specific intrării în intersecție unei mașini (de exemplu, semn(M1, cedează) înseamnă că mașina are cedează trecerea).
- La linia 43, dacă două mașini sunt pe drumul cu semnul de prioritate, atunci predicatul peAceeasiStrada(x, y, prioritate) este adevărat.
- La linia 46, dacă două mașini sunt pe drumul cu semnul de cedează sau stop, atunci predicatul peAceeasiStrada(x, y, faraPrioritate) este adevărat.

Code:

```
1  %Avem patru mașini intr-o intersecție dirijată: M1, M2, M3, M4.
2  %Intersecția e bazată pe desenul din cerință.
3  %M1 semnalizează stânga si se află pe strada cu semnul CEDEAZĂ
4  %M2 nu semnalizează si merge in față si se află pe strada cu prioritate.
5  % M3  nu semnalizează si merge in față si se află pe strada cu semnul STOP.
6  %M4  semnalizează stânga si se află pe strada cu prioritate.
7  %Vrem să determinăm dacă trece M2 primul, M4 al doilea, M3 al treilea si M1 ultimul.
8  formulas(assumptions).
9
10
11 % domain size 4 : {M1, M2, M3, M4}
12
13 % constant: M1, M2, M3, M4
14 % predicate: semnalizeaza(x): masina x semnalizeaza
15 % predicate: semnalizeaza_st(x): masina x semnalizeaza stanga
16 % predicate: semnalizeaza_dr(x): masina x semnalizeaza dreapta
17 % predicate: fata(x): masina x merge in fata
18 % predicate: dreapta(x): masina x merge in dreapta
19 % predicate: stanga(x): masina x merge in stanga
20 % predicate: semn(x, cedeaza), semn(x, prioritate), semn(x, stop): masina x are semnul de cedeaza/pri
21 % predicate: pe_aceeasi_strada(x, y, prioritate), pe_aceeasi_strada(x, y, fara_prioritate): masinile
22 % predicate: trece1(x): masina x trece prima
23 % predicate: trece2(x): masina x trece a doua
```

```
% predicate: trece3(x): masina x trece a treia
% predicate: trece4(x): masina x trece a patra

-semnalizeaza_st(x) & -semnalizeaza_dr(x) -> fata(x).
semnalizeaza_dr(x) -> dreapta(x).
semnalizeaza_st(x) -> stanga(x).
-semnalizeaza(x) -> -semnalizeaza_st(x) & -semnalizeaza_dr(x).

semnalizeaza_st(M1).
-semnalizeaza(M2).
-semnalizeaza(M3).
semnalizeaza_st(M4).

semn(M1, cedeaza).
semn(M2, prioritate).
semn(M3, stop).
semn(M4, prioritate).

% daca doua masini sunt pe aceeasi strada cu prioritate
semn(x, prioritate) & semn(y, prioritate) -> pe_aceeasi_strada(x, y, prioritate).

% daca doua masini sunt pe aceeasi strada fara prioritate
(semn(x, cedeaza) & semn(y, stop)) | (semn(x, stop) & semn(y, cedeaza)) -> pe_aceeasi_strada(x, y, fa

% daca masina are prioritate si merge in fata, trece prima
pe_aceeasi_strada(x, y, prioritate) & fata(x) & stanga(y) -> trece1(x) & -trece2(x) & -trece3(x) & -t

% daca masina are prioritate si merge in stanga, trece al doilea
pe_aceeasi_strada(x, y, prioritate) & stanga(x) & fata(y) -> -trece1(x) & trece2(x) & -trece3(x) & -t

% daca masina are stop si o alta masina are cedeaza, si merge in fata, are prioritate
pe_aceeasi_strada(x, y, fara_prioritate) & fata(x) & stanga(y) ->  -trece1(x) & -trece2(x) & trece3(x

% daca masina are cedeaza trecerea si o ia la stanga, iar o alta masina are stop si o ia in fata, e u
pe_aceeasi_strada(x, y, fara_prioritate) & stanga(x) & fata(y) ->  -trece1(x) & -trece2(x) & -trece3(

end_of_list.

formulas(goals).
end_of_list.
```

Comenzi:

Pentru rularea codului ne-am folosit de tool-ul Prover9 care demonstrează dacă o mașină trece în ordine sau nu. Astfel, se pot pune următoarele goal-uri pentru a testa codul:

- trece1(M2), trece2(M4), trece3(M3) sau trece4(M1) o să demonstreze ordinea trecerii mașinilor: M2 trece prima, M4 o să treacă a doua, M3 a treia și M1 ultima.
- trece1(M1), trece1(M3), trece1(M4), trece2(M1), trece2(M2), trece2(M3), trece3(M1), trece3(M2), trece3(M4), trece3(M2), trece3(M3) sau trece3(M4) vor da erori deoarece algoritmul nu o să reușească să găsească o demonstrație pentru ele.
- -trece1(M1), -trece1(M3), -trece1(M4), -trece2(M1), -trece2(M2), -trece2(M3), -trece3(M1), -trece3(M2), -trece3(M4), -trece3(M2), -trece3(M3) sau -trece3(M4) vor reuși să demonstreze faptul că mașinile nu vor trece în ordine în intersecție.
- -trece1(M2), -trece2(M4), -trece3(M3) sau -trece4(M1) vor da eroare deoarece algoritmul nu o să reușească să le demonstreze.

Comanda pentru rularea codului din terminal: prover9 -f "aut_dir.in"

4 Rezultate

4.1 Intersecție nedirijată

Demonstrație:

Pentru a ilustra corectitudinea acestui cod, vom începe prin a demonstra un caz particular: cel în care goal-ul este trace3(M4). Asta înseamnă că mașina M4 este a treia care trece în intersecție. Se știe că:

- Mașina M1 nu semnalizează. Rezultă că M1 nu semnalizează nici la stânga, nici la dreapta, deci merge în față. (1)
- Mașina M2 semnalizează dreapta. Rezultă că M2 merge la dreapta și are prima dată prioritate. (2)
- Mașina M4 nu semnalizează. Analog cu (1), M4 merge în față. (3)
- În dreapta lui M1 se află M2. (4)
- În dreapta lui M4 se află M1. (5)
- Prioritatea este unică (nu poți avea două mașini cu aceeași prioritate). (6)
- Într-o intersecție nedirijată se aplică prioritatea de dreapta. (7)

Din (7) rezultă că următoarea mașină care o să aibă prioritate o să fie mașina care are în dreapta mașina care a avut prima dată prioritate (2) și care merge în față (1). Deci va trece mașina M1 (4) și M1 devine prioritar pentru restul mașinilor (8) (deoarece M2 a trecut și (6)).

Din (7) rezultă că următoarea mașină care o să aibă prioritate o să fie mașina care are în dreapta mașina care a avut a doua oară prioritate (8) și care merge în față (3). Deci va trece mașina M4 (5) (deoarece M2 și M1 au trecut și (6)).

Astfel, am demonstrat că a treia mașină care trece este M4.

Code:

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 19004 was started by biank on Bianca-PC,
Thu Dec  2 17:25:44 2021
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/bin-win32/prover9".
============================== end of head ===========================

============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.00 (+ 0.08) seconds.
% Length of proof is 30.
% Level of proof is 6.
% Maximum clause weight is 4.
% Given clauses 6.

1 -semnalizeaza_st(x) & -semnalizeaza_dr(x) -> fata(x) # label(non_clause).  [assumption].
2 semnalizeaza_dr(x) -> dreapta(x) & prioritate1(x) # label(non_clause).  [assumption].
4 -semnalizeaza(x) -> -semnalizeaza_st(x) & -semnalizeaza_dr(x) # label(non_clause).  [assumption].
10 fata(x) & in_dreapta(x,y) & prioritate1(y) -> prioritate2(x) & -trece1(x) & trece2(x) & -trece3(x)
11 fata(x) & in_dreapta(x,y) & prioritate2(y) -> prioritate3(x) & -trece1(x) & -trece2(x) & trece3(x)
13 trece3(M4) # label(non_clause) # label(goal).  [goal].
15 semnalizeaza_st(x) | semnalizeaza_dr(x) | fata(x).  [clausify(1)].
16 semnalizeaza(x) | -semnalizeaza_st(x).  [clausify(4)].
18 semnalizeaza_dr(M2).  [assumption].
20 -semnalizeaza_dr(x) | prioritate1(x).  [clausify(2)].
```

```
21 semnalizeaza(x) | -semnalizeaza_dr(x).  [clausify(4)].
23 semnalizeaza(x) | semnalizeaza_dr(x) | fata(x).  [resolve(16,b,15,a)].
24 -fata(x) | -in_dreapta(x,y) | -prioritate1(y) | prioritate2(x).  [clausify(10)].
25 in_dreapta(M1,M2).  [assumption].
28 in_dreapta(M4,M1).  [assumption].
36 -fata(x) | -in_dreapta(x,y) | -prioritate2(y) | trece3(x).  [clausify(11)].
43 prioritate1(M2).  [resolve(18,a,20,a)].
56 -fata(M1) | -prioritate1(M2) | prioritate2(M1).  [resolve(24,b,25,a)].
75 -fata(M1) | prioritate2(M1).  [resolve(56,b,43,a)].
95 -fata(M4) | -prioritate2(M1) | trece3(M4).  [resolve(36,b,28,a)].
145 -fata(M1) | -fata(M4) | trece3(M4).  [resolve(75,b,95,b)].
146 -trece3(M4).  [deny(13)].
252 -semnalizeaza(M1).  [assumption].
253 -semnalizeaza(M4).  [assumption].
258 semnalizeaza(x) | fata(x) | semnalizeaza(x).  [resolve(23,b,21,b)].
259 semnalizeaza(x) | fata(x).  [copy(258),merge(c)].
263 -fata(M1) | -fata(M4).  [resolve(145,c,146,a)].
268 fata(M4).  [resolve(259,a,253,a)].
269 fata(M1).  [resolve(259,a,252,a)].
270 $F.  [back_unit_del(263),unit_del(a,269),unit_del(b,268)].

============================== end of proof ==========================
```

4.2 Intersecție dirijată

Pentru a ilustra corectitudinea acestui cod, vom începe prin a demonstra un caz particular: cel în care goal-ul este trace3(M3). Asta înseamnă că mașina M3 este a treia care trece în intersecție. Se știe că:

- Mașina M1 semnalizează la stânga. Rezultă că M1 merge la stânga. (1)
- Mașina M2 nu semnalizează. Rezultă că M2 nu semnalizează nici la stânga, nici la dreapta, deci merge în față. (2)
- Mașina M3 nu semnalizează. Analog cu (2), M3 merge în față. (3)
- Mașina M4 semnalizează la stânga. Rezultă că M1 merge la stânga. (4)
- Semnul de la intrarea în intersecție a lui M1 este de cedează trecerea. (5)
- Semnul de la intrarea în intersecție a lui M2 este de drum cu prioritate. (6)
- Semnul de la intrarea în intersecție a lui M3 este stop. (7)
- Semnul de la intrarea în intersecție a lui M4 este de drum cu prioritate. (8)
- Prima dată trec mașinile de pe drumul cu prioritate (9)

Din (6) și (8) rezultă că mașinile M2 și M4 sunt pe aceeasi stradă cu prioritate. (10)
Din (5) și (7) rezultă că mașinile M1 și M3 sunt pe aceeași stradă fără prioritate. (11)
Din (9), (10), faptul că mașina M2 merge in față (2) și mașina M4 merge la stânga (4), rezultă că M2 are prioritate și trece primul, iar M4 trece al doilea.
Din (9), (11), faptul că mașina M3 merge în față (3) și mașina M1 merge la stânga (1), rezultă că M3 trece al treilea și M1 ultimul.
Astfel, am demonstrat că a treia mașină care trece este M3.
Code:

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 16980 was started by biank on Bianca-PC,
Thu Dec  2 19:25:18 2021
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/bin-win32/prover9".
============================== end of head ===========================

```

```
============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.05) seconds.
% Length of proof is 26.
% Level of proof is 6.
% Maximum clause weight is 7.
% Given clauses 23.

1 -semnalizeaza_st(x) & -semnalizeaza_dr(x) -> fata(x) # label(non_clause).  [assumption].
3 semnalizeaza_st(x) -> stanga(x) # label(non_clause).  [assumption].
4 -semnalizeaza(x) -> -semnalizeaza_st(x) & -semnalizeaza_dr(x) # label(non_clause).  [assumption].
6 semn(x,cedeaza) & semn(y,stop) | semn(x,stop) & semn(y,cedeaza) -> pe_aceeasi_strada(x,y,fara_prior
9 pe_aceeasi_strada(x,y,fara_prioritate) & fata(x) & stanga(y) -> -trece1(x) & -trece2(x) & trece3(x)
11 trece3(M3) # label(non_clause) # label(goal).  [goal].
12 -semnalizeaza_st(x) | stanga(x).  [clausify(3)].
13 semnalizeaza_st(x) | semnalizeaza_dr(x) | fata(x).  [clausify(1)].
14 semnalizeaza(x) | -semnalizeaza_st(x).  [clausify(4)].
15 semnalizeaza_st(M1).  [assumption].
19 semnalizeaza(x) | -semnalizeaza_dr(x).  [clausify(4)].
20 semnalizeaza(x) | semnalizeaza_dr(x) | fata(x).  [resolve(14,b,13,a)].
24 -semn(x,stop) | -semn(y,cedeaza) | pe_aceeasi_strada(x,y,fara_prioritate).  [clausify(6)].
34 -pe_aceeasi_strada(x,y,fara_prioritate) | -fata(x) | -stanga(y) | trece3(x).  [clausify(9)].
41 -trece3(M3).  [deny(11)].
44 -fata(x) | -stanga(y) | trece3(x) | -semn(x,stop) | -semn(y,cedeaza).  [resolve(34,a,24,c)].
66 -semnalizeaza(M3).  [assumption].
67 semn(M1,cedeaza).  [assumption].
69 semn(M3,stop).  [assumption].
71 stanga(M1).  [resolve(15,a,12,a)].
76 semnalizeaza(x) | fata(x) | semnalizeaza(x).  [resolve(20,b,19,b)].
77 semnalizeaza(x) | fata(x).  [copy(76),merge(c)].
81 -fata(M3) | -stanga(x) | -semn(M3,stop) | -semn(x,cedeaza).  [resolve(44,c,41,a)].
82 -fata(M3) | -stanga(x) | -semn(x,cedeaza).  [copy(81),unit_del(c,69)].
118 -fata(M3).  [resolve(82,c,67,a),unit_del(b,71)].
146 $F.  [resolve(118,a,77,b),unit_del(a,66)].

============================== end of proof ==========================
```

5 Concluzie

In concluzie, acest proiect ne-a ajutat sa ne aprofundam cunostintele, acumulate pe parcursul laboratoarelor, am invatat sa folosim tool-urile Prover9 si Mace4.