Universitatea Tehnică Cluj-Napoca
Departamentul Calculatoare

Sisteme Expert Bazate pe Reguli

Radu Răzvan Slăvescu

DRAFT 2024

Cuprins

Lab 9

Aplicație: Formalizare de reguli de circulație în CLIPS (I)

Lucrarea prezentă și cea următoare au ca scop formalizarea deciziilor unui șofer virtual ca reguli CLIPS [3]. Aceste decizii se pot referi la prevederi ale regulamentului de circulație (validitatea unei manevre din punct de vedere al prevederilor Codului Rutier), la conduita conducătorului auto etc.

9.1 Codul existent

Dezvoltarea regulilor se face adăugandu-le pe acestea la o aplicatîe de baza structurată în 3 module: MAIN, PERCEPT-MANAGER și AGENT. Primele 2 module constituie un simulator și sunt păstrate de obicei nemodificate. Regulile de comportament al conducătorului sunt amplasate în cel de al treilea modul (AGENT). Cele trei module se execută într-o buclă teoretic infinită. Pratic, aceasta este limitată de numarul `n` specificat ca parametru al lui `run` în fișierul `go`, care asigură că programul se oprește după un număr specificat de reguli executate, deoarece secvența de percepții disponibile pentru testare este de obicei finită.

9.1.1 Percepții

Modelul de timp folosit de simulator este liniar, împărțit în episoade discrete de lungime egală. Fiecărui moment de timp n îi corespunde un set de percepții, descrise sub formă de fapte. Acestea sunt stocate în fișierul `tn.clp` din directorul de percepții specificat in fișierul de configurare (ex. `perceptiiTest05`). Modulul PERCEPT-MANAGER este cel care citește, în fiecare moment de timp, percepțiile corespunzătoare acestuia și le face cunoscute agentului conducător auto.

Percepțiile pe care le poate primi un agent (din mediu, via PERCEPT-MANAGER) sunt descrise strict prin următorul template:

```
(deftemplate PERCEPT-MANAGER::ag_percept
    (slot percept_pname) ; animal|indicator|marcaj
    (slot percept_pval); albina|barza|... or depasire_interzisa|...
                       ;  or (linie_continua|...)
    (slot percept_pdir); left|right|ahead
)
```

Slotul `percept_pname` descrie numele percepției, în timp ce slotul `percept_pval` contine valoarea propriu-zisă a acesteia. Prin acest mecanism bazat pe reificare, se pot manipula orice astfel de percepții fără a fi necesară extinderea setului de sloturi ale templateului. Introducerea unei noi percepții se face adăugând o nouă valoare la setul permis pentru `percept_pval`; în cazul în care este nevoie de adăugarea unei întregi clase de percepții (de exemplu "marcaj pe asfalt"), se va introduce o nouă valoare pentru `percept_pname` împreună cu valorile corespunzătoare pentru `percept_pval` (ex. "marcaj continuu").

Slotul `percept_pdir` este prevăzut pentru a indica directia percepției. De exemplu, este util să putem exprima ideea "percep un automobil pe un drum perpendicular pe cel pe care ne aflăm, și care se apropie din dreapta".

Ca exemplu, fișierul `t1.clp` poate conține percepțiile următoare:

```
(ag_percept
  (percept_pname animal)
  (percept_pval albina))
(ag_percept
  (percept_pname road_sign)
  (percept_pval depasire_interzisa))
```

Aceasta înseamnă că, la momentul t_1, șoferul vede o albină și semnul de circulație "Depașirea interzisă".

9.1.2 Modulul MAIN

Modulul MAIN este minimal. Rolul său este de a muta succesiv focusul între celelalte 2 module. În cazul în care focusul urmează a fi preluat de catre modulul PERCEPT-MANAGER, este generat și asertat un fapt numit `(tic)`, similar unei bătăi a tactului, care va duce la incrementarea corespunzătoare a timpului in PERCEPT-MANAGER.

```
(defrule MAIN::succesiune "change current module in sequence"
 ?lista <- (secventa ?urm $?rest)
=>
 (retract ?lista)
 (assert (secventa $?rest ?urm))
 (if (eq ?urm PERCEPT-MANAGER)
   then (assert (tic)) ; folosit in PERCEPT-MANAGER pt. a incrementa timpul
 (if (eq ?*ag-tic-in-debug* TRUE)
   then (printout t "    <D> MAIN: adaugare tic" crlf)))
 (if (eq ?*main-in-debug* TRUE)
   then (printout t "    <D> MAIN: switch to " ?urm  crlf))
 (focus ?urm))
```

Regulile sunt prevăzute cu posibililtatea de a scrie mesaje suplimentare de depanare prin setarea valorii unor variabile globale (ex. `?*main-in-debug*`)

9.1.3 Modulul PERCEPT-MANAGER

Acest modul îndeplinește următoarele sarcini:

1. actualizează timpul curent
2. șterge percepțiile anterioare
3. citește percepțiile curente din fișierul corespunzător timpului curent din directorul de percepții.

Regula `PERCEPT-MANAGER::hk-ag_percepts` șterge toate percepțiile existente, pentru ca acestea să nu interfereze cu cele nou venite. La fiecare moment de timp, PERCEPT-MANAGER-ul va prezenta doar percepțiile corespunzătoare momentului curent, deci vechile percepții trebuie eliminate.

```
(defrule PERCEPT-MANAGER::hk-ag_percepts
    (tic) ; to avoid deleting newly added percepts
    ?fp <- (ag_percept (percept_pname ?pn) (percept_pval ?pv))
=>
    (retract ?fp))
```

Grație utilizării unui unic template pentru percepții, regula aceasta este suficientă pentru a gestiona percepțiile primite din mediu.

Regula `PERCEPT-MANAGER::advance-time-percepts` va actualiza valoarea timpului curent din secvență și va elimina din memoria de lucru faptul `tic`, pentru a asigura ca are loc o singura astfel de operație.

```
(defrule PERCEPT-MANAGER::advance-time-percepts
    (declare (salience -95)); dupa stergerea perc. anterioare
    ?tc <- (tic) ; pentru a nu intra in bucla infinita
    ?tp <- (timp (valoare ?t))
=>
    (if (eq ?*sim-in-debug* TRUE)
      then (printout t "<D>PERCEPT-MANAGER: advance-time-percepts
      (+ stergere tic)" crlf))
    (bind ?nt (+ 1 ?t))
    (printout t "   PERCEPT-MANAGER: timp = " ?nt crlf)
    (retract ?tp)
    (assert (timp (valoare ?nt)))
    (retract ?tc) ; pentru a nu intra in bucla infinita
    (load-facts (sym-cat ?*perceptsDir* "t" ?nt ".clp"))
)
```

Având valoarea un `salience` de -95, aceasta regulă se executa ultima, deci ea va adăuga la memoria de lucru fapte care descriu percepțiile curente numai după ce percepțiile anterioare au fost eliminate, iar valoarea timpului curent a fost actualizată corespunzător.

9.2 Constante de configurare

Fișierul `headers.clp` definește și un set de constante folosite pentru debug. Acestea condiționează tipărirea unor mesaje de debug în timpul rularii, în regulile în care apar (ex. `?*sim-in-debug*`). Tot aici se specifică directorul care conține secvența de percepții (`?*perceptsDir*`)

9.3 Specificarea manevrelor urmarite

Manevrele cu privire la care agentul trebuie să ia o decizie (de exemplu de validare din punct de vedere al regulamentului) sunt specificare in fișierul `maneuverValidityASK.clp`. Ca exemplu, dacă dorim validarea manevrei de depașire (`overtaking`) și a celei de viraj dreapta, în acest fișier va trebui sa adaugam următorul conținut:

```
(deffacts AGENT::maneuvers-to-validate
    (ASK overtaking-maneuver)
    (ASK right-turn-maneuver))
```

9.4 Rulare

Dupa pornirea CLIPS, se va apela

```
(batch go)
```

Conținutul fișierului `go` este următorul:

```
(clear)
(printout t "---NEW------------------------------" crlf)
(load "headers.clp")
(load "MAIN-rules.clp")
(load "PERCEPT-MANAGER.clp")
(load "DRIVER-AGENT.clp")
(load "initials.clp")
(load "maneuverValidityASK.clp")
(reset)
(run 120)
```

9.5 Ieșirea sistemului

Dacă manevra de validat este depașirea (`overtaking`), ieșirea sistemului ar putea arăta astfel:

```
PERCEPT-MANAGER: timp = 1
      AGENT overtaking-maneuver allowed
PERCEPT-MANAGER: timp = 2
      AGENT overtaking-maneuver prohibited
PERCEPT-MANAGER: timp = 3
      AGENT overtaking-maneuver prohibited
PERCEPT-MANAGER: timp = 4
      AGENT overtaking-maneuver prohibited
PERCEPT-MANAGER: timp = 5
      AGENT overtaking-maneuver allowed
PERCEPT-MANAGER: timp = 6
      AGENT overtaking-maneuver prohibited
PERCEPT-MANAGER: timp = 7
      AGENT overtaking-maneuver allowed
PERCEPT-MANAGER: timp = 8
      AGENT overtaking-maneuver allowed
PERCEPT-MANAGER: timp = 9
      AGENT overtaking-maneuver prohibited
PERCEPT-MANAGER: timp = 10
      AGENT overtaking-maneuver prohibited
```

Aceasta înseamnă că, la momentele de timp 1, 5, 7 și 8 manevra este permisă, în rest este interzisă. Regulile din modulul AGENT trebuie scrise astfel încât ele să aserteze fapte de tip belief având valoarea `overtaking-maneuver` pentru `percept_pname` și `allowed/prohibited` pentru `percept_pvalue`.

Secvența de percepții corespunzătoare este:

```
t1.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval albina))

t2.clp:
(ag_percept
  (percept_pname road_sign)
  (percept_pval depasire_interzisa))
(ag_percept
  (percept_pname animal)
  (percept_pval barza))

t3.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval caine))

t4.clp:
```

```
(ag_percept
  (percept_pname animal)
  (percept_pval dropia))

t5.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval elefantul))
(ag_percept
  (percept_pname road_sign)
  (percept_pval incetarea_tuturor_restrictiilor))
(ag_percept
  (percept_pname road_surface_marking)
  (percept_pval linie_intre))

t6.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval furnicarul))
(ag_percept
  (percept_pname road_surface_marking)
  (percept_pval trecere_pietoni))

t7.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval gaina))

t8.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval hiena))

t9.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval iepurele))
(ag_percept
  (percept_pname road_surface_marking)
  (percept_pval trecere_pietoni))

t10.clp:
(ag_percept
  (percept_pname animal)
  (percept_pval jaguarul))
(ag_percept
  (percept_pname road_surface_marking)
  (percept_pval linie_cont))
```

Regula care gestionează tipărirea este adaugată la AGENT:

```
(defrule AGENT::tell
    (declare (salience -50))
    (timp (valoare ?)) ;make sure it fires each cycle
    (ASK ?bprop)
    ?fcvd <- (ag_bel (bel_type moment) (bel_timeslice 0)
      (bel_pname ?bprop) (bel_pval ?bval))
=>
    (printout t "         AGENT " ?bprop " " ?bval crlf)
    (retract ?fcvd)
)
```

și executarea sa este condiționată de prezența faptelor de tip ASK și a deciziilor corespunzătoare (decizii stocate in slotul `bel_pname` al unor fapte de tip `ag_bel`, generate pe baza regulilor implementate).

9.6 Exemplu de set de fapte inițiale (fișierul initials.clp)

Faptele inițiale ale sistemului sunt specificate aici. Exemplu:

```
(deffacts PERCEPT-MANAGER::timp
    (timp (valoare 0)))

(deffacts MAIN::succesiune-module
   (secventa PERCEPT-MANAGER AGENT))
```

9.7 Exerciții

Exercițiul 9.1 *Rulați exemplul pus la dispoziție și verificați corectitudinea răspunsurilor.*

Exercițiul 9.2 *Alcătuiti 5 secvențe distincte de test pentru manevrele date. Imaginile trebuie să fie consistente. Rulați programul pus la dispoziție și verificați corectitudinea răspunsurilor.*

Exercițiul 9.3 *Alcătuiți o nouă secvență de test selectând fiecare al doilea / al treilea fișier din una din secvențele de la Exercițiul 9.2. Rulați programul pus la dispoziție pe aceasta secvență și verificați corectitudinea răspunsurilor.*

Lab 10

Aplicație: Formalizare de reguli de circulație în CLIPS (II)

10.1 Modulul AGENT

Modulul AGENT implementează validarea manevrelor. Vom exemplifica cu o versiune simplă de manevră de depășire și de virare.

Faptele pe baza cărora un agent ia deciziile sunt intotdeuna niște credințe (”beliefuri”) și sunt descrise strict prin următorul template:

```
(deftemplate ag_bel
   (slot bel_type) ; fluent|moment
   (slot bel_timeslice) ; in which time slice bel_type is true:
                        ; 0=crt, 1,2,...=future
   (slot bel_pname) ; which property we’re talking about:
                    ; overtaking-maneuver|no-overtaking-zone|speed_limit
   (slot bel_pval) ; bel_pname value: (prohibited|allowed)
                   ; or (yes|no) or (50|90|100|130)
   (slot bel_pdir); left|right|ahead
)
```

Sloturile `bel_pname, bel_pval, bel_pdir` sunt similare celor pentru percepții. Slotul numit `bel_type` poate avea valorile `fluent` și `moment`. Un belief de tip fluent este un belief care persistă în timp de la un ciclu la altul. Un exemplu tipic este zona de acțiune a unui indicator ”depășirea interzisa”: un agent poate fi în aceasta zonă de acțiune, chiar dacă în momentul curent nu îl mai percepe. Belieful va fi șters doar în momentul întâlnirii unui indicator care anulează acțiunea primului (de exemplu, indicatorul ”încetarea tuturor restrictiilor”). Până în acel moment, belieful ramane în memoria de lucru și duce la interzicerea manevrei.

Un belief de tip moment este generat strict pe baza unei percepții curente și dispare o data cu aceasta. De exemplu, belieful ”marcaj linie continua”. Dispariția percepției duce imediat la dispariția acestui belief din memoria de lucru și în consecintă la dispariția instantanee a unuia din motivele de interzicere a manevrei de depășire.

Slotul `bel_timeslice` nu este folosit deocamdată. Scopul introducerii sale este de a permite scrierea unor reguli care să gestioneze reacțiile la percepții anticipate (de exemplu, dacă un semafor din plan îndepărtat tocmai comută de pe verde pe roșu, să putem modela decizia de a nu mai accelera.)

Agentul este singurul responsabil de gestionarea propriilor beliefuri. Regula următoare:

```
(defrule AGENT::hk-eliminate-momentan-current-bel
    (declare (salience -90))
    (timp (valoare ?)) ;make sure it fires each cycle
    ?fmcb <- (ag_bel (bel_type moment) (bel_timeslice 0)
                     (bel_pname ?p) (bel_pval ?v))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>hk-eliminate-momentan-current-bel "
            ?p " " ?v crlf))
    (retract ?fmcb)
)
```

se ocupă de ștergerea faptelor de tip belief momentan.

Fiecare manevră e validată pe modelul următor: se presupune inițial că nu este validă legal (la extiderea codului, se va porni de la acest model).

```
(defrule AGENT::initCycle-overtaking
 (declare (salience 99))
 (timp (valoare ?)) ;make sure it fires each cycle
=>
 (assert (ag_bel (bel_type moment) (bel_timeslice 0)
   (bel_pname overtaking-maneuver) (bel_pval prohibited)))
 ;by default, we assume overtaking NOT valid
)
```

Apoi, pe baza percepțiilor curente și a faptelor cunoscute anterior, se testează dacă vreuna din condițiile care fac manevra invalidă din punct de vedere legal este îndeplinită. Dacă niciuna din aceste condiții nu e îndeplinită, se va șterge faptul că manevra nu este validă și se aserta ca este validă legal

```
(defrule AGENT::validate-overtaking
    (declare (salience -10))
    ?f <- (ag_bel (bel_type moment) (bel_timeslice 0)
      (bel_pname overtaking-maneuver) (bel_pval prohibited))
    (not (ag_bel (bel_type fluent)
      (bel_pname no-overtaking-zone) (bel_pval yes)))
    (not (ag_bel (bel_type moment) (bel_timeslice 0)
      (bel_pname pedestrian-crossing-marking) (bel_pval yes)))
    (not (ag_bel (bel_type moment) (bel_timeslice 0)
      (bel_pname continuous-line-marking) (bel_pval yes)))
```

```
    ;TODO: de implementat
    ; intersectie
    ; rampa
    ; curba
    ; vizibilitate redusa
    ; pasaj
    ; pod
    ; sub pod
    ; tunel
    ; cale ferata curenta
    ; urmeaza cale ferata
    ; statie
    ; marcaj dublu continuu
    ; posibila coliziune
    ; coloana
    ; coloana_oficiala
=>
    (retract ?f)
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
      (bel_pname overtaking-maneuver) (bel_pval allowed)))
    ;(facts AGENT)
)
```

Cazurile #1 și #2 din DRIVER-AGENT exemplifică situațiile în care decizia se ia pe baza unui belief de tip fluent (care persistă în timp de la un ciclu la altul), respectiv strict a unei percepții curente. Fluentul este trecut din valoarea False în True la apariția semnului "depășire_interzisa"; trecerea din valoarea True în False se face fie la întâlnirea indicatorului "final_depășire_interzisa", fie la cea a indicatorului "încetarea_tuturor_restricțiilor". Agentul este cel care gestionează, pe baza regulilor scrise, aceste schimbări.

```
;---Case #1: a fluent with:
;             - 1 sign to turn it on and
;             - 2 different signs which might turn it off

(defrule AGENT::rdi "Sign: overtaking prohibited"
    (timp (valoare ?t))
    (ag_percept (percept_pname road_sign)
                (percept_pval depasire_interzisa))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>rdi vad indicator "
            depasire_interzisa crlf))
    (assert (ag_bel (bel_type fluent)
      (bel_pname no-overtaking-zone) (bel_pval yes)))
    ;(facts AGENT)
)
```

```
(defrule AGENT::frdi "Sign: overtaking prohibited end"
    (timp (valoare ?t))
    ?f <- (ag_bel (bel_type fluent)
                  (bel_pname no-overtaking-zone) (bel_pval yes))
    (ag_percept (percept_pname road_sign)
                (percept_pval final_depasire_interzisa))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>frdi vad indicator "
            final_depasire_interzisa crlf))
    (retract ?f)
    ;(facts AGENT)
)

(defrule AGENT::far "Sign: no restrictions"
    (timp (valoare ?t))
    ?f <- (ag_bel (bel_type fluent)
      (bel_pname no-overtaking-zone) (bel_pval yes))
    (ag_percept (percept_pname road_sign)
                (percept_pval incetarea_tuturor_restrictiilor))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>far vad indicator "
            tip incetarea_tuturor_restrictiilor crlf))
    (retract ?f)
    ;(facts AGENT)
)
```

Un belief de tip momentan există doar când există percepția.

```
;----Case #2: an non-fluent belief:
;             it depends on the current percepts only
;--- Pedestrian crossing mark perceived in the current moment
(defrule AGENT::rmtp
    (timp (valoare ?t))
    (ag_percept (percept_pname road_surface_marking)
                (percept_pval trecere_pietoni))
=>
    (if (eq ?*ag-in-debug* TRUE) then
         (printout t "    <D>rmtp vad marcaj "
                     trecere_pietoni crlf))
    (assert (ag_bel (bel_type moment)
                    (bel_timeslice 0)
                    (bel_pname pedestrian-crossing-marking)
                    (bel_pval yes)))
)
```

Dezvoltatorul va decide dacă un belief va fi de tip fluent sau moment. Un indiciu poate fi dat de durata fizică a unei cuante de timp din secvența dată. Astfel, o trecere de pietoni poate fi de tip fluent pentru o durată mare, dar fluent pentru o durată mica a cuantei de timp.

10.2 Exerciții

Exercițiile din acest laborator au ca bază aplicația schelet existentă, care va fi dezvoltată conform următoarelor cerințe:

1. Consistența temporală a percepțiilor. Fișierele de date contin "snapshoturi" ale percepțiilor obținute la diferite momente de timp. Este necesar ca percepțiile din momente succesive de timp să fie consistente, de pildă dacă la t=1 se vede un automobil în fata celui curent, nu e posibil ca la t=2 acesta nu mai fie prezent, iar la t=3 sa reapară.

2. Verificarea posibilității de a obține un fapt din memoria de lucru folosind niște metode cu performanțe credibile puse la dispoziție de tehnologia actuală. Dacă în memoria de lucru există percepția "semn de limitare de viteză", va trebui sa investigați dacă acel semn poate fi obținut automat. Daca se aplică un sistem de recunoaștere de imagini, se va căuta și cita în lista de referințe bibliografice un articol care să descrie un astfel de sistem și performanțele lui, ex. Dariu Gavrila, Traffic Sign Recognition Revisited, Proccedings of DAGM-Symposium, Bonn, 1999, pag. 86–93 (vezi și referința [2] din Bibliografie). Se vor specifica valorile pentru acuratețea, precizia, recallul al metodei. Dacă se poate adopta soluția unei harți de tipul OpenStreetMap [4, 5], se va indica modul în care se pot regăsi semnele de circulație dintr-o astfel de hartă, precizia localizarii GPS etc.

3. Extinderea descrierii datelor cu sloturi/valori care să conțină informație relevantă pentru îmbunătățirea deciziilor. De exemplu, dacă se justifică păstrarea de informații despre distanța la care se află un obstacol, vizibilitate etc. Fiecare slot și fiecare valoare posibilă pentru un slot se vor denumi în mod uniform. De exemplu, pagina [1] descrie numele și indicativele standard pentru semnele de circulație din Germania și poate fie exploatată. De asemenea, numele pentru orice manevră se va termina cu `-maneuver` (ex. overtaking-maneuver, right-turn-maneuver etc)

4. Adăugarea de reguli sau modificarea celor existente astfel încât să acopere cazuri noi. De exemplu, limita de viteză în condiții de gheață.

5. Evaluarea timpilor de răspuns în diferite scenarii. Se vor măsura timpii de raspuns în cazul repetării raționarii de mai multe ori.

Exercițiul 10.1 *Implementați validarea, din punct de vedere al regulamentului, a manevrei de virare dreapta (`right-turn-maneuver`), respectiv stânga (`left-turn-maneuver`).*

Exercițiul 10.2 *Identificați obiecte dintr-o scenă de obiecte care ar putea fi relevante pentru o anumită manevră la un moment de timp ulterior celui curent. Indicație: un semafor aflat la orizont are culoarea roșie.*

10.3 Soluții

Soluție exercitiul 10.1:

```
;;---------------------------------
;;
;;    Right turn
;;
;;---------------------------------
(defrule AGENT::initCycle-right-turn
    (declare (salience 99))
    (timp (valoare ?)) ;make sure it fires each cycle
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>initCycle-right-turn prohibited by default "
            crlf))
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
            (bel_pname right-turn-maneuver) (bel_pval prohibited)))
)

;
;--- Sign forbidding right turn or forcing either go ahead or left turn
;
(defrule AGENT::r-no-right-turn-sign
    (timp (valoare ?t))
    (ag_percept (percept_pname road_sign)
                (percept_pval ?v&interzis_viraj_dreapta |
                               obligatoriu_inainte |
                               obligatoriu_stanga |
                               obligatoriu_inainte_stanga))
=>
    (if (eq ?*ag-in-debug* TRUE)
     then (printout t "    <D>r-no-right-turn-sign" crlf))
    (assert (ag_bel (bel_type fluent)
                    (bel_pname no-right-turn-zone)
                    (bel_pval yes))))

(defrule AGENT::r-no-right-turn-zone-end
    (timp (valoare ?t))
    ?f <- (ag_bel (bel_type fluent)
                  (bel_pname no-right-turn-zone)
                  (bel_pval yes))
    (ag_percept (percept_pname intersection_end)
                (percept_pval yes))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>r-no-right-turn-zone-end
                       we crossed an intersection" crlf))
    (retract ?f)
)
```

```
;
;--- Sign forbidding access on a street
;
(defrule AGENT::r-no-access
    (timp (valoare ?t))
    (ag_percept (percept_pname road_sign)
                (percept_pval ?v& accesul_interzis |
                                  circulatia_interzisa_in_ambele_sensuri)
                (percept_pdir ?pd& right |
                                   left))
=>
    (if (eq ?*ag-in-debug* TRUE) then (printout t "    <D>r-no-access" crlf))
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
                    (bel_pname no-access) (bel_pval yes) (bel_pdir ?pd))))

;---Validate intention of right-turn: check if any restriction -------
(defrule AGENT::validate-right-turn
    (declare (salience -10))
    ?f <- (ag_bel (bel_type moment) (bel_timeslice 0)
                  (bel_pname right-turn-maneuver) (bel_pval prohibited))
    (not (ag_bel (bel_type fluent)
                 (bel_pname no-right-turn-zone) (bel_pval yes)))
    (not (ag_bel (bel_type moment) (bel_timeslice 0)
                 (bel_pname no-access) (bel_pval yes) (bel_pdir right)))
=>
    (if (eq ?*ag-in-debug* TRUE)
     then (printout t "    <D>validate-right-turn NU->DA
                        (nu avem restrictii) " crlf))
    (retract ?f)
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
                    (bel_pname right-turn-maneuver) (bel_pval allowed))))
```

```
;;---------------------------------
;;
;;    Left turn
;;
;;---------------------------------
(defrule AGENT::initCycle-left-turn
    (declare (salience 99))
    (timp (valoare ?)) ;make sure it fires each cycle
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>initCycle-left-turn prohibited by default "
             crlf))
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
            (bel_pname left-turn-maneuver) (bel_pval prohibited))))
```

```
;
;--- Sign forbidding left turn or forcing either go ahead or right turn
;
(defrule AGENT::r-no-left-turn-sign
    (timp (valoare ?t))
    (ag_percept (percept_pname road_sign)
                (percept_pval ?v&interzis_viraj_stanga |
                               obligatoriu_inainte |
                               obligatoriu_dreapta |
                               obligatoriu_inainte_dreapta |
                               intersectie_cu_sens_giratoriu))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>r-no-left-turn-sign" ?v crlf))
    (assert (ag_bel (bel_type fluent)
                    (bel_pname no-left-turn-zone)
                    (bel_pval yes)))
)

(defrule AGENT::r-no-left-turn-zone-end
    (timp (valoare ?t))
    ?f <- (ag_bel (bel_type fluent)
                  (bel_pname no-left-turn-zone)
                  (bel_pval yes))
    (ag_percept (percept_pname intersection_end)
                (percept_pval yes))
=>
    (if (eq ?*ag-in-debug* TRUE)
      then (printout t "    <D>r-no-left-turn-zone-end
                         we crossed an intersection" crlf))
    (retract ?f)
)

(defrule AGENT::r-continous-line-mark
    (timp (valoare ?t))
    (ag_percept (percept_pname road_surface_marking)
                (percept_pval linie_cont))
=>
    (if (eq ?*ag-in-debug* TRUE)
        then (printout t "    <D>rmlc vad marcaj " linie_cont crlf))
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
            (bel_pname continuous-line-marking) (bel_pval yes)))
)
```

```
;---Validate intention of left-turn: check if there is any restriction -----
(defrule AGENT::validate-left-turn
    (declare (salience -10))
    ?f <- (ag_bel (bel_type moment) (bel_timeslice 0)
                  (bel_pname left-turn-maneuver) (bel_pval prohibited))
    (not (ag_bel (bel_type fluent)
                 (bel_pname no-left-turn-zone) (bel_pval yes)))
    (not (ag_bel (bel_type moment) (bel_timeslice 0)
                 (bel_pname no-access) (bel_pval yes) (bel_pdir left)))
    (not (ag_bel (bel_type moment) (bel_timeslice 0)
                 (bel_pname continuous-line-marking) (bel_pval yes)))
=>
    (if (eq ?*ag-in-debug* TRUE)
     then (printout t "    <D>validate-left-turn (no restrictions) " crlf))
    (retract ?f)
    (assert (ag_bel (bel_type moment) (bel_timeslice 0)
                    (bel_pname left-turn-maneuver) (bel_pval allowed)))
)
```

Bibliografie

[1] German Wikipedia Contributors. *Bildtafel der Verkehrszeichen in der Bundesrepublik Deutschland seit 2017*, 2017 (accessed January 24, 2018). `https://de.wikipedia.org/wiki/Bildtafel_der_Verkehrszeichen_in_der_Bundesrepublik_Deu`

[2] Dariu Gavrila. Traffic sign recognition revisited. In *Proceedings of DAGM-Symposium, Bonn, Germany*, pages 86–93, 1999.

[3] Anca Nicoleta Marginean, Andra Petrovai, Radu Razvan Slavescu, Mihai Negru, and Sergiu Nedevschi. Enhancing digital maps to support reasoning on traffic sign compliance. In *IEEE 12th International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania*, pages 277–284, 2016.

[4] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org. `https://www.openstreetmap.org`, 2017.

[5] OpenStreetMap contributors. *OSM Map Features*, 2017 (accessed January 25, 2018). `https://wiki.openstreetmap.org/wiki/Map_Features`.