UNIVERSITAT ROVIRA i VIRGILI

KNOWLEDGE REPRESENTATION AND ENGINEERING

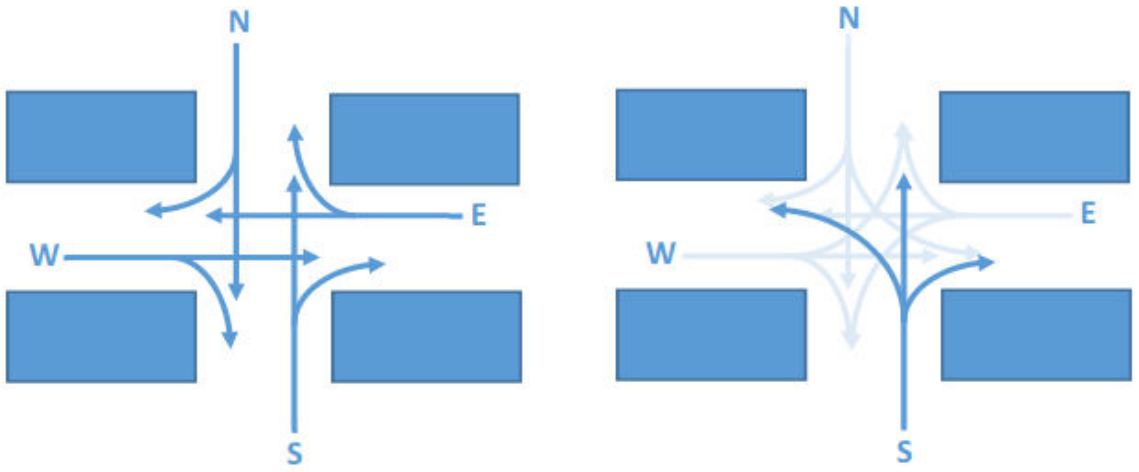
Intelligent Crossing Roads

Bartosz Paulewicz - bartosz.paulewicz@student.put.poznan.pl
Eros Zaupa - eros.zaupa@estudiantat.upc.edu

In the new era of the Internet of Things (IOT) and autonomous cars (AC), a change in the control of crossing roads is expected that will replace all traffic lights by peer-to-peer communication between cars and crossing road controls (CRC). With these new intelligent crossings, cars will arrive in one of four possible directions (North, South, East, or West), they will communicate their arrival, and will wait in line until they are given the way. Two types of CRCs are expected to coexist implementing two alternative preference policies:
(1) straight-right crossing (SRC) and (2) straight-right-left crossing (SRLC).

While, SRC allows a car to cross either following the same lane or turning to the right (see left hand figure where, for example, a car arriving from the north can continue to the south or right to the west), the SRLC control allows a car to cross following the same lane, but also turning either to the right or to the left (see the right hand figure where a car arriving from the south can continue to the north, to the east, or to the west).

System



Allowed trajectories of cars arriving from the North, South, East and West.

(1) Straight-Right Crossing (SRC) (2) Straight-Right-Left Crossing (SRLC)

SRLC control follows a “the first car arriving is the first car crossing” strategy, this meaning that cars cross in the strict order in which they communicate arrival to the crossing. In contrast, the SRC control rotates the crossing from North & South to East & West, in alternate turns, communicating to the N first cars in each two directions that they can cross. For example, if the turn is North & South, the first N cars waiting in the north, and the first N cars waiting in the south receive a permission to cross. If there are less than N cars waiting in one direction, only the orders to the waiting cars are sent. When all these cars have crossed, the round is given to the other two directions. When a car crosses, it notifies that to the system.

Implement the SRLC and the SRC as two separate sets of rules in a Rule Production System with CLIPS. Implement also the arrival of cars, randomly from the north, south, east, and west, and run both CRC systems separately, to check a correct behavior. Pay special attention to the representation of the tokens describing the different states of a car: car arrives, car is waiting to get permission to cross, car got permission to cross, and car has passed. Adjust the intelligent rules for the management of both systems according to these tokens and others that the system may require. Use printout messages to see the correct dynamic evolution of cars arriving and crossing. Please, pay attention to the priority of the rules implemented.

Expected documentation delivery:

1. One PDF document with (1) a main page with the name of all the members of the group and the title of the practice. The document must contain (2) a description of the knowledge base for each one of the two policies: straight-right crossing (SRC) and straight-right-left crossing (SRLC). For each policy (3) a separated capture of the execution showing how your knowledge performs correctly under both policies. Optionally, it could also contain a section (4) with your personal comments about pros and cons found during the development of the practice.
2. One text file with the code of each strategy so that the professor could upload and test your practice.

The two policies implement the following permission to cross rules
SRLC - FIFO
SRC
On rotation, permission goes either to cars from North & South or from East & West

- If it is the N&S turn, n cars from North and n cars from South receive the permission to go
- If there are m<n cars waiting from one direction, only m orders are sent
- If zero cars are waiting to cross, the turn goes to East & West (and vice versa)
- If a car crosses, it notifies the crossing to the system

Exercises
Implement a set of rules of in CLIPS for SRLC
Implement a set of rules of in CLIPS for SRC
Implement the random arrival of cars from any direction to check the correct behaviour in SRLC and SRC
Delivery
PDF document with
Main page
Members of the group + Practice title
Description of the KB
For both SRC and SRLC
Capture of correct execution
For both SRC and SRLC
Text file with the code for both SRC and SRLC
Useful resources
Witold Paluszynski - CLIPS tutorial

Knowledge base

Straight-right crossing (SRC)

Global variables

- **nCars** - number of cars to generate, **20** by default
- **N** - maximum number of cars passed from each direction in a turn, **2** by default

Templates

- **Car**
  - **id** - unique identifier
  - **from** - random source direction mapped to integer from set {0, 1, 2, 3}
  - **to** - random target direction based on source direction, either straight or right
  - **arrival_time** - time of arrival
- **Counter**
  - **symbol -** random direction mapped to integer from set {0, 1, 2, 3}
  - **value -** number of cars that already passed from given direction
- **Turn**
  - **symbol -** random direction mapped to integer from set {0, 1, 2, 3}

Functions

- **Mapping direction integer to symbol**

```
(deffunction int2symbol (?int)
     (switch ?int
          (case 0 then (bind ?symbol N))
          (case 1 then (bind ?symbol W))
          (case 2 then (bind ?symbol S))
          (case 3 then (bind ?symbol E))
     )
     (return ?symbol))
```

- **Mapping direction symbol to integer**

```
(deffunction symbol2int (?symbol)
     (switch ?symbol
          (case N then (bind ?int 0))
          (case W then (bind ?int 1))
          (case S then (bind ?int 2))
          (case E then (bind ?int 3))
     )
     (return ?int))
```

Rules

- **Crossing**

```
(defrule ruleBase
      ; it's right turn
      (turn (symbol ?from))
      ; less than N cars already passed from given direction
      ?counter <- (counter (symbol ?from) (value ?v&:(< ?v ?*N*)))
      ; car from given direction
      ?car <- (car (from ?from) (to ?to) (arrival_time ?a))
      ; no cars from given direction that arrived before
      (not (car (from ?from) (arrival_time ?a2&:(< ?a2 ?a))))
=>
      ; car passed so remove it from system
      (retract ?car)
      ; modify number of cars passed
      (modify ?counter (value (+ ?v 1))))
```

- **Change the turn**

```
(defrule ruleTurn
      ; either a)North or b)West turn
      ?turn1 <- (turn (symbol ?from1))
      ; either a)South or b)East turn
      ?turn2 <- (turn (symbol ?from2&:(=
            (+ ?from1 2)
            ?from2
      )))
      ; number of cars that already passed from given directions
      ?counter1 <- (counter (symbol ?from1) (value ?v1))
      ?counter2 <- (counter (symbol ?from2) (value ?v2))
      ; from given directions either no cars left or N already passed
      (or (test (= ?v1 ?*N*)) (not(car (from ?from1))))
      (or (test (= ?v2 ?*N*)) (not(car (from ?from2))))
      ; there is a waiting for the turn change
      (car (from ?from3&:(neq (mod ?from1 2) (mod ?from3 2))))
=>
      ; new turn directions
      (bind ?from3 (+ ?from1 1))
      (bind ?from4 (mod (+ ?from1 3) 4))
      ; reset number of cars passed from given directions
      (modify ?counter1 (value 0))
      (modify ?counter2 (value 0))
      ; change turns
      (modify ?turn1 (symbol ?from3))
      (modify ?turn2 (symbol ?from4)))
```

- **Cars in one group but no cars in the other group**

```
(defrule ruleEndCase
      ; car from group North-South or West-East
      ?car <- (car (from ?from) (to ?to) (arrival_time ?a))
      ; no cars from other group
      (not(car (from ?from2&:(neq (mod ?from1 2) (mod ?from2 2)))))
=>
      ; car passed so remove it from system
      (retract ?car))
```

Straight-right-left crossing (SRLC)

Variables

- **seed** - Is set to guarantee reproducibility

Templates

- **crc**
  - **policy** - (0=SRLC, 1=SRC), is used when generated random traffic
  - **incars** - how many cars are generated at a time
  - **outcars** - how many cars cross at a time
  - **turn** - keep track of the turn
  - **time** - keep track of the time
  - **tflag** - when true, enables the traffic generation rule
  - **dflag** - when true, enables the crossing of cars
  - **dcount** - counter used for departures
  - **ttx** - time to cross, how long does it take for a car to cross
- **car**
  - **from** - random source direction mapped to integer
  - **to** - random target direction based on source direction, supports both SRLC and SRC policies
  - **arrival_time** - time of arrival
  - **departure_order** - departure order

Functions

- **Mapping direction integer to symbol**

```
(deffunction getdirection (?int)
      (switch ?int
            (case 0 then (bind ?symbol N))
            (case 1 then (bind ?symbol W))
            (case 2 then (bind ?symbol S))
            (case 3 then (bind ?symbol E))
      )
      (return ?symbol))
```

Rules

- **Traffic generation**

```
(defrule gentraffic
  ?crc <- (crc (policy ?policy) (incars ?incars) (time ?t) (tflag TRUE))
=>
  (loop-for-count (?i 1 ?incars) do
      (bind ?newtime (+ ?i ?t))
      (bind ?from (random 0 3)) ; From any direction
      (switch ?policy ; To direction allowed by the CRC policy
      (case 0 then (bind ?to (mod (+ ?from (random 1 3)) 4)))
      (case 1 then (bind ?to (mod (+ ?from (random 1 2)) 4)))
      )
      (assert (car (from (getdirection ?from)) (to (getdirection ?to))
(arrival_time ?newtime)))
```

```
      (printout t "car arrives at time " ?newtime " from " (getdirection
?from) ", direction " (getdirection ?to) crlf)
  )
  ; Update the time and prevent the turn the flag for traffic generation off
  (modify ?crc (time (+ ?t ?incars)) (tflag FALSE))
)

; Assert the CRC settings
(deffacts crc "Overall settings of the CRC system"
  (crc (policy 0) (incars 10) (outcars 10))
)
```

- **CRC settings**

```
(deffacts crc "Overall settings of the CRC system"
  (crc (policy 0) (incars 10) (outcars 10))
)
```

- **Crossing**

```
(defrule crossing_srlc
  ; if there is a car that is waiting to departe
  ?car <- (car (from ?f) (to ?t) (arrival_time ?a) (departure_order nil))
  ; and such a car arrived before any other car waiting (FIFO)
  (not (car (arrival_time ?a2&:(< ?a2 ?a)) (departure_order nil)))
  ; and the crc settings are such that
  ; 1. the departure flag is on
  ; 2. the departure counter hasn't reach the maximum number of cars that can
cross
  ?crc <- (crc (policy ?policy) (outcars ?outcars) (time ?ti) (turn ?tv)
(dflag TRUE) (dcount ?dc&:(< ?dc ?outcars)) (ttx ?ttx))
=>
  (printout t "turn " ?tv ": car arrived at time " ?a " departs from " ?f "
and goes " ?t crlf)
  ; set the departure order
  (modify ?car (departure_order ?tv))
  (if (= (+ 1 ?dc) ?outcars)
      then ; if the departure counter reaches the max, set the departure flag
off and reset the counter
      (modify ?crc (time (+ ?ttx ?ti)) (turn (+ 1 ?tv)) (dflag FALSE) (dcount
0))
      else ; otherwise, increase the departure counter
      (modify ?crc (time (+ ?ttx ?ti)) (turn (+ 1 ?tv)) (dcount (+ 1 ?dc)))
  )
)
```

Captured execution

Straight-right crossing (SRC)

Initial procedure

```
; number of cars passed from North/West/South/East in current turn
(assert (counter (symbol (symbol2int N)) (value 0)))
(assert (counter (symbol (symbol2int W)) (value 0)))
(assert (counter (symbol (symbol2int S)) (value 0)))
(assert (counter (symbol (symbol2int E)) (value 0)))

; from directions allowed in current turn, initially North and South
(assert (turn (symbol (symbol2int N))))
(assert (turn (symbol (symbol2int S))))

; generating nCars cars
(loop-for-count (?i 1 ?*nCars*) do
     ; random int from set {0, 1, 2, 3}
     (bind ?from (random 0 3))
     ; from + [1(right) | 2(straight)]
     (bind ?to (mod (+ ?from (random 1 2)) 4))
     ; add new car
     (assert (car (from ?from) (to ?to) (arrival_time ?i))))
```

Results

```
Car8 passed from S to N
Car10 passed from S to N
Car2 passed from N to W
Car17 passed from N to S
Changed turn from NS to WE
Car4 passed from E to W
Car5 passed from E to N
Car1 passed from W to E
Car3 passed from W to E
Changed turn from WE to SN
Car11 passed from S to N
Car16 passed from S to N
Changed turn from NS to WE
Car6 passed from E to N
Car7 passed from E to N
Car9 passed from W to S
Car12 passed from W to S
Changed turn from WE to SN
Car20 passed from S to N
Changed turn from NS to WE
Car13 passed from E to W
Car14 passed from E to N
Car18 passed from W to S
No cars from other group so Car19 passed from E to N
No cars from other group so Car15 passed from E to W
```

Straight-right-left crossing (SRLC)

Results

```
CLIPS> (facts)

f-0   (initial-fact)

For a total of 1 fact.

CLIPS> (reset)

CLIPS> (facts)

f-0   (initial-fact)

f-1   (crc (policy 0) (incars 10) (outcars 10) (turn 1) (time 0) (tflag TRUE)
(dflag FALSE) (dcount 0) (ttx 1))

For a total of 2 facts.

CLIPS> (run)

car arrives at time 1 from N, direction W

car arrives at time 2 from N, direction W

car arrives at time 3 from E, direction W

car arrives at time 4 from W, direction N

car arrives at time 5 from W, direction N

car arrives at time 6 from S, direction N

car arrives at time 7 from N, direction W

car arrives at time 8 from N, direction W

car arrives at time 9 from N, direction E

car arrives at time 10 from W, direction E

CLIPS> (facts)

f-0   (initial-fact)

f-2   (car (from N) (to W) (arrival_time 1) (departure_order nil))

f-3   (car (from N) (to W) (arrival_time 2) (departure_order nil))

f-4   (car (from E) (to W) (arrival_time 3) (departure_order nil))
```

```
f-5   (car (from W) (to N) (arrival_time 4) (departure_order nil))

f-6   (car (from W) (to N) (arrival_time 5) (departure_order nil))

f-7   (car (from S) (to N) (arrival_time 6) (departure_order nil))

f-8   (car (from N) (to W) (arrival_time 7) (departure_order nil))

f-9   (car (from N) (to W) (arrival_time 8) (departure_order nil))

f-10  (car (from N) (to E) (arrival_time 9) (departure_order nil))

f-11  (car (from W) (to E) (arrival_time 10) (departure_order nil))

f-12  (crc (policy 0) (incars 10) (outcars 10) (turn 1) (time 10) (tflag
FALSE) (dflag FALSE) (dcount 0) (ttx 1))

For a total of 12 facts.

CLIPS> (modify 12 (dflag TRUE))

<Fact-13>

CLIPS> (run)

turn 1: car arrived at time 1 departs from N and goes W

turn 2: car arrived at time 2 departs from N and goes W

turn 3: car arrived at time 3 departs from E and goes W

turn 4: car arrived at time 4 departs from W and goes N

turn 5: car arrived at time 5 departs from W and goes N

turn 6: car arrived at time 6 departs from S and goes N

turn 7: car arrived at time 7 departs from N and goes W

turn 8: car arrived at time 8 departs from N and goes W

turn 9: car arrived at time 9 departs from N and goes E

turn 10: car arrived at time 10 departs from W and goes E

CLIPS> (facts)

f-0   (initial-fact)

f-14  (car (from N) (to W) (arrival_time 1) (departure_order 1))

f-16  (car (from N) (to W) (arrival_time 2) (departure_order 2))

f-18  (car (from E) (to W) (arrival_time 3) (departure_order 3))
```

f-20 (car (from W) (to N) (arrival_time 4) (departure_order 4))

f-22 (car (from W) (to N) (arrival_time 5) (departure_order 5))

f-24 (car (from S) (to N) (arrival_time 6) (departure_order 6))

f-26 (car (from N) (to W) (arrival_time 7) (departure_order 7))

f-28 (car (from N) (to W) (arrival_time 8) (departure_order 8))

f-30 (car (from N) (to E) (arrival_time 9) (departure_order 9))

f-32 (car (from W) (to E) (arrival_time 10) (departure_order 10))

f-33 (crc (policy 0) (incars 10) (outcars 10) (turn 11) (time 20) (tflag FALSE) (dflag FALSE) (dcount 0) (ttx 1))

For a total of 12 facts.