UNIVERSITAT ROVIRA i VIRGILI

KNOWLEDGE REPRESENTATION AND ENGINEERING

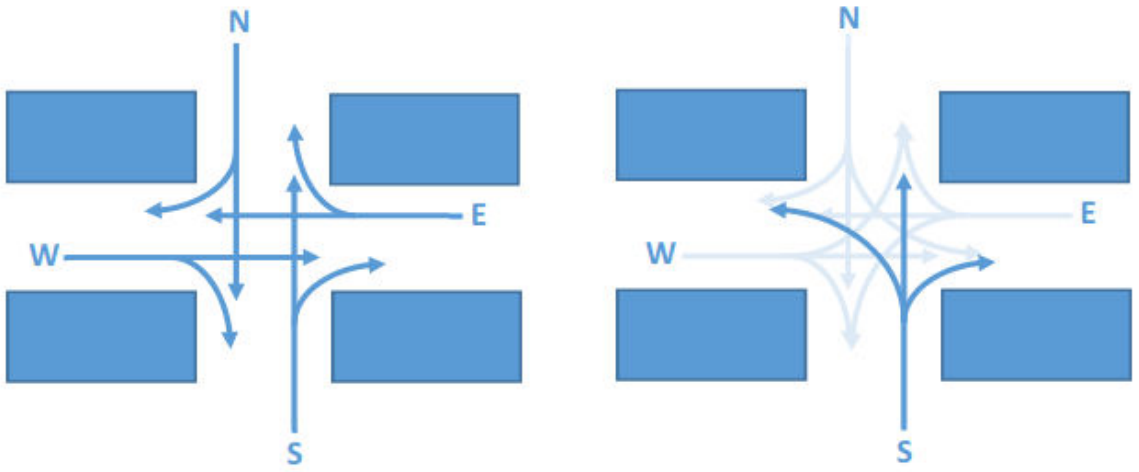
Intelligent Crossing Roads

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, **40** 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, (from + [1 (right) | 2 (straight)]) % 4
  - **queued** - whenever car is already queued, **false** by default
  - **last** - whenever car is last in its queue, **false** by default
  - **car-before** - car waiting in a queue before a car, **nil** by default
- **Counter**
  - **symbol -** random direction mapped to integer from set {0, 1, 2, 3}
  - **value -**
- **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

- **Queue up first car in a queue**

```
(defrule queue-first
     ; car not queued up yet
     ?car <- (car (id ?id) (from ?from) (queued false))
     ; no cars in a queue
     (not (car (from ?from) (queued true)))
=>
     ; queue up car as last in a queue
     (modify ?car (queued true) (last true)))
```

- **Queue up next car in a queue**

```
(defrule queue-next
     ; car not queued up yet
     ?car <-   (car (id ?id) (from ?from) (queued false))
     ; current last car in a queue
     ?last <-   (car (id ?lastid) (from ?from) (last true))
=>
     ; queue up car as last in a queue and set car before
     (modify ?car (queued true) (last true) (car-before ?lastid))
     ; previous last car is not last anymore
     (modify ?last (last false)))
```

- **Crossing**

```
(defrule ruleBase
     ; base rule so higher priority
     (declare (salience 10))
     ; no cars waiting to be queued up
     (not (car (queued false)))
     ; it's right turn
     (turn (symbol ?from))
     ; first car in a queue
     ?car <- (car (id ?id) (from ?from) (to ?to)
                (queued true) (car-before nil))
     ; less than N cars already passed from given direction
     ?counter <- (counter (symbol ?from) (value ?v&:(< ?v ?*N*)))
=>
     ; car passed so remove it from system
     (retract ?car)
     ; modify number of cars passed
     (modify ?counter (value (+ ?v 1))))
```

- **Clean up car-before**

```
(defrule cleanup-car-before
     ; car with car-before defined
     ?car <- (car (car-before ?id&:(neq ?id nil)))
     ; car-before doesn't exist anymore
     (not (car (id ?id)))
=>
     ; reset car-before field
     (modify ?car (car-before nil)))
```

- **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 other group**

```
(defrule ruleEndCase
      ; car from group North-South or West-East
      ?car <- (car (id ?id) (from ?from1) (to ?to))
      ; 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)

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))))
```

Results

```
gen13 comes first from S
gen12 comes after gen13
gen5 comes after gen12
gen4 comes after gen5

gen11 comes first from N
gen10 comes after gen11

gen9 comes first from E
gen6 comes after gen9
gen2 comes after gen6

gen8 comes first from W
gen7 comes after gen8
gen3 comes after gen7
gen1 comes after gen3

gen13 pass from S to N
gen12 pass from S to E
gen11 pass from N to W
gen10 pass from N to W

Changed turn from NS to WE
gen9 pass from E to N
gen8 pass from W to E
gen7 pass from W to E
gen6 pass from E to N

Changed turn from WE to SN
```

```
gen5 pass from S to E
gen4 pass from S to E

Changed turn from NS to WE
gen3 pass from W to E
gen2 pass from E to N

No cars from other group so gen1 pass from W to E
```

Straight-right-left crossing (SRLC)