***Simulation project 2014:***

***Members:***

* *Franco, Federico*
* *Gastaldi, Diego*

***Semaphore Module:***

* Parameters:
  + time\_green: Real+ (semaphore time in green).
  + time\_red: Real+ (semaphore time in red).
  + time\_yellow: Real+ (semaphore time in yellow).
  + Initial\_color: {0, 1, 2}
* Model:
  + S = R+  >< {red (0), yellow (1), green (2)} => (clock, next\_color)
  + Y = {green (2), yellow (1), red (0)}

(time\_red, green) if (next\_color == red)

* + dint(sigma, next\_color) = (time\_green, yellow) if (next\_color == green)

(time\_yellow, red) if (next\_color == yellow)

* + ta(sigma, color) = sigma
  + Lambda(sigma, next\_color) = next\_color
* Initial state: (time\_of\_initial\_color, initial\_color)

***Córner Module:***

* Parameters:
  + size\_cars: R+ (size of the cars in meters).
  + speed\_cars: R+ (speed of the cars in meters/seconds).
  + size\_corner: R+ (size of the corner in meters).
  + out\_percent0: R+ (percentage of output on port 0).
  + out\_percent1: R+ (percentage of output on port 1) (not used)
* Model:
  + S = R+ >< [R+] >< R+ => amount\_cars >< [distance] >< sigma
  + X = {0, 1} => port
  + Y = {1} >< {0, 1}

Error if (((a\_cars + 1) \* size\_cars > size\_corner) && ((a\_cars == 0)))

* dext((amount\_cars, list, s), e, p) = (amount\_cars, update\_list (list, e), s-e) if (amount\_cars >= 1 ^ (amount\_cars+1)\*size\_cars < size\_corner ^

size\_corner- head (update\_list(list)) >= size\_cars)

(amount\_cars+1, update\_list (list, e).push\_front(size\_corner), s-e) if (amount\_cars >= 1 ^ (amount\_cars+1)\*size\_cars < size\_corner ^ size\_corner-head (update\_list(list)) >= size\_cars)

(1, size\_corner, size\_corner/ speed\_cars) if (amount\_cars == 0)

Error if (amount\_cars <= 0)

* + dint((amount\_cars, list, s)) = (amount\_cars-1, update\_list(pop\_back (list), s), list.back()/speed\_cars)

if (amount\_cars >=2)

(0, empty, infinite) if (amount\_cars ==1)

* + ta((amount\_cars, list, s)) = s

(1, 0) if (random (100) <= out\_percent0)

* + Lambda((amount\_cars, list, s)) =

(1, 1) otherwise

* Notes:
  + As all cars have the same speed, the first to enter the corner will be the first out.
* Initial state: (0, empty, infinite)

***Street Module:***

* Parameters:
  + size\_cars: R+ (size of the cars).
  + speed\_cars: R+ (speed of the cars).
  + size\_street: R+ (size of the street).
  + Output: bool (indicate if the cars can out of the street)
* Model:
  + S = R+ >< [R+] >< R+ >< Bool=> (amount\_street, [distance], sigma, output)
  + X = {1} >< {Green (2), red (0), yellow (1)}

The Zero input port (semaphore input)

The One input port (car input)

* + Y = {1} >< {0}

(a\_street, update (dist, e, o), dist.back()/speed\_cars, true) if ((xp == 0) ^ (xv == green) ^ (a\_street > 0))

(a\_street, dist, infinite, xv == green) if ((xp == 0) ^ (a\_street == 0))

* + dext ((a\_street, dist, s, o), e, (xv, xp)) = (a\_street, update (dist, e, o), infinite, false) if ((xp == 0) ^ (xv != green) ^ (a\_street > 0))

(a\_street+1, update (dist, e, o).push\_front(size\_street), s-e, o) if ((xp == 1) ^ (a\_street >= 1) ^ (a\_street+1)\*size\_cars<=size\_street ^ (size\_street –(update (dist, e, o)).front() >= size\_cars)

Error if ((xp == 1) ^ (a\_street+1)\*size\_cars>size\_street)

(a\_street, update (dist, e, o), s-e, o) if ((xp == 1) ^ ((a\_street+1)\*size\_cars<=size\_street) ^ (a\_street > 0) ^ (size\_street –last (update (dist, e, o)) < size\_cars)

(1, size\_street, size\_street/speed\_cars, o) if ((xp == 1) ^ (a\_street == 0) ^ o)

(1, size\_street, s, o) if ((xp == 1) ^ (a\_street == 0) ^! o)

(a\_street - 1, update ((dist).pop\_back(), s, o), dist.back() /speed\_cars, o) if (o ^ (a\_street >= 2))

(0, empty, infinite, o) if (o ^ ((a\_street == 1) || (a\_street == 0))

* + dint((a\_street, dist, s, o)) =

error if (!o || a\_street <= 0)

(0, empty, infinite, o) if (output && (amount\_street < 0))

* + ta((a\_street, dist, s, o)) = s
  + Lambda((a\_street, dist, s, o)) = 1
* Initial state: (0, empty, infinite, output)

***Generator Module:***

* Parameters:
  + Paramteter\_distribution: Real+
* Model:
  + S = {Real+} -> (sigma)
  + Y = {1}
  + Dint(sigma) = sigma = exponential (parameter\_distribution)
  + ta((sigma)) = sigma
  + Lambda((sigma)) = 1

**Alternative proposals:**

* The speed in the corner are:
  + Corner 1: 8.3 m/s because if it is greater than the speed of the conflicting street, ignore many cars on this street.
  + Corner 2: between 8.3 and 18, because if less than 8.3 many cars on the corner would be ignored (by the speed of the streets), and if greater than 18, would be ignored when entering 3rd Street.
* The speed in the street are:
  + Street 1: 8.3 m/s unmodified
  + Street 2: 8.3 m/s unmodified
  + Street 3: 18 m/s unmodified
  + Street 4: 8.3 m/s unmodified

With these speeds, the system does not collapse but many cars will be ignored in the corner 1 because here the speed is less than the street 3.

With these parameters:

* On the street 1 the average time is 19.4.
* On the street 3 the average time is 26.6.
* On the street 4 the average time is 20.4.
* On the street 2 the average time is 14.1.

If we change the times of the semaphores (red 10s, yellow 1s and green 9s), then we get:

* On the street 1 the average time is 15.5.
* On the street 3 the average time is 17.9.
* On the street 4 the average time is 14.8.
* On the street 2 the average time is 14.6.

With these changes we reduce the time that takes to cars get off the streets. This is because (considering the frequency of car) the stopped cars need less time than the green.

But if the frequency of cars increases, it will be necessary to increase the time of the semaphore.