Asynchronous Testing of Synchronous Components in GALS Systems

Lina Marsso Radu Mateescu
Ioannis Parissis Wendelin Serwe

Team CONVECS (LIG / Inria)

iFM 2019, Bergen, Norway

GALS Systems

- GALS (Globally Asynchronous Locally Synchronous)
  - ✓ complex and critical systems
  - ✓ examples: Internet of things, autonomous cars

© inria smart home

© inria team CHROMA

  - ✓ difficult to test & debug
- Testing GALS systems
  - ✓ rigorous approach based on formal methods
  - ✓ combination of synchronous and asynchronous approaches

Proposed Solution

Leverage conformance test generation for asynchronous systems to automatically derive realistic test scenarios for synchronous components

- Integration of
  - ✓ synchronous and asynchronous *concurrent models*
  - ✓ functional unit testing and behavioral conformance testing
  - ✓ various formal methods and their tools

GRL LNT BCG XTL

Lustre V6

Outline

- Running Example
- Asynchronous validation (CADP)
  - ✓ model checking
  - ✓ conformance testing
- Synchronous testing techniques (Lustre v6)
- Derivation of test scenarios (CADP and Lustre v6 integration)
- Conclusion

GALS Example : Autonomous Car

C

B

D

A

A: Action

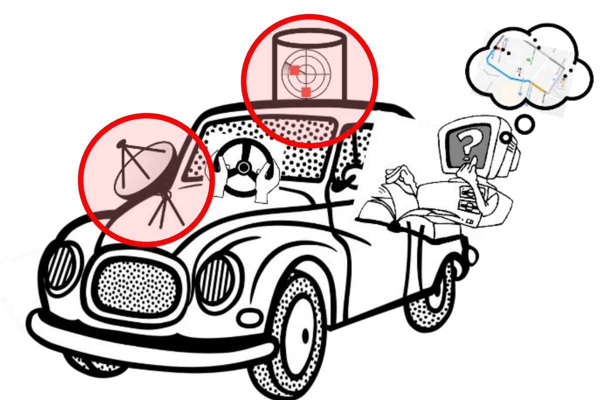
B: GPS

C: Radar

D: Decision

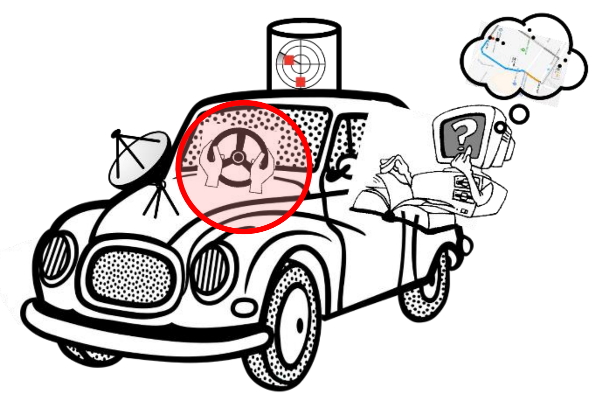
Perception

- GPS
  - ✓ keeps the car position updated
  - ✓ sends the localization upon request
- Radar
  - ✓ detects the presence of the obstacles
  - ✓ builds the radar grid with obstacle information
  - ✓ sends periodically the radar grid to the controller



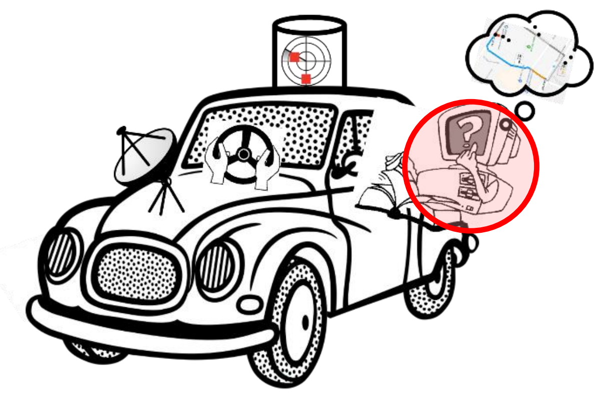
Action (Engine & Direction Command)

- Analyzes the radar grid and reacts if needed
- Asks the trajectory controller for an itinerary with **some constraints** (e.g., streets to avoid)
- Controls the car (go straight, brake, right, left)



Decision (Trajectory Controller)

- Knows the desired destination
- Upon request from the controller
  - ✓ requests the localization from the GPS
  - ✓ computes an itinerary respecting the **constraints**
  - ✓ sends the itinerary to the controller



Physical Environment

Coronation Street

Corporation Street

Sackville Street

Spring Gardens

ncess Street

New Cathedral Street

Model in GALS Representation Language (GRL)

GPS
UPDATE_POSITION
MAP_MANAGEMENT
POSITIONS
RADAR
SEND_POSITION
REQUESTED_POSITION
ARRIVAL
CAR_MOVE
SEND_CURRENT_GRID
POSITION
CURRENT_GRID
RECEIVE_POSITION
REQUEST_POSITION
RECEIVE_CURRENT_GRID
PATH_AVOIDING
RECEIVE_PATH
DECISION
PATH
ACTION
SEND PATH
REQUEST PATH

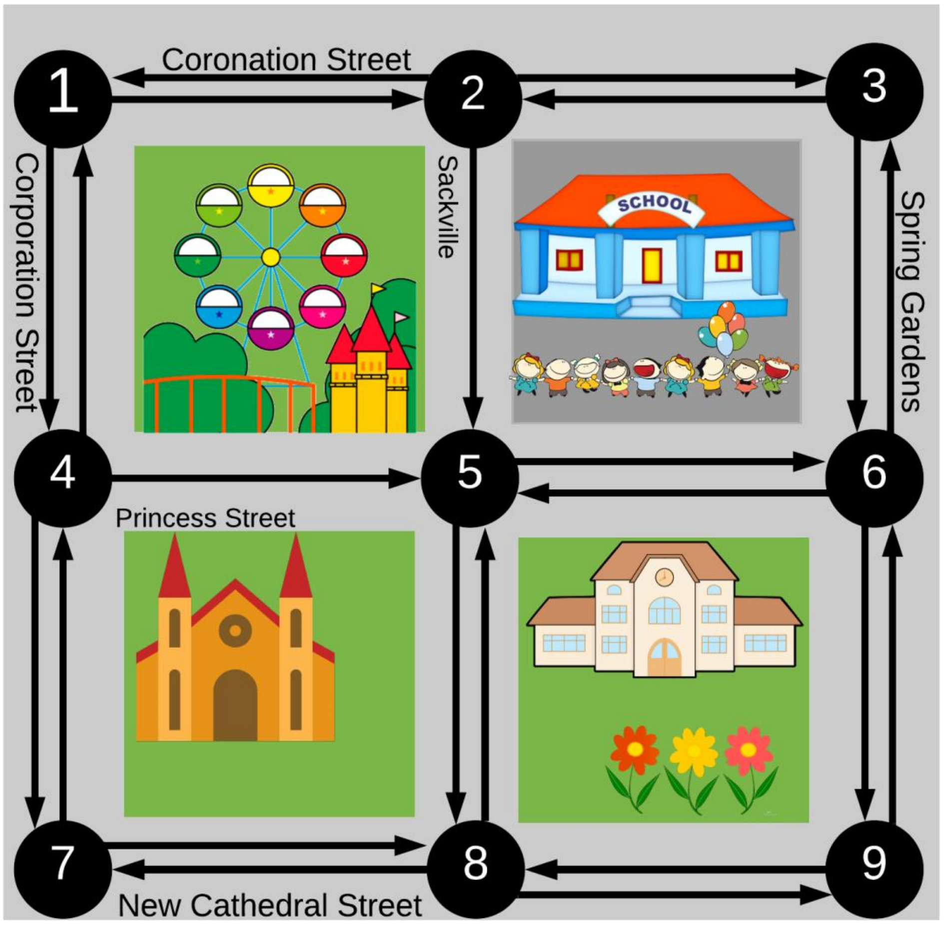
block — synchronous components, deterministic

medium — asynchronous communication between two blocks

environment — data constraints, block activations

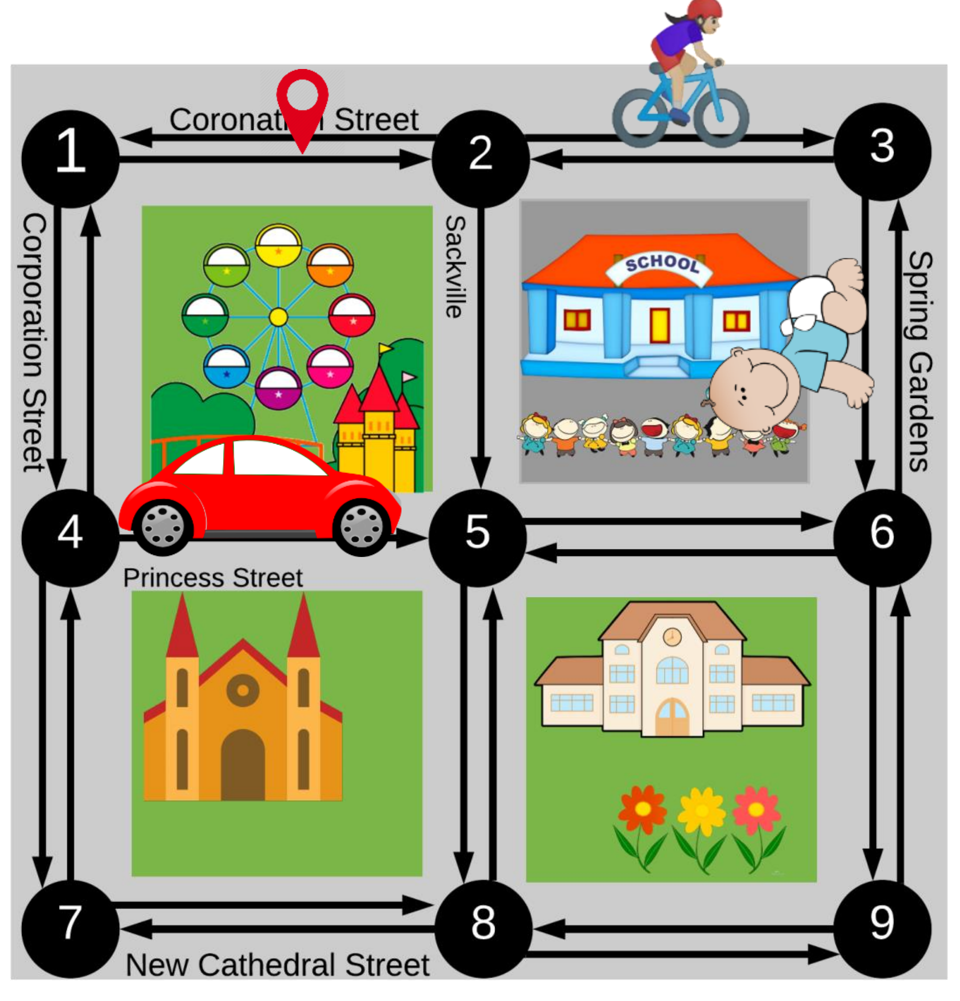
channels

Geographical Map Representation




```
function initial_map : Graph is
  var e: Edges, v: Vertices in
    v := {0, 1, 2, 3, 4, 5, 6, 7, 8};
    e := {Edge (0, Coronation_Street, 1),
          Edge (0, Corporation_Street, 3),
          Edge (1, Coronation_Street_bis, 0),
          Edge (1, two_Coronation_Street, 2),
          Edge (1, Sackville, 4),
          Edge (2, two_Coronation_Street_bis, 1),
          Edge (2, Spring_Gardens, 5),
          Edge (3, Corporation_Street_bis, 0),
          Edge (3, Princess_Street, 4),
          Edge (3, two_Corporation_Street, 6),
          Edge (4, two_Princess_Street, 5),
          Edge (4, two_Sackville, 7),
          Edge (5, Spring_Gardens_bis, 2),
          Edge (5, two_Princess_Street_bis, 4),
          Edge (5, two_Spring_Gardens, 8),
          Edge (6, two_Corporation_Street_bis, 3),
          Edge (6, New_Cathedral_Street, 7),
          Edge (7, two_Sackville_bis, 4),
          Edge (7, New_Cathedral_Street_bis, 6),
          Edge (7, two_New_Cathedral_Street, 8),
          Edge (8, two_Spring_Gardens_bis, 5),
          Edge (8, two_New_Cathedral_Street_bis, 7)
          };
    return Graph (v, e)
  end var
end function
```

Environment Configuration

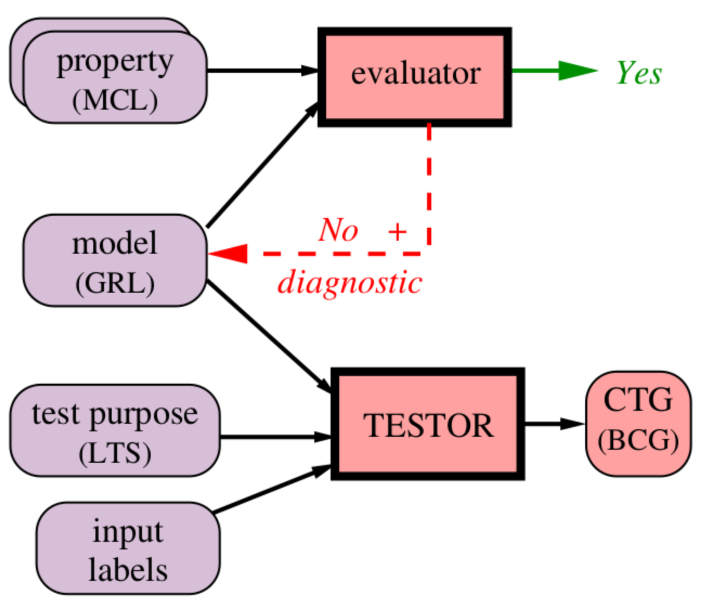


- Obstacle actions
  - ✓ turned_n (N: Nat)
  - ✓ leave
  - ✓ random (possible choices)
- Lilly
  - ✓ appears
  - ✓ 2 actions

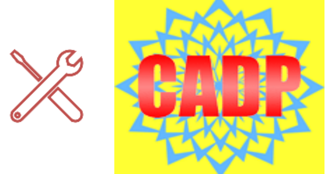
- Leo
  - ✓ appears
  - ✓ 2 actions

1189 lines, 287103 states, 406780 transitions

Asynchronous Validation



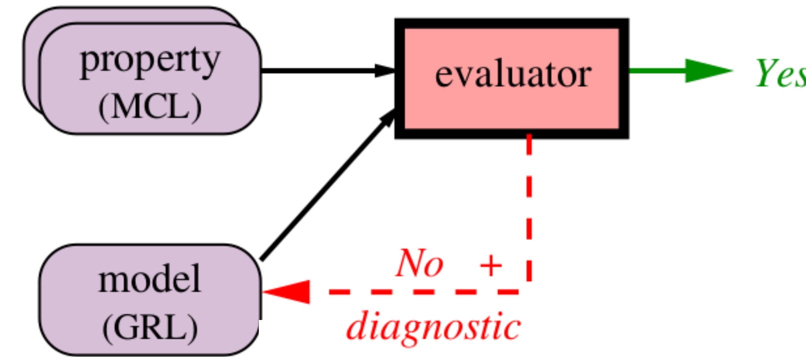
- **Model checking**
- **Conformance testing**



https://cadp.inria.fr/

Model Checking

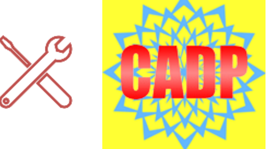
- Properties in MCL
  (**M**odel **C**hecking **L**anguage)



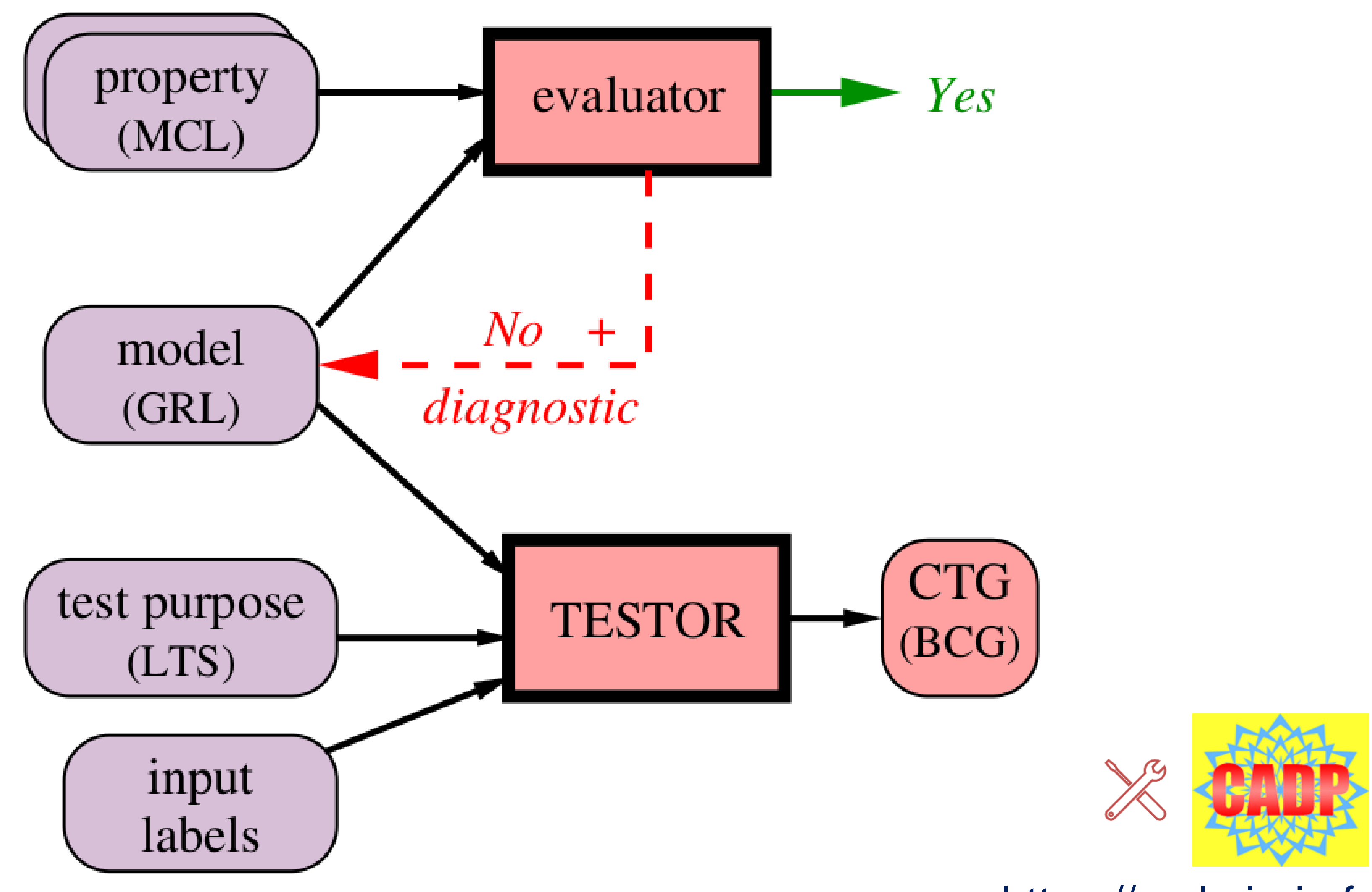
- The position of the car is correctly updated after any movement of the car

```
[ true* .
    { UPDATE_POSITION ?current_street:String } .
    (not ({ CAR_MOVE ... } or { UPDATE_POSITION ... }))* .
    { CAR_MOVE ?control:String } .
    (not ({ CAR_MOVE ... } or { UPDATE_POSITION ... }))* .
    { UPDATE_POSITION ?new_street:String where
     not (Consistent_Move (current_street, control, new_street)) }
] false ;
```

- Duplicate messages from one of the components must be considered only once
- The system should inevitably reach a final state



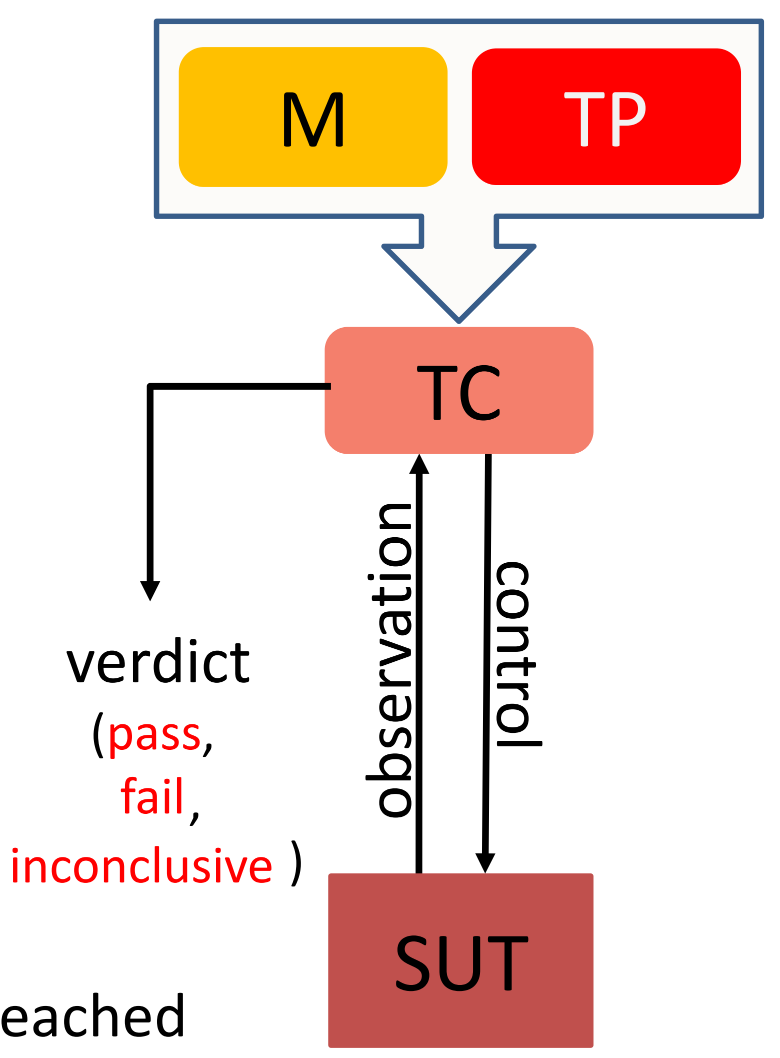
Conformance Testing



https://cadp.inria.fr/

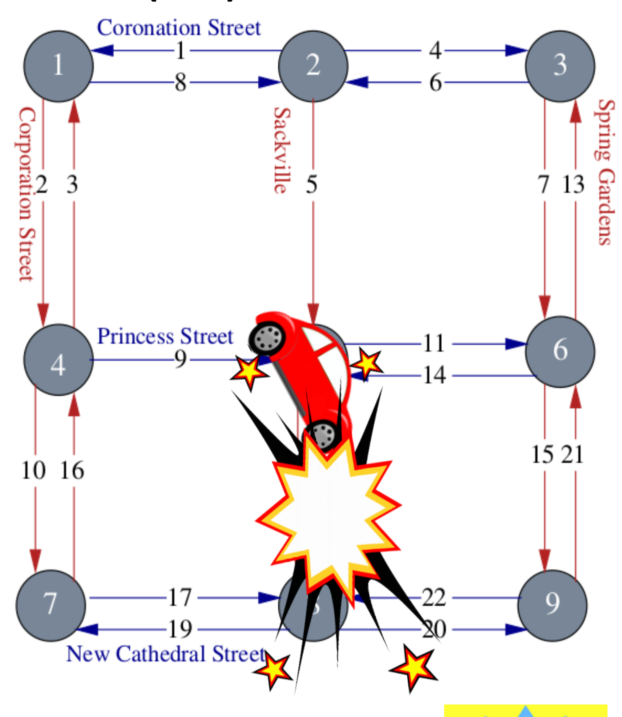
Conformance Testing (with Test Purposes)

- Check *conformance* between
  - ✓ formal **model** (M) and **test purpose** (TP)
  - ✓ **system under test** (SUT)
- Test purpose (P): functionality to be tested
- Test case (TC): controls the SUT
- Verdicts:
  - ✓ *fail*: SUT not conform to M
  - ✓ *pass*: no error
  - ✓ *inconclusive*: no error, but TP not reached



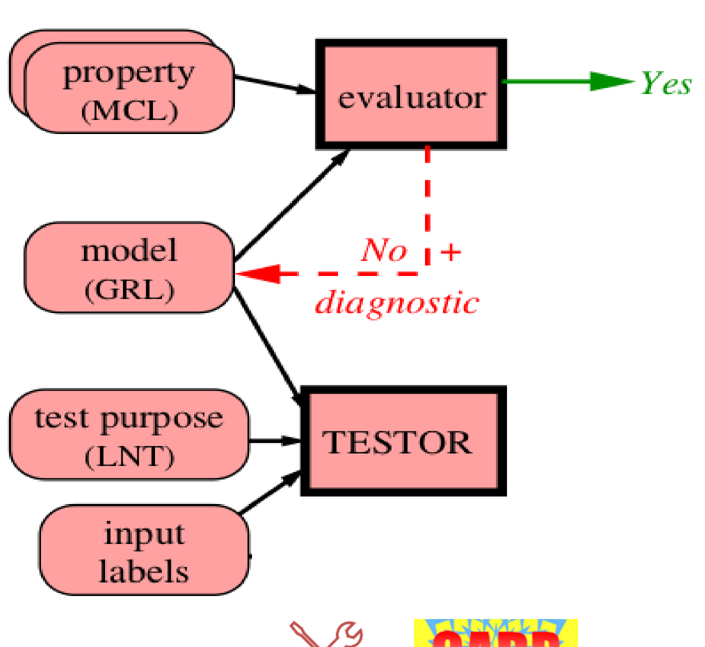
Extraction of Asynchronous Test Cases

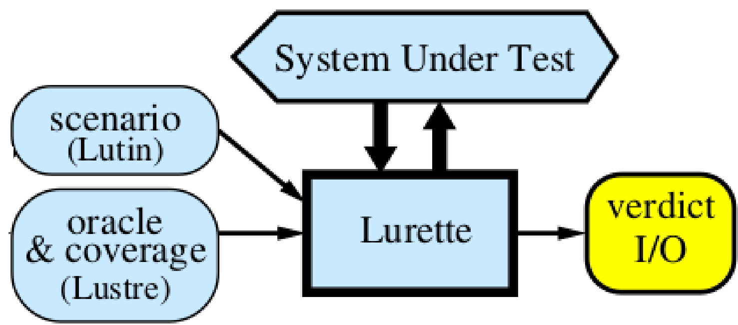
- Examples of test purposes
  - ✓ the car reached the destination (T1, T3, T4, and T5)
  - ✓ the car crashed in a collision with an obstacle (T2)



Test	TP		CTG	
	states	transitions	states	transitions
T1	5	4	15,464	29,663
T2	4	3	10,2983	211,453
T3	5	4	15,442	29,955
T4	5	4	2,276	4,957
T5	5	5	21,928	42,786

Synchronous Testing Technique





Synchronous Testing of a Component

test scenario

environment constraints

oracle & coverage criteria

SUT

system under test

Lurette

verdict

inputs / outputs

- **Lutin** specification dynamically constraints the inputs
- **Lustre** oracle implements:
  - ✓ the test decisions
  - ✓ the coverage criteria evaluating the input sequences generated

https://www-verimag.imag.fr

Summary of the Manual Approach

- Automated asynchronous testing ✓
- Handcrafting a scenario automaton
  - ✓ limit the possible behaviors (input & output values)
  - ✓ example: the car or the pedestrian can not teleport
- Translating test scenarios (input constraints, oracles)
  - ✓ Boolean and numerical types (Lutin)
  - ✓ encoding the geographical map

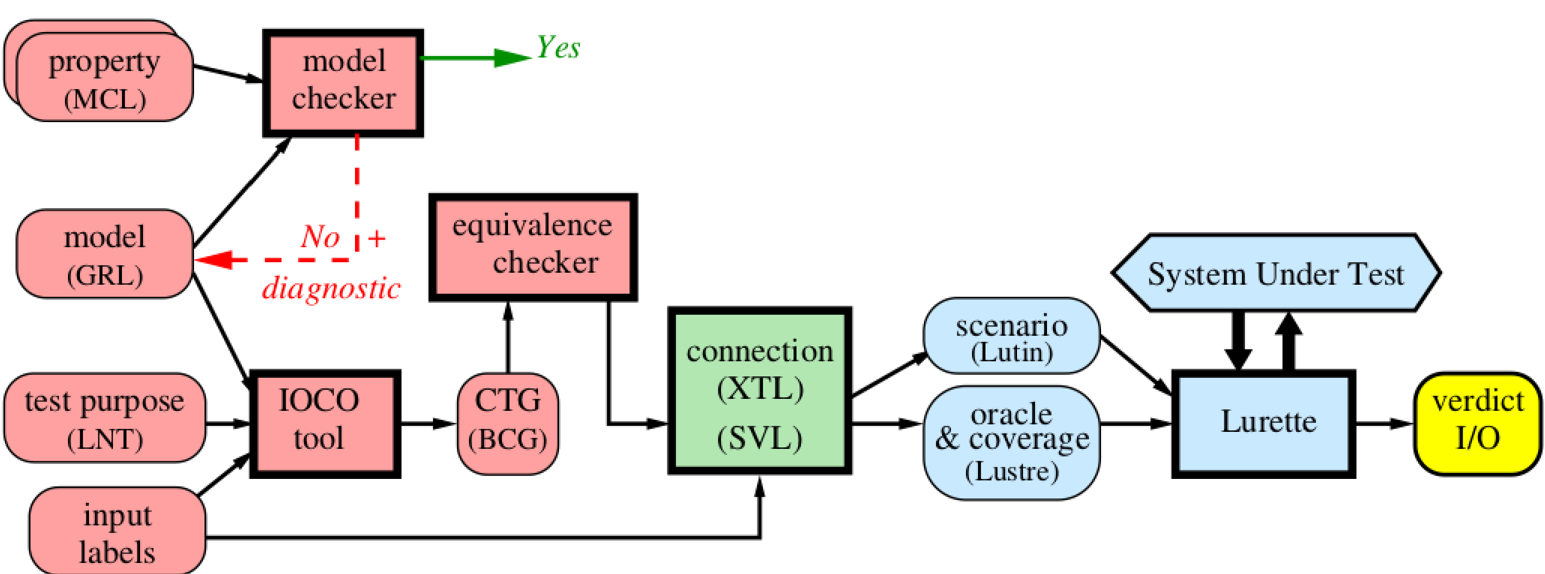
0
1
2
3
4

Coronation Street
Sackville
Spring Gardens
Corporation Street
Princess Street
New Cathedral Street
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

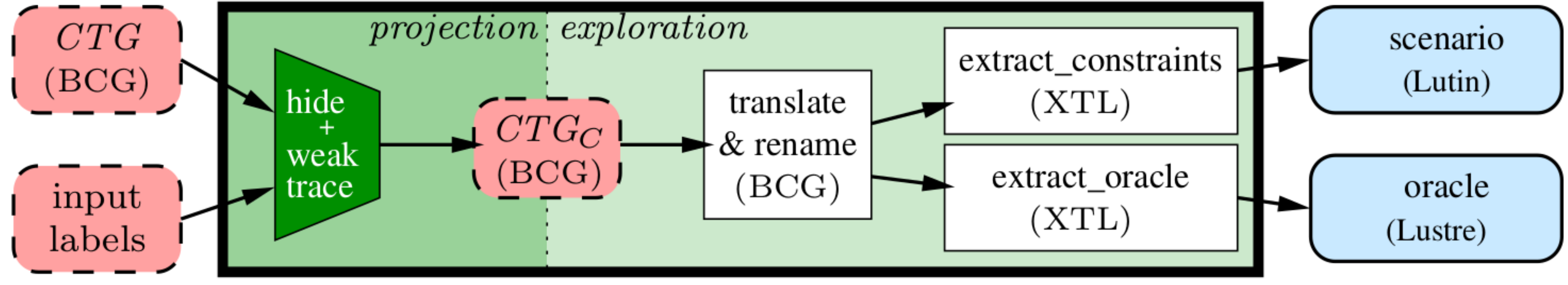
```
node input_constraints () returns (car, leo, lilly, s: int) =
  let not_visible: int = 2000 in
  (* initial state: car on street 9 and no visible obstacles *)
  car = 9 and lilly = not_visible and leo = not_visible and s = 0 fby
  loop {
    | (* s = 0 -> car on street 9, lilly on street 5, leo on street 14, s = 1 *)
      (pre s = 0) and car = 9 and [illegible] 5 and leo = 14 and s = 1
    | (* s = 0 -> car on street 12, [illegible]ly on street [illegible]leo on street 14, s = 2 *)
      (pre s = 0) and car = 1[illegible] and lilly = 5 [illegible]nd leo = 14 and s = 2
    | (* s = 1 -> car on street [illegible] leo o[illegible] street [illegible], lilly on street 12, s = 3 *)
      (pre s = 1) and car = [illegible] and [illegible] 12 a[illegible]d leo = 11 and s = 3
    | (* s = 1 -> car on street 1[illegible], le[illegible] on street [illegible], lil[illegible] on street 5, s = 4 *)
      (pre s = 1) and car = 1[illegible] and lilly = 5 [illegible]nd leo = 11 and s = 4
    | (* s = 2 -> car on street 12, [illegible] on street 1[illegible] lilly on street 5, s = 4 *)
      (pre s = 2) and car = 12 and [illegible] = 5 and leo = 11 and s = 4
    | (* s = 3 -> car on street 12, leo on street 14, lilly on street 5, s = 1 *)
      (pre s = 3) and car = 9 and lilly = 5 and leo = 14 and s = 1
    | (* s = 4 -> car on street 12, leo on street 14, lilly on street 5, s = 2 *)
      (pre s = 4) and car = 12 and lilly = 5 and leo = 14 and s = 2
  }
```

```
const invisible = 2000;
const already_sent = 3000;

node oracle (s, car, lilly, leo, perception_leo, perception_lilly: int)
returns (res, pass, blocked: bool);
let res = true ->
  ((* lilly and leo are visible from the street 9 *)
    (s = 0 and car = 9 and lilly [illegible]ble and leo = invisible and
     perception_lilly = invisib[illegible] and perce[illegible]tion_leo = invisible)
   or
   (* the perception did not chang[illegible] it [illegible]lrea[illegible] sent [illegible])
    (s = 1 and car = 9 and l[illegible]lly = [illegible]nd leo [illegible] 14 and
     perception_lilly = alrea[illegible]y_se[illegible] and [illegible]perc[illegible]ption_leo = already_sent)
   or
   (* leo and lilly are visible from th[illegible] street 9 *)
    (s = 3 and car = 9 and lilly [illegible] 12 an[illegible] leo = 11 and
     perception_lilly = 12 and perce[illegible]on_leo = 11)
   or ... );
   (* true if the car reached the destination (state 2) *)
   pass = false -> if s = 2 then true else pre pass;
   (* true, if the car is blocked by the obstacles *)
   blocked = false -> if s = 3 then true else pre pass;
tel
```

Derivation of Test Scenarios



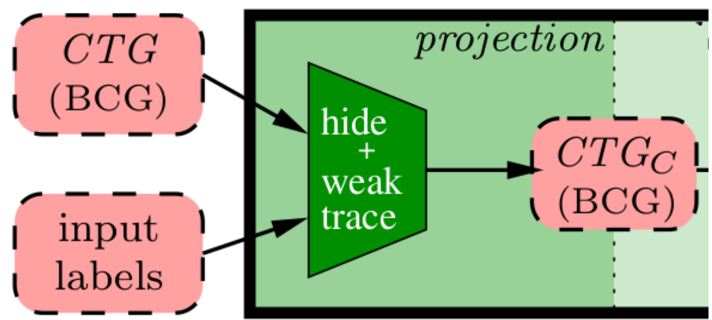
Automatic Derivation of Scenarios



Exploit the global GALS validation to improve the unit test of a synchronous component

CTG Projection ($CTG_{(C)}$)

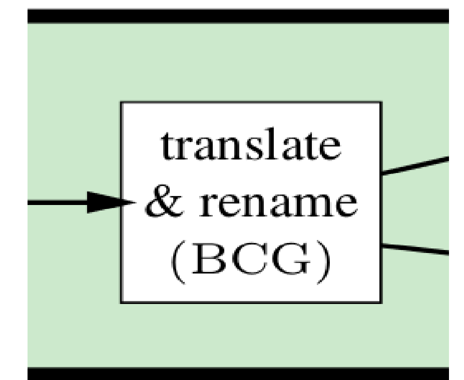
- Hide all transitions except inputs or outputs of C
- Reduce by weak trace equivalence
  - ✓ removes all internal transitions
  - ✓ determinizes $CTG_{(C)}$
- Example: CTG (radar)

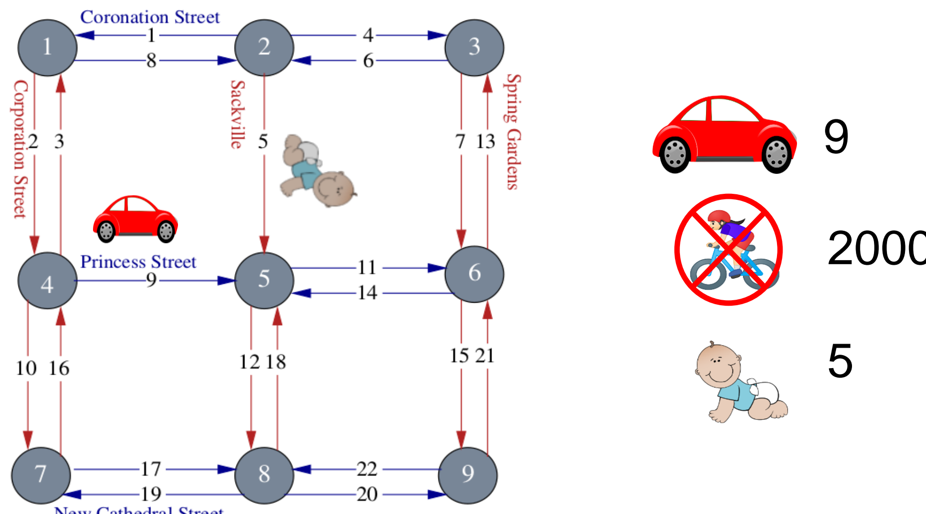


Test	CTG		$CTG_{(C)}$	
	states	trans.	states	trans.
T1	15,464	29,663	87	279
T2	102,983	211,453	582	3,615
T3	15,442	29,955	81	256
T4	2,276	4,957	216	1,117
T5	21,928	42,786	103	354

Translating & Renaming $CTG_{(C)}$

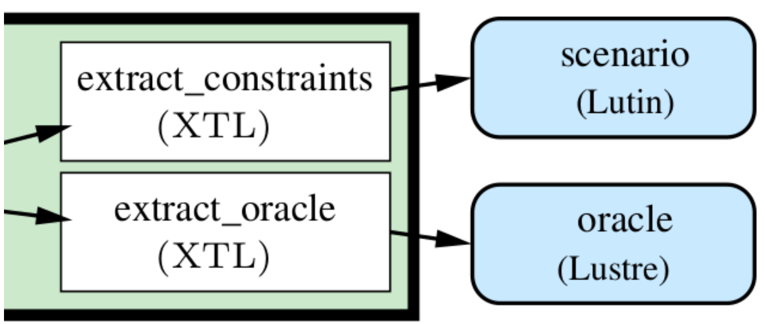
- Values on transition labels
- Boolean and numerical types (Lutin)
- Non scalar data structures





- Generic format: INPUT/OUTPUT ($!s_1$ $!v_1$ … $!s_n$ $!v_n$)

CTG Exploration



- Input constraints (extract_constraints, 77 lines of XTL):
  - ✓ nondeterministic node in Lutin
  - ✓ same inputs C + variable with the current state of $\boldsymbol{CTG}_{(\boldsymbol{C})}$
  - ✓ one item by transition (nondeterministic choice list)
- Oracle (extract_oracle, 208 lines of XTL)
  - ✓ (corner state + inputs/outputs) ⟶ boolean verdict
  - ✓ coverage variables: verdict states an states coverage

Summary of the Automatic Approach

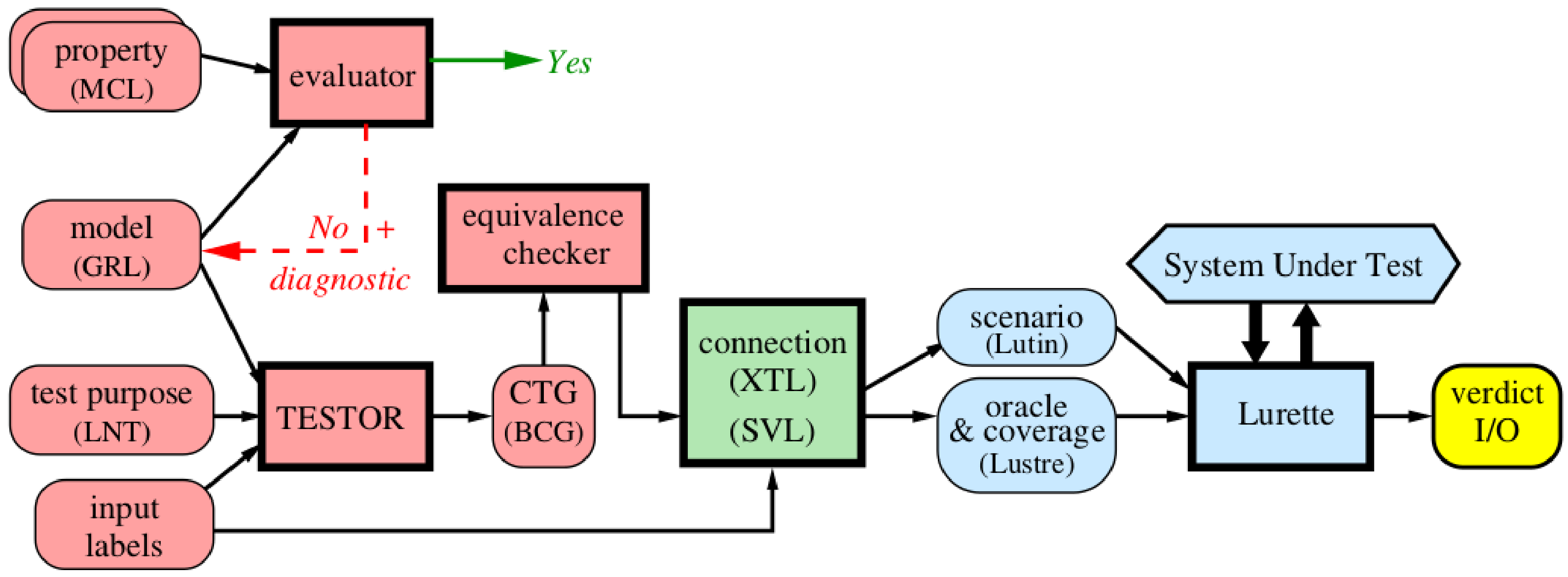
- Automated asynchronous testing ✓
- ~~Handcrafting a scenario automaton~~
- ~~Translating test scenarios (input constraints, oracles)~~
- Automated synchronous testing (relevant test scenarios) ✓

Test	TP		Scenario	Oracle	Time	Mem.
	states	trans.	(Lutin)	(Lustre)	(sec.)	(MB)
T1	5	4	283	291	28.75	199,576
T2	4	3	1911	3619	1237.15	230,520
T3	5	4	260	277	25.63	199,664
T4	5	4	1121	551	85.84	192,688
T5	5	5	358	340	35.98	2007,740

Reuse and integrate existing validation tools (synchronous and asynchronous) to validate GALS systems.

Conclusion

- Summary
  - ✓ automatic approach integrating asynchronous and synchronous testing tools

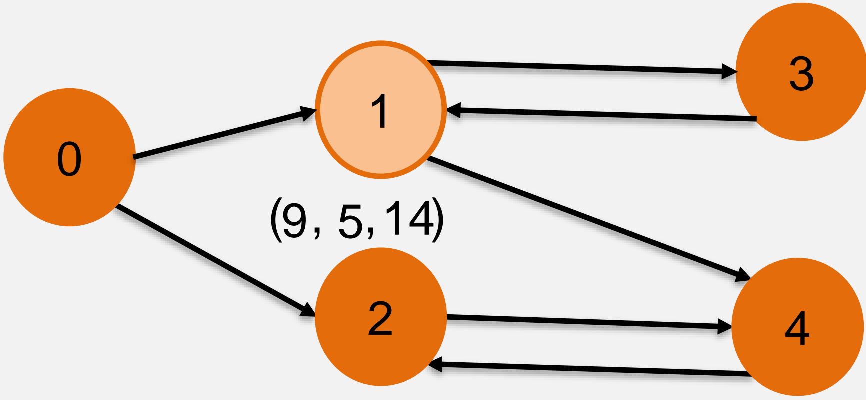


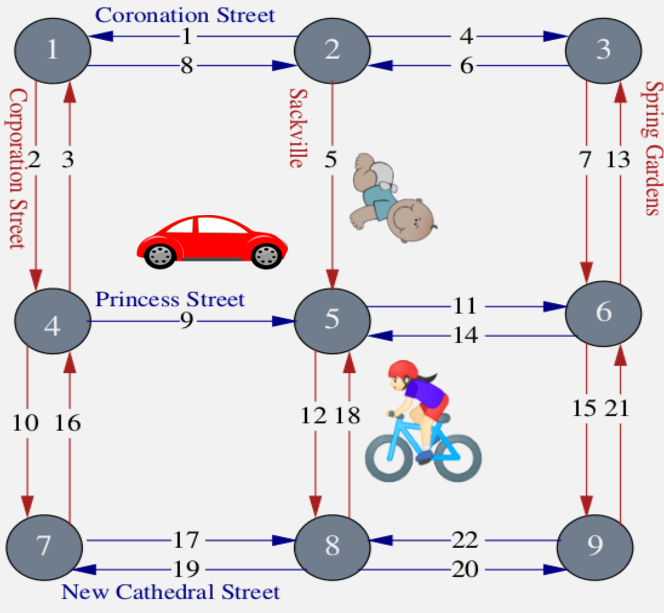
- Future work
  - ✓ behavioral coverage of GALS systems
  - ✓ enrich the model with additional information

Photo by [Rachel C](#) on [Unsplash](#)

Lutin Input Constraints

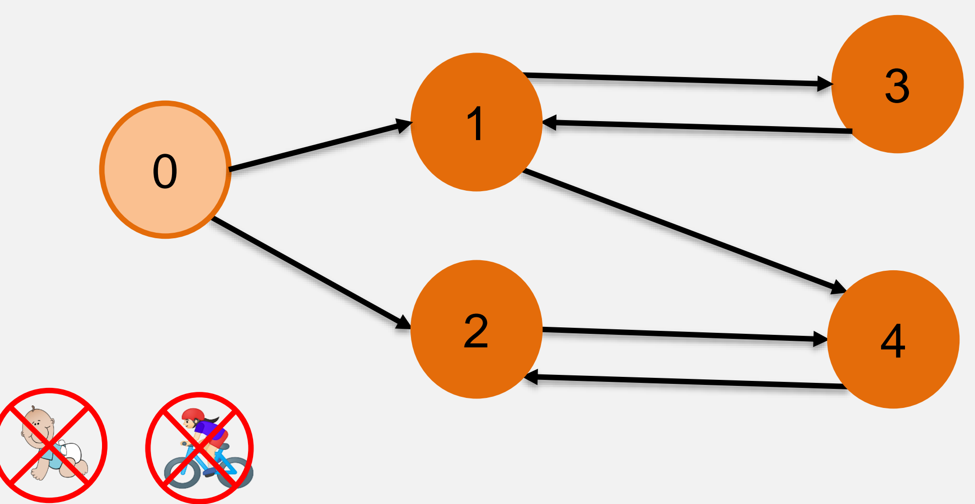
```
node input_constraints () returns (car, leo, lilly, s: int) =
  let not_visible: int = 2000 in
  (* initial state: car on street 9 and no visible obstacles *)
  car = 9 and lilly = not_visible and leo = not_visible and s = 0 fby
  loop {
    | (* s = 0 -> car on street 9, lilly on street 5, leo on street 14, s = 1 *)
      (pre s = 0) and car = 9 and lilly = 5 and leo = 14 and s = 1
```

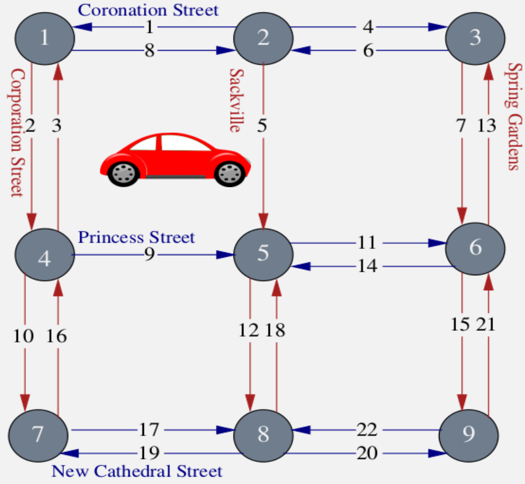




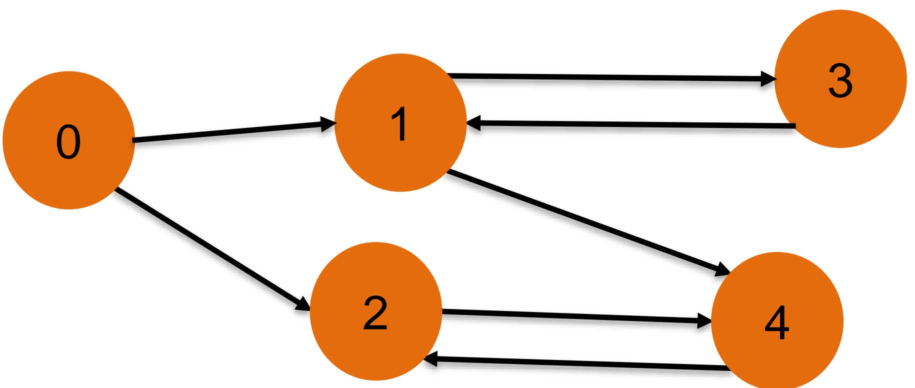
Test Decisions: Oracle

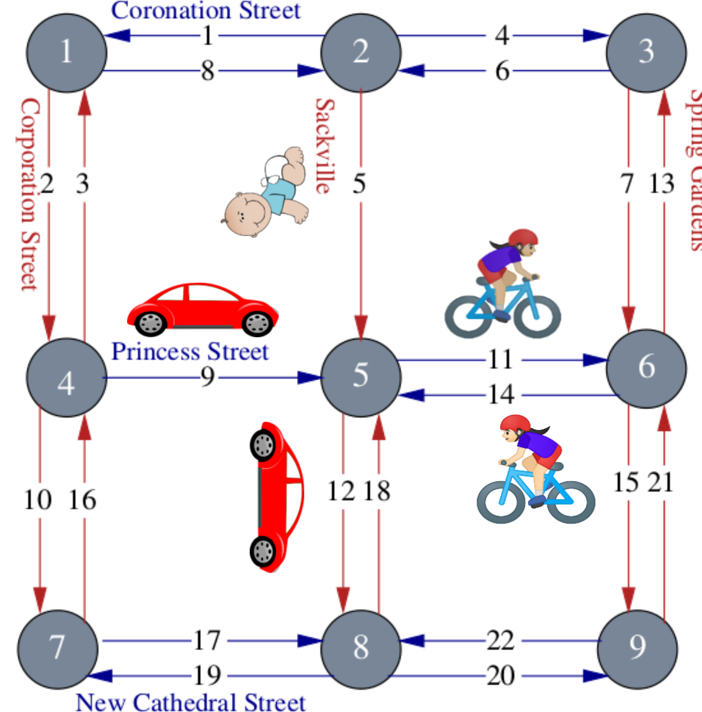
```
const invisible = 2000;
const already_sent = 3000;

node oracle (s, car, lilly, leo, perception_leo, perception_lilly: int)
returns (res, pass, blocked: bool);
let res = true ->
  ((* lilly and leo are visible from the street 9 *)
   (s = 0 and car = 9 and lilly = invisible and leo = invisible and
    perception_lilly = invisible and perception_leo = invisible)
```





Environment Constraints: Scenario





- Limit the possible behaviors (input & output values)
  - ✓ the car or the pedestrian can not teleport
  - ✓ the roads are unidirectional
- Translating test scenarios (input contraints, oracles)
  - ✓ Boolean and numerical types (Lutin)
  - ✓ encoding the geographical map