A Short Talk
on
A CCS and MCRL2 Case-Study: A Safety Critical System

Ram Chandra Bhushan

Ph.D Semester-III

Supervisor - Dr. D. K. Yadav

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad

Allahabad, India

July 18, 2017

Outline

- Introduction to formal verification and model checking
- A level crossing control system
- Architecture of the system
- Introduction to Calculus of Communication System (CCS)
- CCS Specifications for several processes
- CCS verification with concurrency workbench (CWB-NC) tool
- A glimpse of MCRL2
- MCRL2 code for the model
- Verification of safety requirements using Mu-Calculus
- Papers Explored
- References

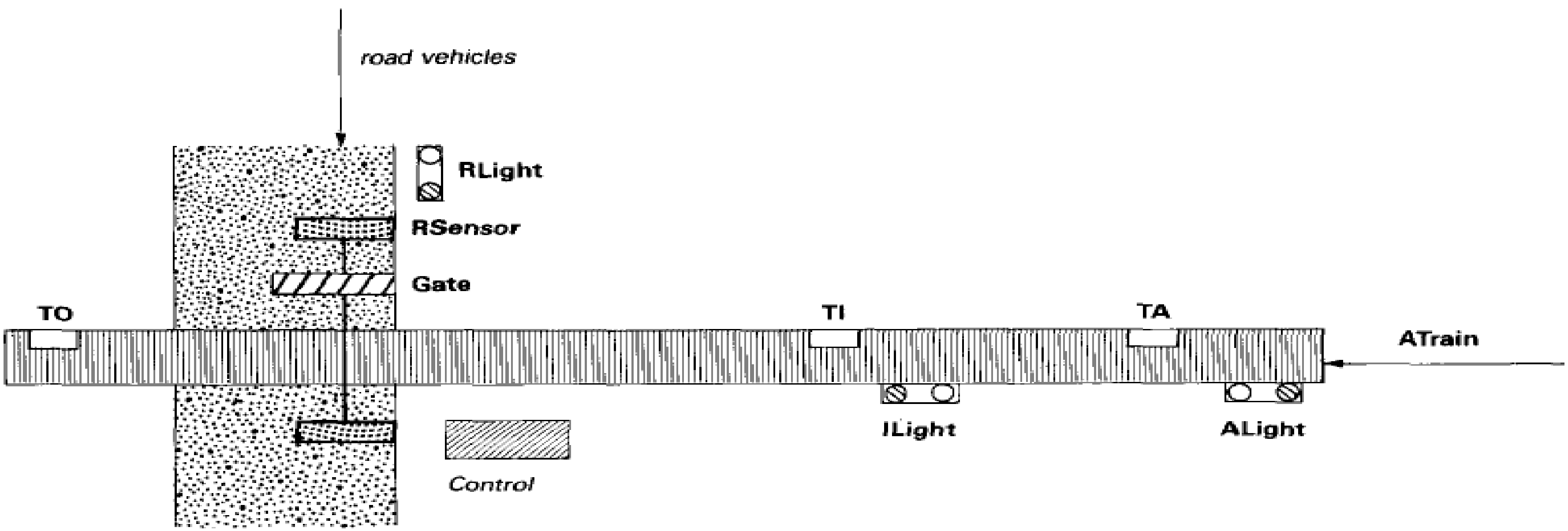
Introduction to formal verification and model checking

- Formal verification
  - Checks whether a design satisfies the requirements (properties) defined for the model?
  - Is a way to verify programs by mathematical proving that the program's post condition will hold as long as the precondition holds
- Model checking
  - Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
  - Properties are written in propositional temporal logic
  - Systems are modeled by finite state machines
  - Verification procedure is an exhaustive search of the state space of the design
  - Model checking complements testing/simulation

A level crossing control system

- TI and TO are the two sensors on the approach side of the line.

- Train driver should stop the train at a red light and proceed when the light is green



A level crossing control system contd..

- Car driver should stop their vehicles when the light is red and do not

  proceed until the light changes to green.

- Cars making legitimate use of the crossing are sensed and are counted in and out

Global model

The top-level safety requirement or ‘global model’ is that there should never be a train and a car inside the crossing at the same time.

A level crossing control system contd..

Global model can be achieved by the following lower level constraints

- ✓For every train, if it is outside the crossing and the approach light is red, train remains outside unless approach light turns back to green.

- ✓For every train, if it is before TI and in light is red, train does not crosses unless in light turns back to green.

- ✓Once the road light has been switched to red, cars in the crossing will be allowed to leave before the barrier is lowered.

A level crossing control system contd..

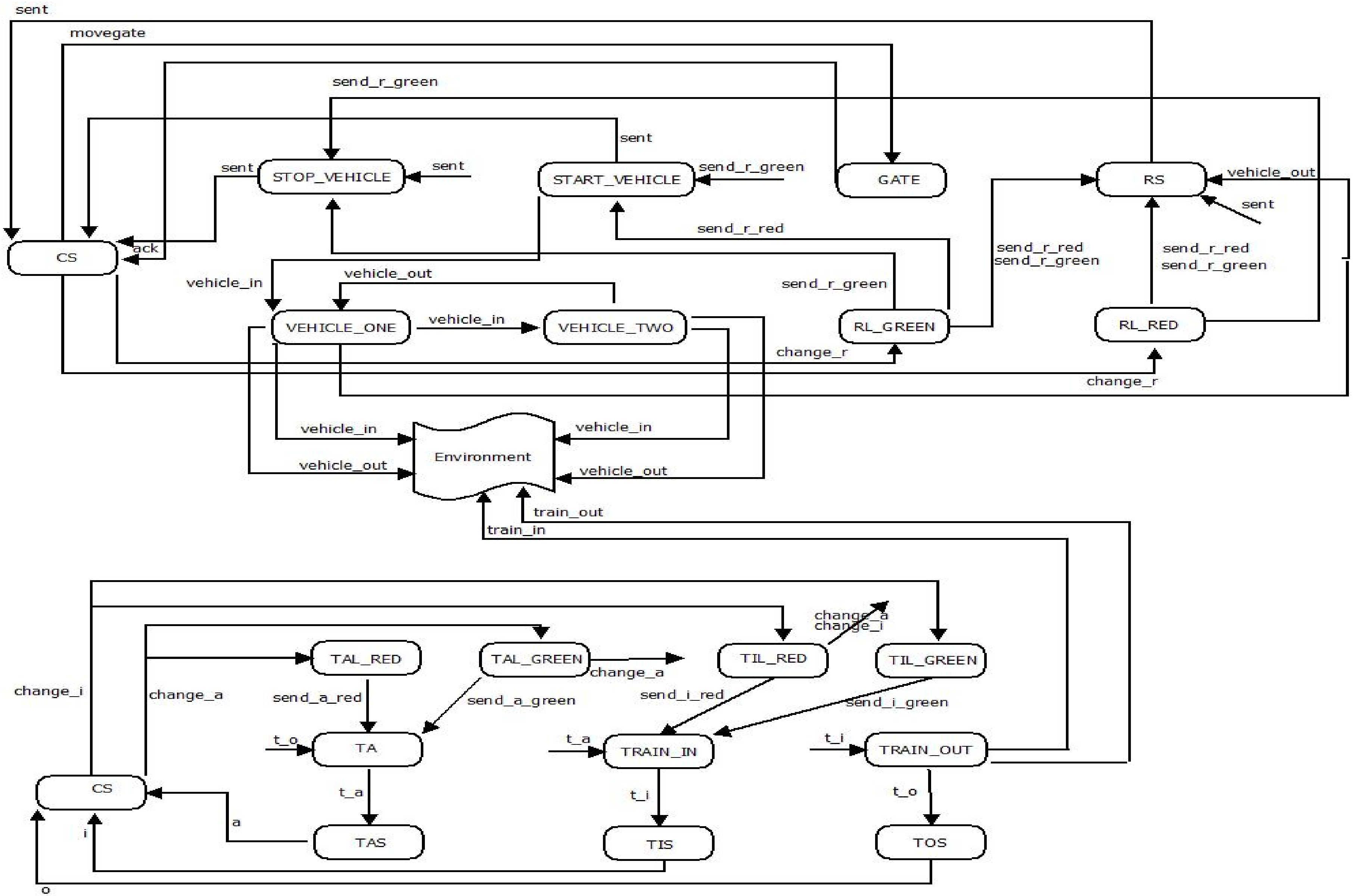
✓Once the road light has been switched to red, cars outside the crossing will not be allowed to enter into crossing.

✓Once the road light has been switched to green, cars outside the crossing will be allowed to enter into crossing.

✓If the crossing is open for cars, the road light must be green and the approach light must be red and there must be no train in the crossing.

✓If the gates are closed, the road light must be red.

A level crossing control system: Architecture



Introduction to CCS

- The calculus of communication systems (CCS) is a process calculus introduced by Robin Milner around 1980

- Is an algebra for specifying and reasoning about concurrent systems

- Provides a set of terms, operators and axioms that can be used to write and manipulate algebraic expressions

- The Concurrency WorkBench (CWB) is a public domain, interactive tool based on CCS which is used to analyze CCS specifications.

CCS Specifications for several processes

proc TAS = t_a . 'a . TAS
proc TIS = t_i . 'i . TIS
proc TOS = t_o . 'o . TOS

proc TAL_RED = 'send_a_red . change_a . TAL_GREEN
proc TAL_GREEN = 'send_a_green . change_a . TAL_RED
proc TIL_RED = 'send_i_red . change_i . TIL_GREEN
proc TIL_GREEN = 'send_i_green . change_i . TIL_RED

proc TA = send_a_red . TA + send_a_green . 't_a . TRAIN_IN

proc TRAIN_IN = send_i_red . TRAIN_IN + send_i_green . 't_i . TRAIN_OUT

proc TRAIN_OUT = 'train_in . 'train_out . 't_o . TA

proc RS = send_r_red . 'sent . STOP_VEHICLE + send_r_green . START_VEHICLE

proc STOP_VEHICLE = send_r_green . 'sent . RS
proc START_VEHICLE = 'vehicle_in . VEHICLES_ONE + send_r_red . 'sent . STOP_VEHICLE

proc VEHICLES_ONE = 'vehicle_in . VEHICLES_TWO + 'vehicle_out . RS

proc VEHICLES_TWO = 'vehicle_out . VEHICLES_ONE

proc RL_RED = 'send_r_red . RL_RED + change_r . 'send_r_green . RL_GREEN

proc RL_GREEN = 'send_r_green . RL_GREEN + change_r . 'send_r_red . RL_RED

proc GATE = movegate . 'ack . GATE

proc CS = a . 'change_a . 'change_r . sent . 'movegate . ack . 'change_i . i . 'change_i . o . 'movegate . ack . 'change_r . Sent . 'change_a . CS

proc CROSSING = TA | TAS | TIS | TOS | TAL_GREEN | TIL_RED | CS | TIL_GREEN | GATE | RS \ {a, i, o, t_a, t_i, t_o,
change_a, change_i, change_r, send_a_green, send_a_red, send_i_green, send_i_red, send_r_red, send_r_green, sent, movegate, ack}

CCS Verification – Formula 1

For every train, if it is outside the crossing and the approach light is red, train remains outside unless approach light turns back to green.

Which means

Once approach light is red the train stops and it will not send any further signal to the next sensor(TIS)

Verification Formula could be

prop Can_Send_ta = min Y = <t_a> tt <-> Y

prop Approach_Light_red = AG(([send_a_red] (not Can_Send_ta))

([send_a_green] (Can_Send_ta)))

CCS Verification – Formula 2

For every train, if it is before TI and in light is red, train does not crosses unless in light turns back to green.

Which means

Once approach light is green the train enters into approach section and send further signal to the next sensor(TIS)

Verification Formula could be

prop Can_Send_ti = min Y = <t_i> tt <-> Y

prop In_Light_red = AG(([send_i_red] (not Can_Send_ti)) ([send_i_green] (Can_Send_ti)))

CCS Verification - Formula 3

Once the road light has been switched to red, cars in the crossing will be allowed to leave before the barrier is lowered.

Verification Formula could be

prop Accident_prevention = (not <send_r_red>tt)
AG([vehicle_in]
EF(<vehicle_out>tt / <movegate>tt))

CCS Verification - Formula 4

Once the RLight (road light) has been switched to red, cars outside the crossing will not be allowed to enter into crossing.

Verification Formula could be

prop Car_Not_Allowed = AG([send_r_red](not Can_Vehicle_In))

CCS Verification - Formula 5

Once the RLight (road light) has been switched to green, cars outside the crossing will be allowed to enter into crossing.

Verification Formula could be

prop Car_Allowed = AG([send_r_green](Can_Vehicle_In))

CCS Verification - Formula 6

If the crossing is open for cars, the RLight must be green and the ALight must be red and there must be no train in the crossing.

Verification Formula could be

prop Crossing_Open = (not<vehicle_in>tt)
AG(<send_a_red>tt /
<send_r_green>tt)

CCS Verification - Formula 7

If the gates are closed, the RLight must be red.

Verification Formula could be

prop Gate_Close = (Crossing_Open)
AG((not<vehicle_in>tt) /
(<send_r_red>tt))

CCS Verification - Formula 8

In the model there should not be a deadlock. The below property will give false while executing unlike all other properties which gives true.

Verification Formula could be

prop Can_Deadlock = min X = [-]ff <->X

A glimpse of MCRL2

- MCRL2 is a formal specification language with an associated toolset

- The toolset can be used for modeling, validation and verification of concurrent

  systems and protocols

Equivalent MCRL2 code

```
sort light=struct Red|Green;

act  send_a,  send_a', rsend_a, send_i, send_i',  rsend_i,
send_r, send_r', rsend_r : light;

act t_a, t_a', rt_a, a, a', ra, i', ri,  o', ro, t_i, t_i', rt_i, i, t_o,
t_o', rt_o, o, change_a, change_a', rchange_a, change_i,
change_i', rchange_i, train_in, train_in', train_out',
train_out, change_r, change_r', rchange_r, movegate',
movegate, rmovegate, done, done', rdone, sent, sent', rsent,
car_in, car_in', car_out, car_out';

map change_val: light -> light;

var m:Int;
eqn change_val(Red)=Green;

change_val(Green)=Red;

proc TA=t_a.a'.TA;

proc TI=t_i.i'.TI;

proc TO=t_o.o'.TO;

proc
Alight(x:light)=send_a'(x).change_a.Alight(change_val(x))
```

```
proc
Ilight(y:light)=send_i'(y).change_i.Ilight(change_val(y));

proc Atrain=send_a(Red).Atrain +
send_a(Green).t_a'.Itrain;

proc Itrain=send_i(Red).Itrain + send_i(Green).t_i'.Ctrain;

proc Ctrain=train_in'.train_out'.t_o'.Atrain;

proc Rsensor=send_r(Red).sent'.Stop + send_r(Green).Go;

proc Stop=send_r(Green).sent'.Rsensor;

proc Go=car_in'.Cars(1) + send_r(Red).sent.Stop;

proc Cars(m:Int)=(m > 0 && m < 3) -> (car_in'.Cars(m+1)
+ car_out'.(m==1)->Rsensor <> Cars(m-1)) <> delta;

proc Gate=movegate.done'.Gate;

proc
Control=a.change_a'.change_r'.sent.movegate'.done.change
_i'.i.change_i'.

 proc Rlight(z:light)=send_r'(z).Rlight(z) +
change_r.send_r'(change_val(z)).Rlight(change_val(z));
```

Equivalent MCRL2 code contd..

```
proc Crossing = Atrain || TA || TI || TO || Alight(Green) ||

Ilight(Red) || Control || Rlight(Green) || Gate || Rsensor;

  init hide(


{ra, ri, ro, rt_a, rt_i, rt_o, rchange_a, rchange_i, rchange_r,

rsend_a, rsend_i, rsend_r, rsent, rmovegate, rdone },

allow( {
        train_in, train_in', train_out, train_out', car_in,

car_in', car_out, car_out', ra, ri, ro, rt_a, rt_i, rt_o,

rchange_a, rchange_i, rchange_r, rsend_a,  rsend_i,

rsend_r, rsent, rmovegate, rdone
   },
```

```
comm( {        a | a' -> ra,             i | i' -> ri,

                o | o' -> ro,            t_a | t_a' -> rt_a,

          t_i | t_i' -> rt_i,              t_o | t_o' -> rt_o,

          change_a | change_a' -> rchange_a,

          change_i | change_i' -> rchange_i,

          change_r | change_r' -> rchange_r,

        send_a | send_a' -> rsend_a,

        send_i | send_i' -> rsend_i,

        send_r | send_r' -> rsend_r,
        sent | sent' -> rsent,
        movegate | movegate' -> rmovegate,
        done | done' -> rdone
        },
  Crossing
)));
```

Verification

For every train t, if t is outside the crossing and the approach light is red, t remains outside unless it ‘sees’ the green light.

Which means

Once approach light is red the train stops and it will not send any further signal to the next sensor(Itrain)

Verification Formula could be

[true*][send_a(Red)]<!t_a‘>[send_a(Green)]<t_a‘>true

Verification contd..

For every train t, if t is outside the crossing and the in light is red, t remains outside unless it 'sees' the green light.

Which means

Once in light is red the train stops and it will not send any further signal to the next process

Verification Formula could be

[true] [send_i(Red).!t_i'] [send_i(Green).t_i']true*

Verification contd..

After the road light has been switched to red, cars in the crossing will be allowed to leave before the barrier is lowered

Which means

After the road light has been switched to Red, no cars will be allowed to enter into the crossing whereas if it has been switched to green cars will be allowed to enter as well as to exit from other end

Verification Formula could be

[true*] [send_r(Red)] <sent'> [send_r(Red)] !

<car_in‘><car_out'> [send_r(Green)] <car_in‘><car_out'> true

Deadlock and Livelock

This formula expresses that there is no deadlock for all the reachable states.

[true*]<true>true

This formula expresses that there is no livelock for all the reachable states.

[true*]mu X.[tau]X

Verification

For every train t, it follows the following sequence of actions.

- Train receives green approach light
- Train approach light sends green to TAS(Train approach sensor)
- TAS sends green to TIS(Train in sensor)
- TOS(Train out sensor) sends red to control once train goes out of the crossing

$$[true^*][send\_a(Green)]\langle t\_a' \rangle \langle send\_i\_green \rangle$$
$$\langle t\_i' \rangle \langle train\_in' \rangle \langle train\_out' \rangle \langle t\_o' \rangle true$$

Papers explored

[1] Baillie, Jean. "A CCS cast study: a safety-critical system." Software Engineering Journal 6.4 (1991): 159-167

[2] Cranen, Sjoerd, et al. "An overview of the mCRL2 toolset and its recent advances." International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2013.

[3] Groote, Jan Friso, et al. "The mCRL2 toolset." Proceedings of the International Workshop on Advanced Software Development Tools and Techniques (WASDeTT 2008). 2008.

[4] Cleaveland, Rance, Tan Li, and Steve Sims. "The Concurrency Workbench of the New Century, Version 1.2-User's Manual." (2000).

Conferences and journals explored:

ICSE: International Conference on Software Engineering

FSE :Foundations of Software Engineering

ISSTA : International Symposium on Software Testing and Analysis

ICSME: International Conference on Software Maintenance and Evolution

FMICS: Formal Methods for Industrial Critical Systems

FMSPACM SIGSOFT Workshop on Formal Methods in Software Practice

International Conference on Rewriting Techniques and Applications

IEEE Transactions on Software Engineering (TSE): main software engineering research journal

ACM Transactions on Software Engineering and Methodology (TOSEM): first issue dated January 1992Software Testing, Verification and Reliability aimed at practitioners; dissemination of new techniques, methodologies and standards

Automated Software Engineering - An International Journal

Journal of Systems and Software: meant to be more practitioner-oriented than other research journals Software Quality Journal: academic research and industrial case studies and experience

Empirical Software Engineering - An International Journal

Journal of Software Maintenance and Evolution: Research and Practice: refereed; intended for both researchers and practitioners; joint US/UK editorial board

Software: Practice and Experience: not always software engineering; good reputation for practice

International Journal on Software Tools for Technology Transfer

Transactions on Aspect-Oriented Software Development Journal

References

[1] Baillie, Jean. "A CCS case study: a safety-critical system." Software Engineering Journal 6.4 (1991): 159-167

[2] R. Milner . (1980) , A calculus of communicating systems.

[3] R. Milner . (1989) , Communication and concurrency.

[4] Walker, D.: `Introduction to a calculus of communicating systems', ECS-LFCS-87-22, Expository Report, March 1987.

[5] Gorski, J.: `Formal support for development of safety related systems', Safety and Reliability Symp., 1987, Attrincham UK, (Elsevier Applied Sciences).

[6] Cleaveland, Parrow, Steffen, : `The Concurrency Workbench: a semantics-based tool for the verification of concurrent systems', ECS-LFCS-89-83, Technical Report, August 1989, LFCS.

[7] M. Hennessy . (1988) , Algebraic theory of processes.

[8] C.A.R. Hoare . (1985) , Communicating sequential processes.

[9] M. Hennessy , R. Milner . Algebraic laws for non-determinism and concurrency. J. ACM , 1 , 137 - 161

References

[10] Groote, Jan Friso, et al. "The formal specification language mCRL2."Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[11] Cranen, Sjoerd, et al. "An overview of the mCRL2 toolset and its recent advances." International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2013.

[12] J.F. Groote, M.R. Mousavi. Modeling and analysis of communicating systems The MIT press. 2014.

Thank You