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
• 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.*
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.
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
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

\[
\begin{align*}
\text{proc } \text{TAS} &= t_a \cdot 'a \cdot \text{TAS} \\
\text{proc } \text{TIS} &= t_i \cdot 'i \cdot \text{TIS} \\
\text{proc } \text{TOS} &= t_o \cdot 'o \cdot \text{TOS} \\
\text{proc } \text{TAL\_RED} &= '\text{send\_a\_red} \cdot \text{change\_a} \cdot \text{TAL\_GREEN} \\
\text{proc } \text{TIL\_RED} &= '\text{send\_i\_red} \cdot \text{change\_i} \cdot \text{TIL\_GREEN} \\
\text{proc } \text{TAL\_GREEN} &= '\text{send\_a\_green} \cdot \text{change\_a} \cdot \text{TAL\_RED} \\
\text{proc } \text{TIL\_GREEN} &= '\text{send\_i\_green} \cdot \text{change\_i} \cdot \text{TIL\_RED} \\
\text{proc } \text{TA} &= \text{send\_a\_red} \cdot \text{TA} + \text{send\_a\_green} \cdot 't_a \cdot \text{TRAIN\_IN} \\
\text{proc } \text{TRAIN\_IN} &= \text{send\_i\_red} \cdot \text{TRAIN\_IN} + \text{send\_i\_green} \cdot 't_i \cdot \text{TRAIN\_OUT} \\
\text{proc } \text{TRAIN\_OUT} &= '\text{train\_in} \cdot '\text{train\_out} \cdot 't_o \cdot \text{TA} \\
\text{proc } \text{RS} &= \text{send\_r\_red} \cdot \text{sent} \cdot \text{STOP\_VEHICLE} + \text{send\_r\_green} \cdot \text{START\_VEHICLE} \\
\text{proc } \text{STOP\_VEHICLE} &= \text{send\_r\_green} \cdot \text{sent} \cdot \text{RS} \\
\text{proc } \text{START\_VEHICLE} &= '\text{vehicle\_in} \cdot \text{VEHICLES\_ONE} + \text{send\_r\_red} \cdot \text{sent} \cdot \text{STOP\_VEHICLE} \\
\text{proc } \text{VEHICLES\_ONE} &= '\text{vehicle\_in} \cdot \text{VEHICLES\_TWO} + '\text{vehicle\_out} \cdot \text{RS} \\
\text{proc } \text{VEHICLES\_TWO} &= '\text{vehicle\_out} \cdot \text{VEHICLES\_ONE} \\
\text{proc } \text{RL\_RED} &= '\text{send\_r\_red} \cdot \text{RL\_RED} + \text{change\_r} \cdot '\text{send\_r\_green} \cdot \text{RL\_GREEN} \\
\text{proc } \text{RL\_GREEN} &= '\text{send\_r\_green} \cdot \text{RL\_GREEN} + \text{change\_r} \cdot '\text{send\_r\_red} \cdot \text{RL\_RED} \\
\text{proc } \text{GATE} &= \text{movegate} \cdot '\text{ack} \cdot \text{GATE} \\
\text{proc } \text{CS} &= a \cdot '\text{change\_a} \cdot '\text{change\_r} \cdot \text{sent} \cdot \text{movegate} \cdot '\text{ack} \cdot '\text{change\_i} \cdot '\text{change\_i} \cdot '\text{movegate} \cdot \text{ack} \cdot '\text{change\_r} \cdot \text{Sent} \cdot '\text{change\_a} \cdot \text{CS} \\
\text{proc } \text{CROSSING} &= \text{TA} | \text{TAS} | \text{TIS} | \text{TOS} | \text{TAL\_GREEN} | \text{TIL\_RED} | \text{CS} | \text{TIL\_GREEN} | \text{GATE} | \text{RS} \setminus \{a, i, o, t_a, t_i, t_o, '\text{change\_a}, '\text{change\_i}, '\text{change\_r}, '\text{send\_a\_green}, '\text{send\_a\_red}, '\text{send\_i\_green}, '\text{send\_i\_red}, '\text{send\_r\_red}, '\text{send\_r\_green}, \text{sent}, \text{movegate}, \text{ack}\} \]
\]
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

\[
\text{prop } \text{Can\_Send\_ta} = \min Y = \langle t_a \rangle tt \leftrightarrow Y
\]

\[
\text{prop } \text{Approach\_Light\_red} = AG((\text{send\_a\_red}) \ (\text{not} \ \text{Can\_Send\_ta}))\\
((\text{send\_a\_green}) \ (\text{Can\_Send\_ta}))
\]
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

\[ \text{prop Can\_Send\_ti} = \min Y = <t_i> tt \leftrightarrow Y \]

\[ \text{prop In\_Light\_red} = AG((\text{send\_i\_red} \ (\text{not Can\_Send\_ti})) \ (\text{send\_i\_green} \ (\text{Can\_Send\_ti}))) \]
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

\[
\text{prop Accident\_prevention} = (\neg <send\_r\_red> tt) \\
AG([vehicle\_in] \\
EF(<vehicle\_out> tt / <movegate> tt))
\]
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

\[ \text{prop Car\_Not\_Allowed} = AG([send\_r\_red](\neg \text{Can\_Vehicle\_In})) \]
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

\[
\text{prop Car\_Allowed} = AG([send\_r\_green](\text{Can\_Vehicle\_In}))
\]
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

\[
\text{prop Crossing\_Open} = (\text{not}<\text{vehicle\_in}>tt) \\
\text{AG}(<\text{send\_a\_red}>tt / <\text{send\_r\_green}>tt)
\]
If the gates are closed, the RLight must be red.

Verification Formula could be

\[
prop \text{ Gate\_Close} = (\text{Crossing\_Open}) \\
AG((\text{not}<vehicle\_in>tt) / \\
\quad (<send\_r\_red>tt))
\]
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

$$\text{prop Can\_Deadlock} = \min X = [-]ff \leftrightarrow X$$
• 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
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 Rlight(z:light)=send_r'(z).Rlight(z) + change_r.send_r'(change_val(z)).Rlight(change_val(z));

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 Control(Change_a'.Change_r'.Sent.Movegate'.Done.Change_i'.i.Change_i');
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,
change_a, change_i, change_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

)));
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

\[ [true*][send_a(\text{Red})]<!t_a'>[send_a(\text{Green})]<t_a'\textgreater \text{true} \]
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

$$[true^*] [send_i(\text{Red}).!t_i'] [send_i(\text{Green}).t_i']true$$
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

\[ [true^*] [send_r(\text{Red})] <sent'> [send_r(\text{Red})] ! <car_in'?><car_out'> [send_r(\text{Green})] <car_in'?><car_out'> true \]
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

$[true^*][send_a(Green)] <t_a^'>> send_i_green <t_i^'>> train_in^'>> train_out^'>> t_o^'>> true$
Papers explored


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 1992

Software 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


References


Thank You