Discrete Control of Response for Cybersecurity in Industrial Control

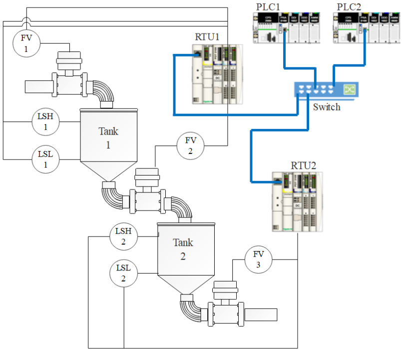
Gwenaël Delaval, Ayan Hore, Stéphane Mocanu,
Lucie Muller, Eric Rutten

Lig, Université Grenoble Alpes, Inria, Grenoble INP

Synchron 2021

Cybersecurity in Industrial Control Systems

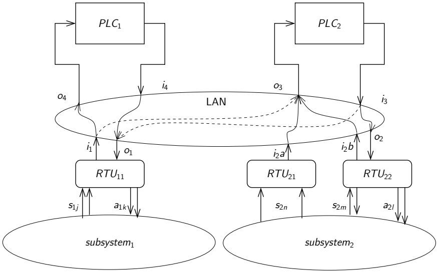
- Industrial Control Systems (ICS): critical infrastructure
- Need for cybersecurity
- Control of a response mechanism to potential attacks
- Proposal: use of controller synthesis to produce automatically a controller for this response mechanism



Controlled ICS

Industrial control system:

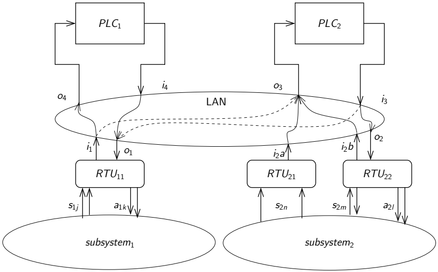
- composed of *Remote Terminal Units* (RTU), connected with sensors and actuators of the physical process
- Programmable Logic Controllers (PLC)
- PLCs and RTUs are connected by a LAN
- PLCs run *programs* controlling the RTUs (possibly several programs by PLC)



Controlled ICS

Industrial control system:

- composed of *Remote Terminal Units* (RTU), connected with sensors and actuators of the physical process
- Programmable Logic Controllers (PLC)
- PLCs and RTUs are connected by a LAN
- PLCs run *programs* controlling the RTUs (possibly several programs by PLC)



Attacks on PLCs $\longrightarrow$ need for dynamic reconfigurations

Responses to attacks

What kind of response to attacks/alarms?

- Type of attacks considered: alarms on PLCs, triggered by an Intrusion Detection System (IDS)
- Dynamic reconfigurations:
  - isolation of nodes on the LAN
  - execution location of programs on PLCs
  - execution modes: Nominal, Degraded, Safe
- Execution modes $\Rightarrow$ different execution times

Responses to attacks

What kind of response to attacks/alarms?

- Type of attacks considered: alarms on PLCs, triggered by an Intrusion Detection System (IDS)
- Dynamic reconfigurations:
  - isolation of nodes on the LAN
  - execution location of programs on PLCs
  - execution modes: Nominal, Degraded, Safe
- Execution modes $\Rightarrow$ different execution times

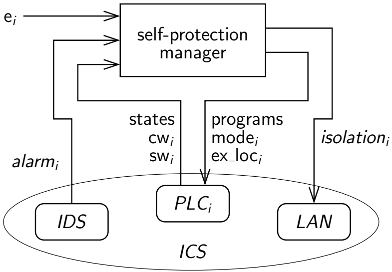
Objectives

- execution of programs on non-alarmed PLCs
- keep programs in Nominal or Degraded modes as long as possible
- bound execution time on each PLC

Cybersecurity as a Control Problem

Closing the loop:

- inputs: alarms from the IDS
- outputs: isolation of nodes of the LAN, modes and execution location of programs
- state: current execution modes/location of programs

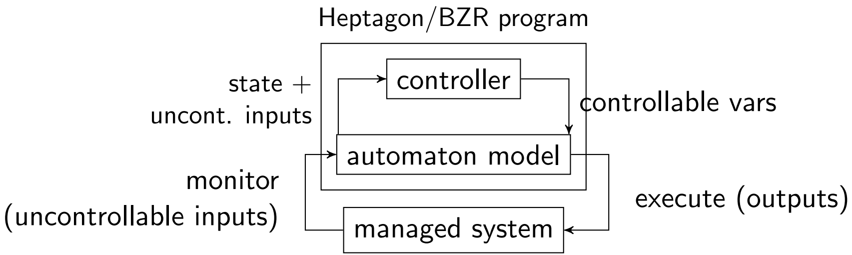


Combinatorics of solutions ⇒ controller difficult to program "manually"

Heptagon/BZR

Automation of controller generation: use of Heptagon/BZR

- Managed system modelled as automata and (synchronous) dataflow equations
- Controllable variables defined at runtime by a synthesized controller, to enforce *synthesis objectives*: invariant temporal properties
- Controller synthesized offline



Classical design cycle

Programmer

Model-checker

Does my program satisfy the property P ?

No.

Does my (modified) program satisfy the property P ?

No.

Does my (modified) program satisfy the property P ?

No.

Heptagon/BZR design cycle

Programmer

BZR chain tool

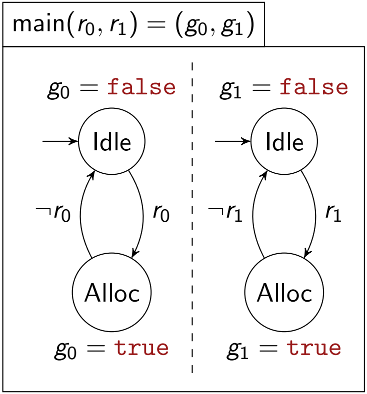
My program is nondeterministic.
Can you please constrain it so that it satisfies the property P ?

Sure: here is the constraint.

Synchronous programming in Heptagon/BZR

```
node alloc(r : bool) returns (g : bool)
let
  automaton
    state Idle
      do g = false until r then Alloc
    state Alloc
      do g = true until (not r) then Idle
  end
tel

node main(r0,r1 : bool) returns (g0,g1 : bool)
let
  g0 = inlined alloc(r0);
  g1 = inlined alloc(r1);
tel
```



t	1	2	3	4	5	6	7	8	...
r_0	0	1	1	1	1	0	0	0	...
r_1	0	0	0	1	1	1	1	1	...
g_0	0	0	1	1	1	1	0	0	...
g_1	0	0	0	0	1	1	1	1	...

```
node main(r0,r1 : bool) returns (g0,g1 : bool)



let

  g0 = inlined alloc(r0);
  g1 = inlined alloc(r1);
tel
```

t	1	2	3	4	5	6	7	8	...
r_0	0	1	1	1	1	0	0	0	...
r_1	0	0	0	1	1	1	1	1	...
g_0	0	0	1	1	1	1	0	0	...
g_1	0	0	0	0	1	1	1	1	...

```
node main(r0,r1 : bool) returns (g0,g1 : bool)
  contract
  assume true
  enforce not (g0 & g1)
let

  g0 = inlined alloc(r0);
  g1 = inlined alloc(r1);
tel
```

t	1	2	3	4	5	6	7	8	...
r_0	0	1	1	1	1	0	0	0	...
r_1	0	0	0	1	1	1	1	1	...
g_0	0	0	1	1	1	1	0	0	...
g_1	0	0	0	0	1	1	1	1	...

- contract mechanism

```
node main(r0,r1 : bool) returns (g0,g1 : bool)
  contract
  assume true
  enforce not (g0 & g1) with (c0,c1:bool)
let

  g0 = inlined alloc(r0 & c0);
  g1 = inlined alloc(r1 & c1);
tel
```

t	1	2	3	4	5	6	7	8	...
r_0	0	1	1	1	1	0	0	0	...
r_1	0	0	0	1	1	1	1	1	...
g_0	0	0	1	1	1	1	0	0	...
g_1	0	0	0	0	0	0	1	1	...

- contract mechanism
- nondeterminism: controllable variables

```
node main(r0,r1 : bool) returns (g0,g1 : bool)


  var c0,c1:bool
let
  (c0,c1) = controller(r0,r1);
  g0 = inlined alloc(r0 & c0);
  g1 = inlined alloc(r1 & c1);
tel
```

t	1	2	3	4	5	6	7	8	...
r_0	0	1	1	1	1	0	0	0	...
r_1	0	0	0	1	1	1	1	1	...
g_0	0	0	1	1	1	1	0	0	...
g_1	0	0	0	0	0	0	1	1	...

- contract mechanism
- nondeterminism: controllable variables
- constraint: controller computed by discrete controller synthesis

Method for obtention of response mechanism controller

Using Heptagon/BZR:

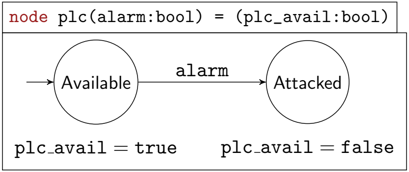
- model PLCs and programs as automata + dataflow equations
- express response objectives as *synthesis objectives*
- compile and synthesize the controller

Modelling ICS

Problem stated as:

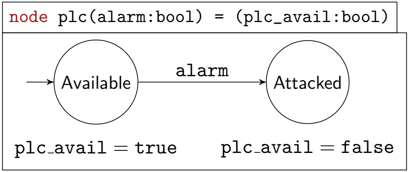
- a set of n control programs $P_i, i = 1, \ldots, n$;
- a set of p PLCs $C_j, j = 1, \ldots, p$;
- $\max_j$ is the maximum cycle duration of PLC C_j;
- n_{ij} is the duration of the *nominal* version of program P_i on PLC C_j;
- d_{ij} is the duration of the *degraded* version of program P_i on PLC C_j.

PLC model



- Input: `alarm`, true when the IDS detects an alarm for this PLC
- Output: `plc_avail`, true when the PLC is “available” (until first alarm)

PLC model



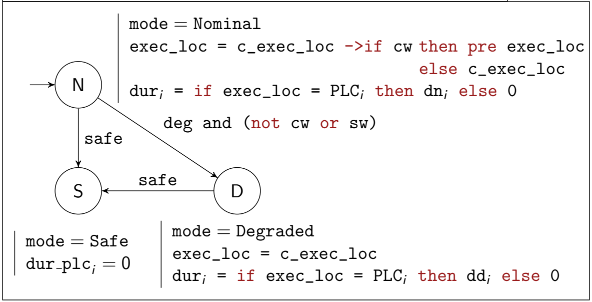
- Input: `alarm`, true when the IDS detects an alarm for this PLC
- Output: `plc_avail`, true when the PLC is "available" (until first alarm)

Parallel instances for each PLC:

$$\texttt{plc\_avail}_1 \;\texttt{=}\; \texttt{plc(alarm}_1\texttt{);}$$
$$\vdots$$
$$\texttt{plc\_avail}_p \;\texttt{=}\; \texttt{plc(alarm}_p\texttt{);}$$

Program model

```
node prog<<dn1,...,dnp,dd1,...,ddp>>
        (c_exec_loc:arch_element;
         deg,safe,cw,sw:bool)
      = (mode:prog_mode;exec_loc:arch_element;
         dur1,..., durp:int)
```



- states corresponding to program modes: Nominal (N), Degraded (D), Safe (S)
- input `c_exec_loc`: controllable variable, control the location of the program

Program model — instances

Node `prog` instantiated for each program:

```
(mode_1,ex_loc_1,dur_11,...,dur_1p) =
        prog<<n_11,...,n_1p,d_11,...,d_1p>> (el_1,cd_1,es_1 or cs_1,cw_1,sw_1);
⋮
(mode_n,ex_loc_n,dur_n1,...,dur_np) =
        prog<<n_n1,...,n_np,d_n1,...,d_np>> (el_n,cd_n,es_n or cs_n,cw_n,sw_n);
```

In this instantiation:

- $\mathtt{el}_i$ are controllable variables for execution locations of program i
- $\mathtt{cd}_i$ and $\mathtt{cs}_i$ are controllable variables for switching programs to degraded or safe modes
- $\mathtt{dur}_{ij}$ is:
  - 0 if program i is not executed on PLC j;
  - duration of current mode, if program i is executed on PLC j

Global cost model and control objectives

Computation of total duration of programs on each PLC:

$$\texttt{dur\_plc}_1 = \texttt{dur}_{11} + \ldots + \texttt{dur}_{n1}$$

$$\vdots$$

$$\texttt{dur\_plc}_p = \texttt{dur}_{1p} + \ldots + \texttt{dur}_{np}$$

Global cost model and control objectives

Computation of total duration of programs on each PLC:

$\texttt{dur\_plc}_1 = \texttt{dur}_{11} + \ldots + \texttt{dur}_{n1}$

$\vdots$

$\texttt{dur\_plc}_p = \texttt{dur}_{1p} + \ldots + \texttt{dur}_{np}$

Synthesis objective: cycle duration on PLCs

Duration of execution of programs on PLCs should be less than the cycle time of this PLC

$$\texttt{enforce} \bigwedge_{i=1}^{p} \texttt{dur\_plc}_i \leq \max_i$$

Control objectives (contd)

Synthesis objective: no program on attacked PLCs

$$\texttt{enforce} \bigwedge_{i=1}^{p} \neg\texttt{plc\_avail}_i \Rightarrow (\texttt{dur\_plc}_i = 0)$$

Synthesis objective: dependencies between safe/emergency stops modes

$$\texttt{enforce}\ (\texttt{mode}_i = \texttt{Safe}) \Rightarrow (\texttt{mode}_j = \texttt{Safe})$$

Control objectives (contd)

Synthesis objective: no program on attacked PLCs

$$\texttt{enforce} \bigwedge_{i=1}^{p} \neg\texttt{plc\_avail}_i \Rightarrow (\texttt{dur\_plc}_i = 0)$$

Synthesis objective: dependencies between safe/emergency stops modes

$$\texttt{enforce}\ (\texttt{mode}_i = \texttt{Safe}) \Rightarrow (\texttt{mode}_j = \texttt{Safe})$$

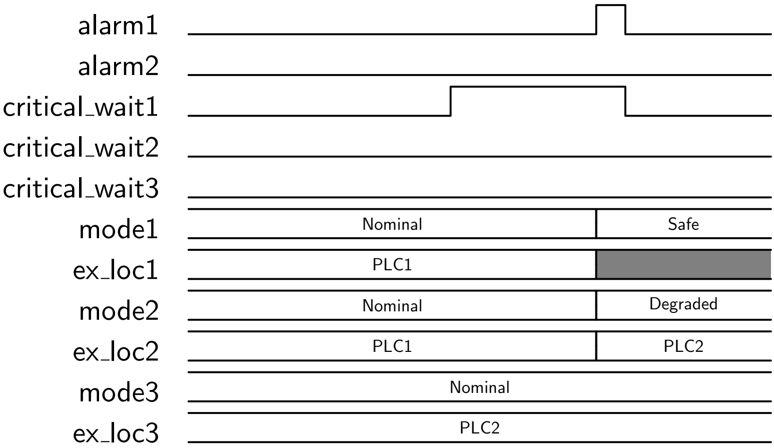
One-step optimization: maximize Nominal modes

```
count₁ = if mode₁ = Nominal then 1 else 0;
...
countₙ = if modeₙ = Nominal then 1 else 0;
count = count₁ + ... + countₙ
```

$\longrightarrow$ maximize count at each execution step

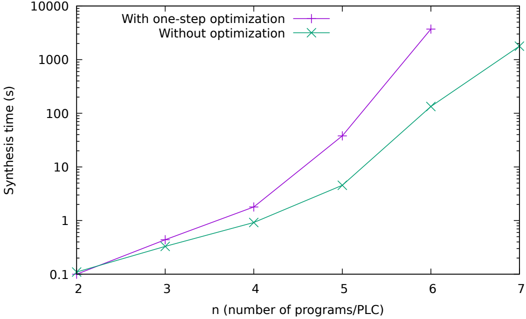
Simulation example

Use-case scenario: 3 programs on 2 PLCs



Scalability

Synthesis time for *n* programs, running on *n* PLCs



Conclusion

Conclusion

- Approach for the cybersecurity of Industrial Control Systems
- Automated reaction by self-protection to attacks
- Automatically produced controller by controller synthesis

Conclusion

Conclusion

- Approach for the cybersecurity of Industrial Control Systems
- Automated reaction by self-protection to attacks
- Automatically produced controller by controller synthesis

Perspectives

- use of modularity, or hierarchical/distributed controllers to handle scalability
- larger size use-case experiment
- consider possible attacks on communication between the self-protection manager and PLCs

Why (not) use Heptagon/BZR?

Gwenaël Delaval

Lig, Université Grenoble Alpes, etc.

Synchron 2021

Heptagon/BZR design cycle

Programmer

BZR chain tool

My program is nondeterministic.
Can you please constrain it so that it satisfies the property P ?

Sure: here is the constraint.

(Actual) Heptagon/BZR design cycle

Programmer

BZR chain tool

My program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

This is not possible.

My (modified) program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

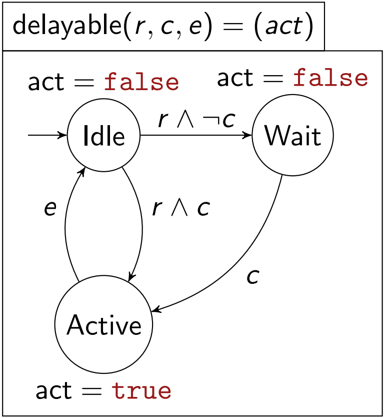
This is not possible.

My (modified) program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

This is not possible.

Example: delayable tasks

```
node delayable(r,c,e:bool) returns (act:bool)
let
  automaton
    state Idle
      do act = false
      unless (r & c) then Active
           | r then Wait
    state Wait
      do act = false
      unless c then Active
    state Active
      do act = true
      unless e then Idle
  end
tel
```



Example (cont'd)

Set of n exlusive delayable tasks

$$\mathtt{ntasks}(\mathtt{r}_1, \ldots, \mathtt{r}_n, \mathtt{e}_1, \ldots, \mathtt{e}_n) = (\mathtt{a}_1, \ldots, \mathtt{a}_n)$$

$$\mathtt{ca}_1 = \mathtt{a}_1 \wedge (\mathtt{a}_2 \vee \ldots \vee \mathtt{a}_n)$$

$$\ldots$$

$$\mathtt{ca}_{n-1} = \mathtt{a}_{n-1} \wedge \mathtt{a}_n$$

$$\mathtt{assume\ true}$$

$$\mathtt{enforce}\ \neg(\mathtt{ca}_1 \vee \ldots \vee \mathtt{ca}_{n-1})$$

$$\mathtt{with}\ \mathtt{c}_1, \ldots, \mathtt{c}_n$$

$$\mathtt{a}_1 = \mathtt{inlined}\ \mathtt{delayable}(\mathtt{r}_1, \mathtt{c}_1, \mathtt{e}_1)$$

$$\ldots$$

$$\mathtt{a}_n = \mathtt{inlined}\ \mathtt{delayable}(\mathtt{r}_n, \mathtt{c}_n, \mathtt{e}_n)$$

Example: composition

$$\mathtt{main}(\mathtt{r}_1, \ldots, \mathtt{r}_{2n}, \mathtt{e}_1, \ldots, \mathtt{e}_{2n}) = (\mathtt{a}_1, \ldots, \mathtt{a}_{2n})$$

$$\mathtt{ca}_1 = \mathtt{a}_1 \wedge (\mathtt{a}_2 \vee \ldots \vee \mathtt{a}_{2n})$$

$$\ldots$$

$$\mathtt{ca}_{2n-1} = \mathtt{a}_{2n-1} \wedge \mathtt{a}_{2n}$$

assume true

enforce $\neg(\mathtt{ca}_1 \vee \ldots \vee \mathtt{ca}_{2n-1})$

with $\emptyset$

$$(\mathtt{a}_1, \ldots, \mathtt{a}_n) = \mathtt{ntasks}(\mathtt{r}_1, \ldots, \mathtt{r}_n, \mathtt{e}_1, \ldots, \mathtt{e}_n)$$

$$(\mathtt{a}_{n+1}, \ldots, \mathtt{a}_{2n}) = \mathtt{ntasks}(\mathtt{r}_{n+1}, \ldots, \mathtt{r}_{2n}, \mathtt{e}_{n+1}, \ldots, \mathtt{e}_{2n})$$

$\longrightarrow$ the contract of `ntasks` is not controllable enough to enforce the `main` contract

Example (refinement, naive version)

Contract refinement for composition of several `ntasks` components:

$$\mathtt{ntasks}(\mathtt{c}, \mathtt{r}_1, \ldots, \mathtt{r}_n, \mathtt{e}_1, \ldots, \mathtt{e}_n) = (\mathtt{a}_1, \ldots, \mathtt{a}_n)$$

$$\begin{aligned}
\mathtt{ca}_1 &= \mathtt{a}_1 \wedge (\mathtt{a}_2 \vee \ldots \vee \mathtt{a}_n) \\
&\ldots \\
\mathtt{ca}_{n-1} &= \mathtt{a}_{n-1} \wedge \mathtt{a}_n \\
\mathtt{one} &= \mathtt{a}_1 \vee \ldots \vee \mathtt{a}_n
\end{aligned}$$

$$\mathbf{assume}\ \mathtt{true}$$

$$\mathbf{enforce}\ \neg(\mathtt{ca}_1 \vee \ldots \vee \mathtt{ca}_{n-1}) \wedge (\mathtt{c} \vee \neg\mathtt{one})$$

$$\mathbf{with}\ \mathtt{c}_1, \ldots, \mathtt{c}_n$$

$$\begin{aligned}
\mathtt{a}_1 &= \mathbf{inlined}\ \mathtt{delayable}(\mathtt{r}_1, \mathtt{c}_1, \mathtt{e}_1) \\
&\ldots \\
\mathtt{a}_n &= \mathbf{inlined}\ \mathtt{delayable}(\mathtt{r}_n, \mathtt{c}_n, \mathtt{e}_n)
\end{aligned}$$

(Actual) Heptagon/BZR design cycle

Programmer

BZR chain tool

My program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

Sure: here is the constraint.

(Actual) Heptagon/BZR design cycle

Programmer

BZR chain tool

My program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

Sure: here is the constraint.

But... The constrained program is stuck in the initial state!

(Actual) Heptagon/BZR design cycle

Programmer

BZR chain tool

My program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

Sure: here is the constraint.

But... The constrained program is stuck in the initial state!

Well... isn't the property satisfied or not?

(Actual) Heptagon/BZR design cycle

Programmer

BZR chain tool

My program is nondeterministic. Can you please constrain it so that it satisfies the property P ?

Sure: here is the constraint.

But... The constrained program is stuck in the initial state!

Well... isn't the property satisfied or not?

Example: composition, 2nd try

$$\mathtt{main}(\mathtt{r}_1, \ldots, \mathtt{r}_{2n}, \mathtt{e}_1, \ldots, \mathtt{e}_{2n}) = (\mathtt{a}_1, \ldots, \mathtt{a}_{2n})$$

$$\mathtt{ca}_1 = \mathtt{a}_1 \wedge (\mathtt{a}_2 \vee \ldots \vee \mathtt{a}_{2n})$$

$$\ldots$$

$$\mathtt{ca}_{2n-1} = \mathtt{a}_{2n-1} \wedge \mathtt{a}_{2n}$$

$$\mathtt{assume\ true}$$

$$\mathtt{enforce}\ \neg(\mathtt{ca}_1 \vee \ldots \vee \mathtt{ca}_{2n-1})$$

$$\mathtt{with\ c}$$

$$(\mathtt{a}_1, \ldots, \mathtt{a}_n) = \mathtt{ntasks}(\mathtt{c}, \mathtt{r}_1, \ldots, \mathtt{r}_n, \mathtt{e}_1, \ldots, \mathtt{e}_n)$$

$$(\mathtt{a}_{n+1}, \ldots, \mathtt{a}_{2n}) = \mathtt{ntasks}(\neg\mathtt{c}, \mathtt{r}_{n+1}, \ldots, \mathtt{r}_{2n}, \mathtt{e}_{n+1}, \ldots, \mathtt{e}_{2n})$$

$\longrightarrow$ Synthesis succeed, but the controllers of `ntasks` cannot allow the tasks to go into the active state !

Example (refinement, correct version)

Use of environment hypothesis to allow more permissive behaviours:

$$
\begin{array}{l}
\texttt{ntasks}(\texttt{c}, \texttt{r}_1, \ldots, \texttt{r}_n, \texttt{e}_1, \ldots, \texttt{e}_n) = (\texttt{a}_1, \ldots, \texttt{a}_n) \\
\hline
\texttt{ca}_1 = \texttt{a}_1 \wedge (\texttt{a}_2 \vee \ldots \vee \texttt{a}_n) \\
\ldots \\
\texttt{ca}_{n-1} = \texttt{a}_{n-1} \wedge \texttt{a}_n \\
\texttt{one} = \texttt{a}_1 \vee \ldots \vee \texttt{a}_n \\
\texttt{pone} = \texttt{false fby one} \\
\texttt{pc} = \texttt{false fby c} \\
\texttt{ppc} = \texttt{false fby pc} \\
\texttt{atleast2} = \neg(\neg\texttt{ppc} \wedge \texttt{pc} \wedge \neg\texttt{c}) \\
\hline
\texttt{assume } (\texttt{pone} \Rightarrow \texttt{c}) \wedge \texttt{atleast2} \\
\texttt{enforce } \neg(\texttt{ca}_1 \vee \ldots \vee \texttt{ca}_{n-1}) \wedge (\neg\texttt{c} \Rightarrow \neg\texttt{one}) \\
\hline
\texttt{with } \texttt{c}_1, \ldots, \texttt{c}_n \\
\hline
\texttt{a}_1 = \texttt{inlined delayable}(\texttt{r}_1, \texttt{c}_1, \texttt{e}_1) \\
\ldots \\
\texttt{a}_n = \texttt{inlined delayable}(\texttt{r}_n, \texttt{c}_n, \texttt{e}_n)
\end{array}
$$

Diagnosis problems (ongoing work)

- Synthesis can fail: information provided to the programmer?
  - model-checking/verification tools: path of input values leading do fault states
  - controller synthesis: dealing with controllable inputs?
  - $\longrightarrow$ tree of uncontrollable/controllable input values

Diagnosis problems (ongoing work)

- Synthesis can fail: information provided to the programmer?
  - model-checking/verification tools: path of input values leading do fault states
  - controller synthesis: dealing with controllable inputs?
  - $\longrightarrow$ tree of uncontrollable/controllable input values
- Over-constrained controller
  - Information to the programmer: set of reachable states? set of "relevant" reachable states?

Diagnosis problems (ongoing work)

- Synthesis can fail: information provided to the programmer?
  - model-checking/verification tools: path of input values leading do fault states
  - controller synthesis: dealing with controllable inputs?
  - $\longrightarrow$ tree of uncontrollable/controllable input values
- Over-constrained controller
  - Information to the programmer: set of reachable states? set of "relevant" reachable states?
- Issues with:
  - modularity (what if synthesis fails because of contracts of subnodes?)
  - abstractions (synthesis on over-approximations)

Actual conclusion (or perspectives?)

Why/in which cases use Heptagon/BZR and controller synthesis?

- It is fun!
- Do automatically part of the programming work: useful in
  - Closed systems
  - Where part of the problem is combinatorics
  - Where system can be easily modelled as Boolean/basic numerical equations

Actual conclusion (or perspectives?)

Why/in which cases use Heptagon/BZR and controller synthesis?

- It is fun!
- Do automatically part of the programming work: useful in
  - Closed systems
  - Where part of the problem is combinatorics
  - Where system can be easily modelled as Boolean/basic numerical equations

Why you shouldn't actually use it?

- Comparison with real-time scheduling / constraint programming not clear (TBD)
- Under capitalism, trying to automate other one's jobs can be a bad idea
- Controller synthesis is not climate-friendly
- … and no control on the rebound effect.