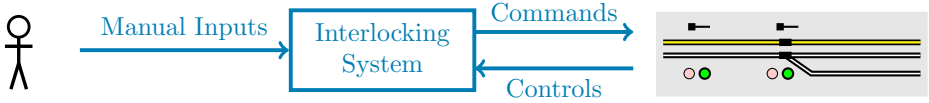
Comparing legacy and modern implementations: migrating from analogical to software-based Railway Interlocking Systems

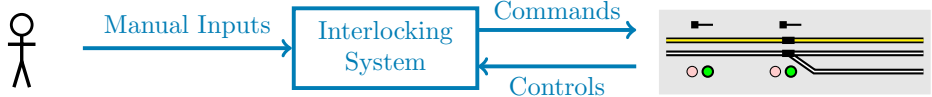
Anna Becchi Alessandro Cimatti

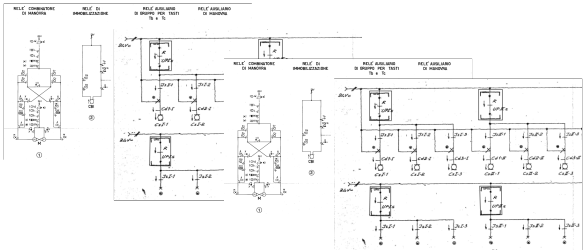
12 October 2023

Industrial example: italian Railway Interlocking Systems



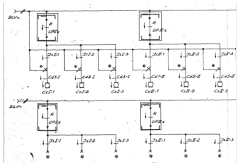
Industrial example: italian Railway Interlocking Systems





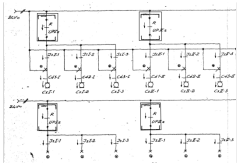
Migration from analogical to software-based

Relay-based circuits (RRIS)

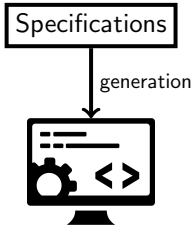


Migration from analogical to software-based

Relay-based circuits (RRIS)

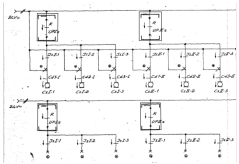


Software (SwRIS)

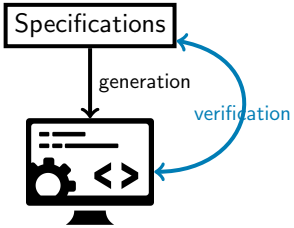


Migration from analogical to software-based

Relay-based circuits (RRIS)

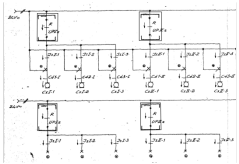


Software (SwRIS)



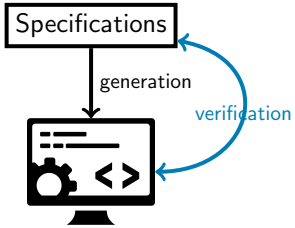
Migration from analogical to software-based

Relay-based circuits (RRIS)



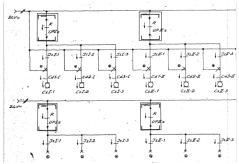
=?

Software (SwRIS)



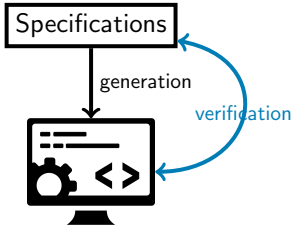
Migration from analogical to software-based

Relay-based circuits (RRIS)



=?

Software (SwRIS)

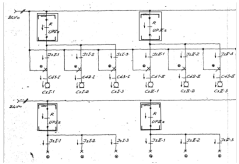


Electrical variables

Software variables

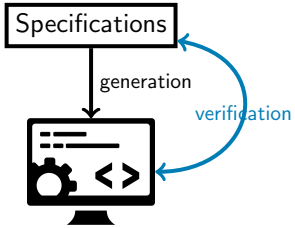
Migration from analogical to software-based

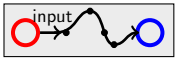
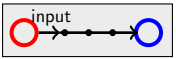
Relay-based circuits (RRIS)



=?

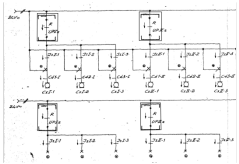
Software (SwRIS)



Electrical variables	Software variables
Continuous time computational model 	Cycle-based computational model 

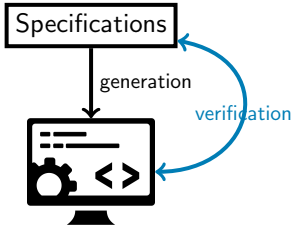
Migration from analogical to software-based

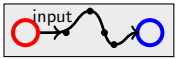
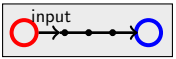
Relay-based circuits (RRIS)



=?

Software (SwRIS)



Electrical variables	Software variables
Continuous time computational model 	Cycle-based computational model 
Instantiated	Parametric

Activities

✓ Making available Relay-based RIS models: from drawings to timed transition systems

Activities

- ✓ Making available Relay-based RIS models: from drawings to timed transition systems
- ✓ Correlate different computational models with Abstraction Modulo Stability

Activities

- ✓ Making available Relay-based RIS models: from drawings to timed transition systems
- ✓ Correlate different computational models with Abstraction Modulo Stability
- Consider a specific configuration p:

Activities

- ✓ Making available Relay-based RIS models: from drawings to timed transition systems
- ✓ Correlate different computational models with Abstraction Modulo Stability
- Consider a specific configuration p:
  - Testing $\mathrm{RRIS}[p] \subseteq \mathrm{SwRIS}[p]$

Activities

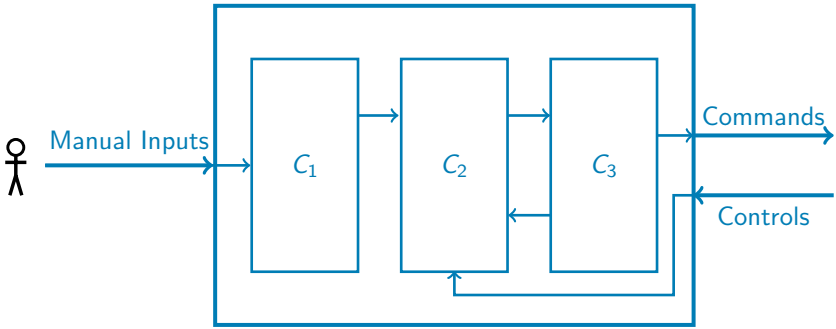
- ✓ Making available Relay-based RIS models: from drawings to timed transition systems
- ✓ Correlate different computational models with Abstraction Modulo Stability
- Consider a specific configuration p:
  - Testing $\mathrm{RRIS}[p] \subseteq \mathrm{SwRIS}[p]$
  - Proving $\mathrm{RRIS}[p] = \mathrm{SwRIS}[p]$

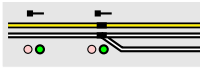
Activities

- ✓ Making available Relay-based RIS models: from drawings to timed transition systems
- ✓ Correlate different computational models with Abstraction Modulo Stability
- Consider a specific configuration p:
  - Testing $\mathrm{RRIS}[p] \subseteq \mathrm{SwRIS}[p]$
  - Proving $\mathrm{RRIS}[p] = \mathrm{SwRIS}[p]$
- Many other open problems...

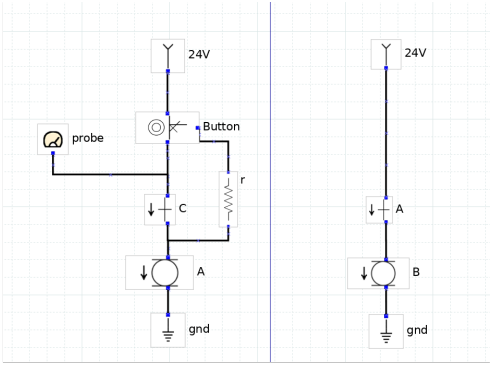
Plan of the talk

Relay-based Railway Interlocking Systems

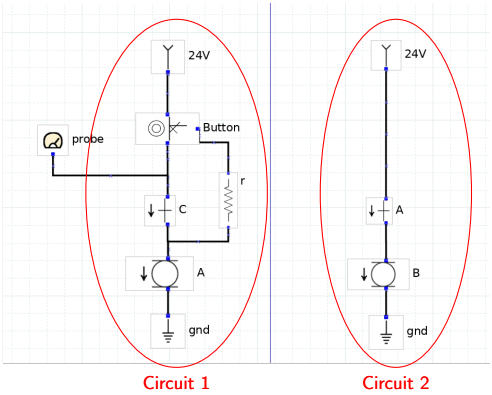




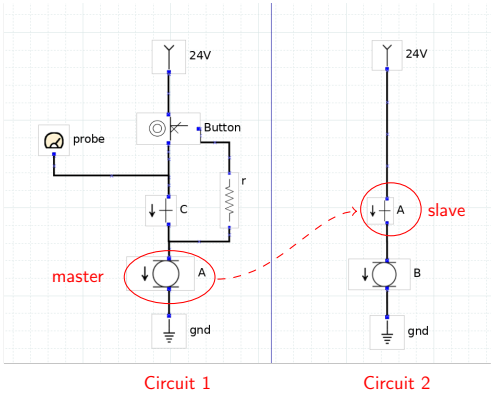
Relay-based Railway Interlocking Systems



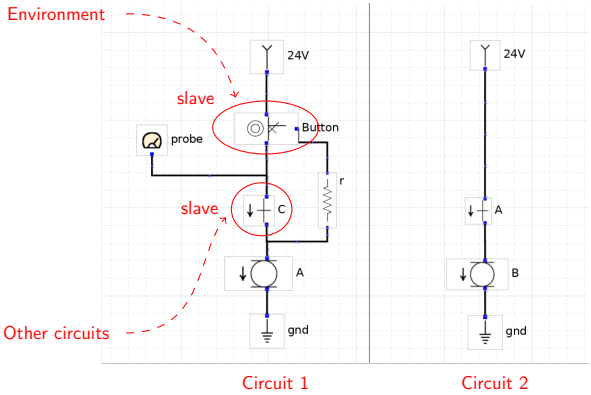
Relay-based Railway Interlocking Systems



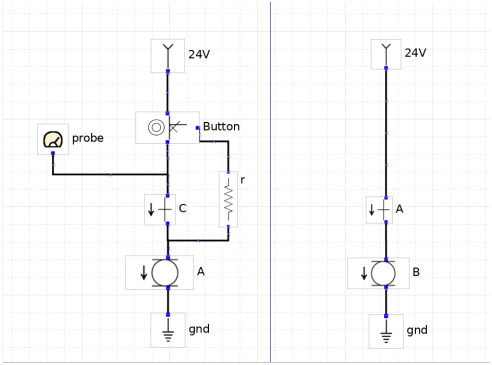
Relay-based Railway Interlocking Systems



Relay-based Railway Interlocking Systems

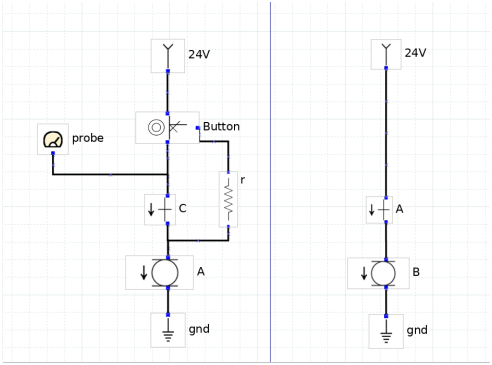


Relay-based Railway Interlocking Systems



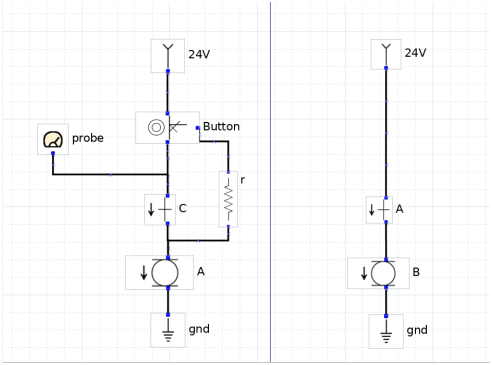
Circuit specified with Kirchhoff laws $C_1(I_1, E_1, O_1)$ $C_2(I_2, E_2, O_2)$

Relay-based Railway Interlocking Systems



Remove Electrical variables $C_1^{(opt)}(I_1, O_1) = \exists E_1 \,.\, C_1$ $C_2^{(opt)}(I_2, O_2) = \exists E_2 \,.\, C_2$

Relay-based Railway Interlocking Systems



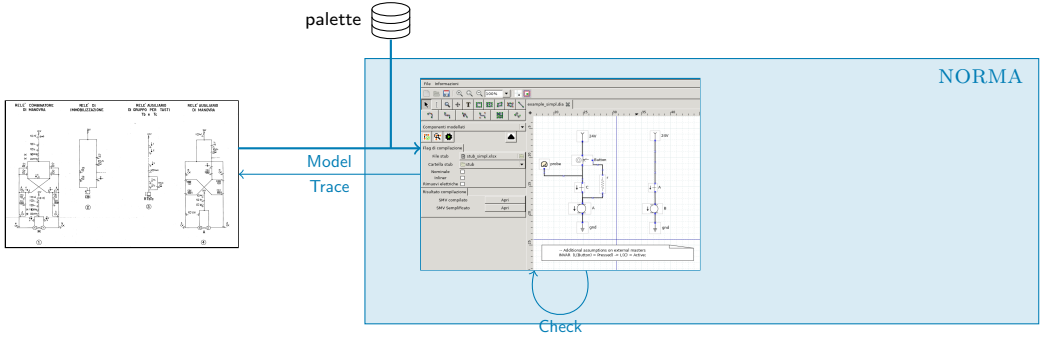
$C_1(I_1, O_1)$ $C_2(I_2, O_2)$

(If needed) break cycles

$A.switch = next(A.coil)$

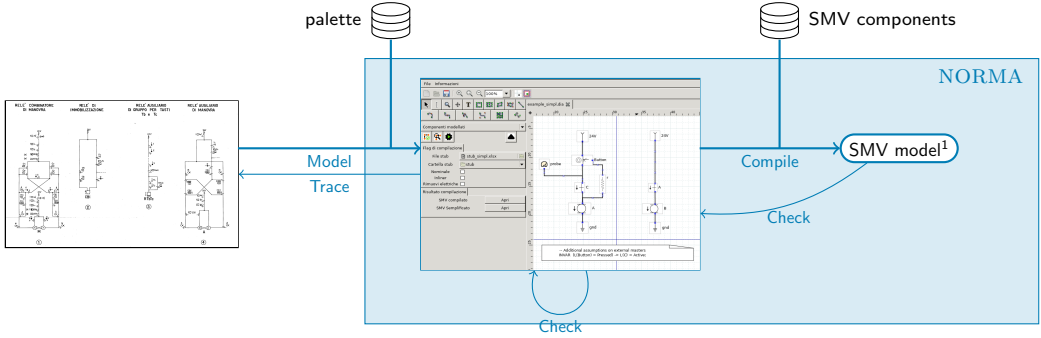
$\texttt{URGENT}\ A.switch \neq A.coil$

Norma: a compiler from RRIS to Timed SMV [TACAS22]



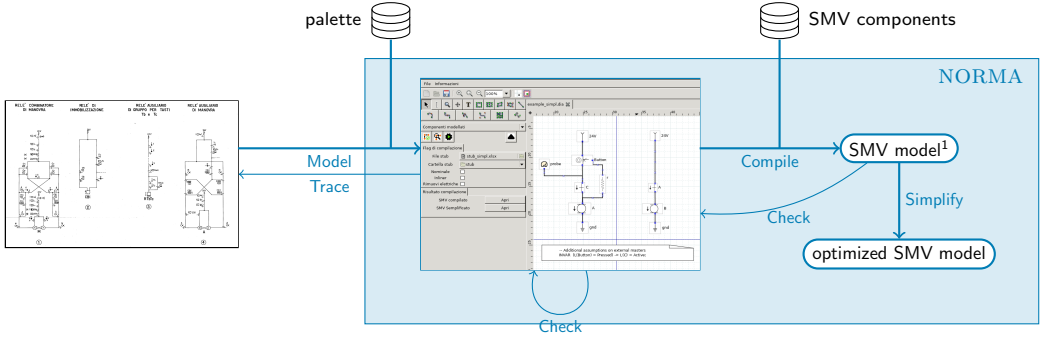
1

Norma: a compiler from RRIS to Timed SMV [TACAS22]



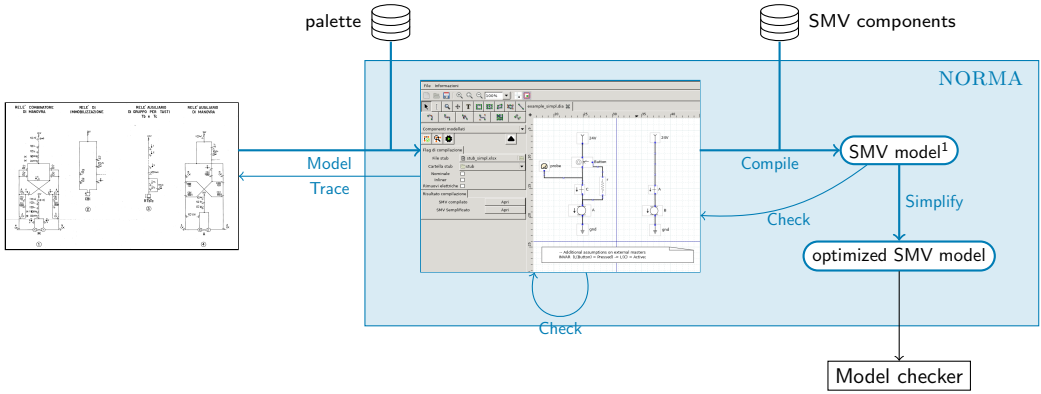
[1]Symbolic infinite-state timed transition system: Timed nuXmv [CimattiGMRT – CAV'19]

Norma: a compiler from RRIS to Timed SMV [TACAS22]



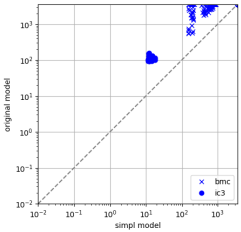
[1]Symbolic infinite-state timed transition system: Timed nuXmv [CimattiGMRT – CAV'19]

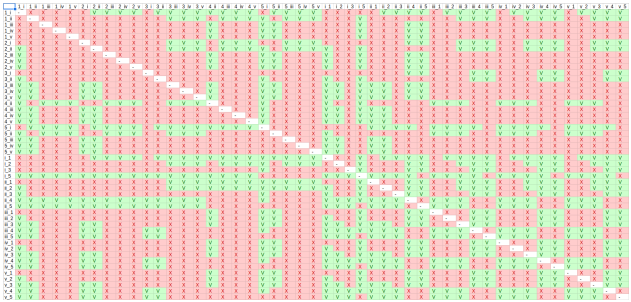
Norma: a compiler from RRIS to Timed SMV [TACAS22]



[1]Symbolic infinite-state timed transition system: Timed nuXmv [CimattiGMRT – CAV'19]

Norma outcomes

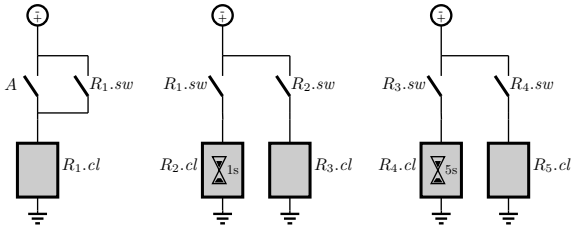




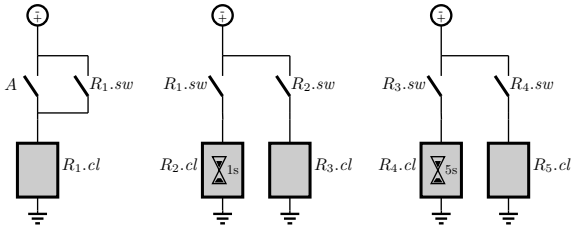
Plan of the talk

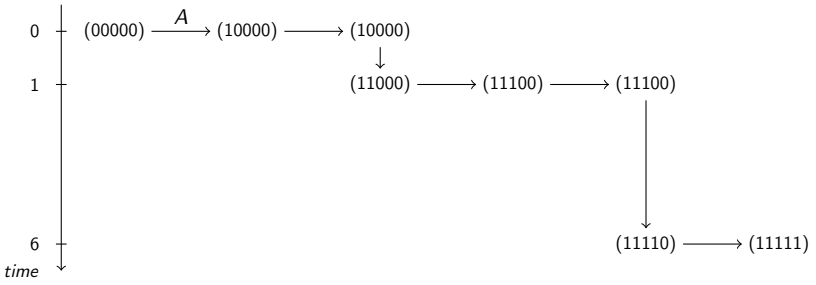
1. Making available the RRIS models
2. Abstraction Modulo Stability
3. Testing $\mathrm{RRIS} \subseteq \mathrm{SwRIS}$
4. Remaining problems

RRIS execution

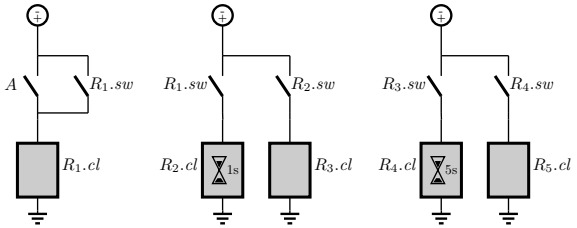


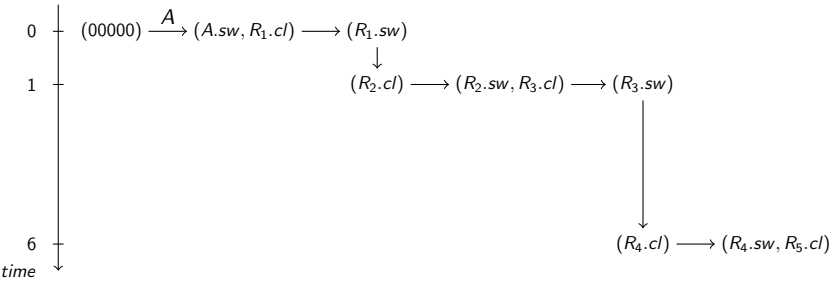
RRIS execution



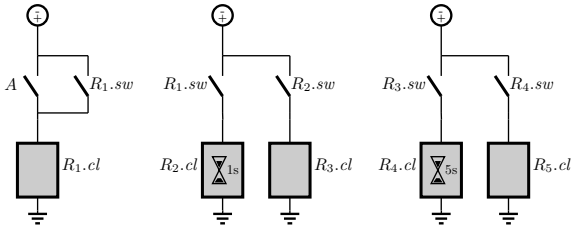


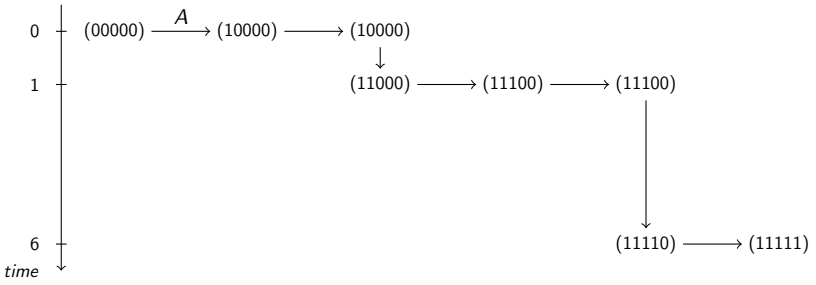
RRIS execution



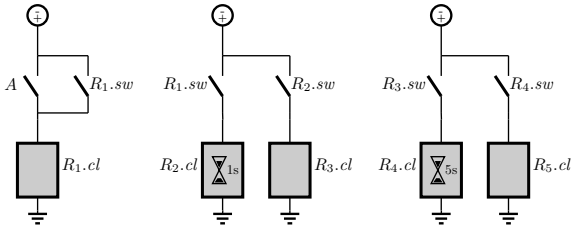


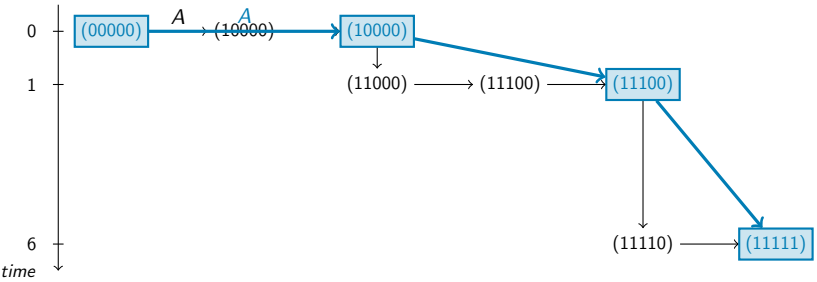
RRIS execution



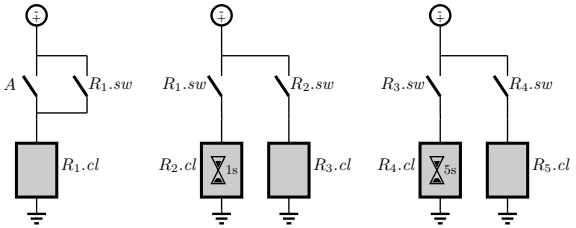


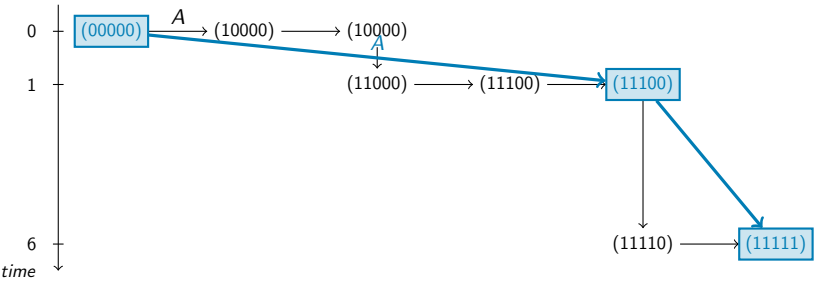
RRIS execution



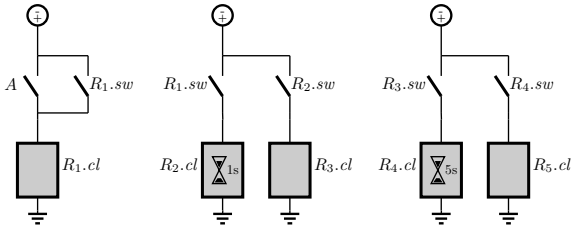


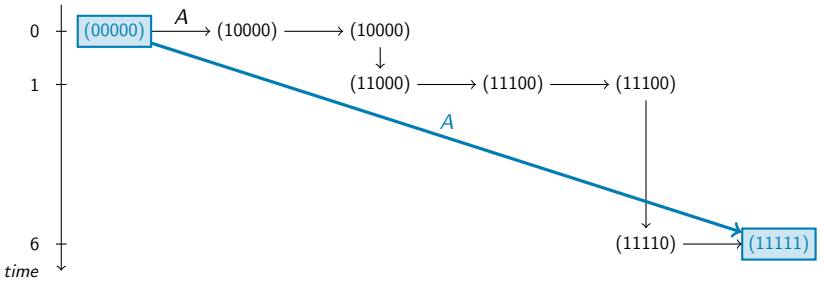
RRIS execution





RRIS execution





Abstraction Modulo Stability [CAV22]

Given:

- $\mathcal{M} = \langle X, C, I, \mathrm{Init}(X), \mathrm{Trans}(X, I, X') \rangle$ timed transition system
- $\mathcal{M}^\tau$ closed system: $\mathrm{Trans}^\tau := \mathrm{Trans} \wedge (I = I')$
- P set of predicate variables ($P \subseteq X$)
- σ stability definition: $[\![\phi_\sigma]\!] = \{s \mid \mathcal{M}^\tau, s \models \sigma\}$ are the stable states

Abstraction Modulo Stability [CAV22]

Given:

- $\mathcal{M} = \langle X, C, I, \mathrm{Init}(X), \mathrm{Trans}(X, I, X') \rangle$ timed transition system
- $\mathcal{M}^\tau$ closed system: $\mathrm{Trans}^\tau := \mathrm{Trans} \wedge (I = I')$
- P set of predicate variables ($P \subseteq X$)
- σ stability definition: $[\![\phi_\sigma]\!] = \{s \mid \mathcal{M}^\tau, s \models \sigma\}$ are the stable states

$\mathrm{AMS}(P, \sigma)$: *TTS* $\rightarrow$ *FSM*

Abstraction Modulo Stability [CAV22]

Given:

- $\mathcal{M} = \langle X, C, I, \mathrm{Init}(X), \mathrm{Trans}(X, I, X') \rangle$ timed transition system
- $\mathcal{M}^\tau$ closed system: $\mathrm{Trans}^\tau := \mathrm{Trans} \wedge (I = I')$
- P set of predicate variables ($P \subseteq X$)
- σ stability definition: $[\![\phi_\sigma]\!] = \{s \mid \mathcal{M}^\tau, s \models \sigma\}$ are the stable states

$\mathrm{AMS}(P, \sigma)$: *TTS* $\rightarrow$ *FSM*

- $\mathrm{AMS}(P, \sigma)(\mathcal{M}) = \mathcal{A} = \langle P, I, \mathrm{Init}_{\mathcal{A}}(P), \mathrm{Trans}_{\mathcal{A}}(P, I, P') \rangle$

Abstraction Modulo Stability [CAV22]

Given:

- $\mathcal{M} = \langle X, C, I, \mathrm{Init}(X), \mathrm{Trans}(X, I, X') \rangle$ timed transition system
- $\mathcal{M}^\tau$ closed system: $\mathrm{Trans}^\tau := \mathrm{Trans} \wedge (I = I')$
- P set of predicate variables ($P \subseteq X$)
- σ stability definition: $[\![\phi_\sigma]\!] = \{s \mid \mathcal{M}^\tau, s \models \sigma\}$ are the stable states

$\mathrm{AMS}(P, \sigma)$: *TTS* $\rightarrow$ *FSM*

- $\mathrm{AMS}(P, \sigma)(\mathcal{M}) = \mathcal{A} = \langle P, I, \mathrm{Init}_\mathcal{A}(P), \mathrm{Trans}_\mathcal{A}(P, I, P') \rangle$
- $p_0 \models \mathrm{Init}_\mathcal{A}$ iff

$$\mathcal{M}^\tau \models_\exists (\neg \phi_\sigma \ \mathrm{U} \ (\phi_\sigma \wedge p_0))$$

Abstraction Modulo Stability [CAV22]

Given:

- $\mathcal{M} = \langle X, C, I, \mathrm{Init}(X), \mathrm{Trans}(X, I, X') \rangle$ timed transition system
- $\mathcal{M}^\tau$ closed system: $\mathrm{Trans}^\tau := \mathrm{Trans} \wedge (I = I')$
- P set of predicate variables ($P \subseteq X$)
- σ stability definition: $[\![\phi_\sigma]\!] = \{s \mid \mathcal{M}^\tau, s \models \sigma\}$ are the stable states

$\mathrm{AMS}(P, \sigma)$: *TTS* $\rightarrow$ *FSM*

- $\mathrm{AMS}(P, \sigma)(\mathcal{M}) = \mathcal{A} = \langle P, I, \mathrm{Init}_\mathcal{A}(P), \mathrm{Trans}_\mathcal{A}(P, I, P') \rangle$
- $p_0 \models \mathrm{Init}_\mathcal{A}$ iff

$$\mathcal{M}^\tau \models_\exists (\neg\phi_\sigma \ \mathrm{U} \ (\phi_\sigma \wedge p_0))$$

- $(p_1, in, p_2) \models \mathrm{Trans}_\mathcal{A}$ iff

$$\mathcal{M} \models_\exists \mathrm{F}\left((\phi_\sigma \wedge p_1) \wedge \left(\mathrm{G}in \wedge \mathrm{X}(\neg\phi_\sigma \ \mathrm{U} \ (\phi_\sigma \wedge p_2))\right)\right).$$

How to choose σ

- "predicate abstraction" $\sigma := \top$
- **non-urgent abstraction** $\sigma := \exists X', I \,.\, \mathrm{Trans}(X, I, X') \wedge \delta > 0$
- T-time abstraction ($T \in \mathbb{R}_+$) $\sigma := \mathrm{Y}(\delta > T)$
- same-predicate abstraction $\sigma := \mathrm{AG}(P = \mathrm{X}P)$

Plan of the talk

Testing $\mathrm{RRIS}[p] \subseteq \mathrm{SwRIS}[p]$

$$\forall \pi \in \mathrm{Simulate}(\mathrm{RRIS}[p]) \,.\, \mathtt{test}_{\mathrm{SwRIS}[p]}(\mu(\pi))$$

where:

- $\mathrm{Simulate}$: $\mathcal{M} \to \wp(\Pi_{\mathcal{M}})$ paths generator
- μ: $\Pi_{\mathrm{RRIS}[p]} \to \mathit{Testcases}_{\mathrm{SwRIS}[p]}$ mapping
- $\mathtt{test}_{\mathrm{SwRIS}[p]}$ test executor for $\mathrm{SwRIS}[p]$

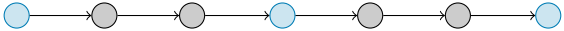
Test executor: TOSCA

A tool developed by FBK for RFI. Provided functionalities:

- Specify abstract test cases in Controlled Natural Language
  - sequence of steps specifying *events*, *commands*, *assumptions*, *assertions*
  - does not need to refer to a specific station
- Automatically instantiate abstract tests in executable test cases
- Execute test cases on the code + simulator of the Environment
- Evaluate coverage level on the code *and on the abstract model*

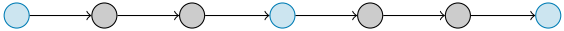
Mapping

π



Mapping

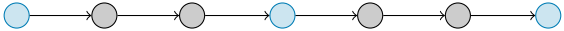
π



$\mathrm{AMS}(\pi)$

Mapping

π



$\mathrm{AMS}(\pi)$

TOSCA abstract test case

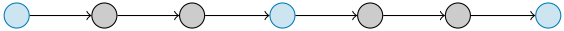
```
step 1:
 command X1;
 within 100 cycles:
  assume Y1;
 within 100 cycles:
  assert Z1;
```

```
step 2:
 command X2;
 within 100 cycles:
  assume Y2;
 within 100 cycles:
  assert Z2;
```

```
step 3:
 command X3;
 within 100 cycles:
  assume Y3;
 within 100 cycles:
  assert Z3;
```

Mapping

π



$\mathrm{AMS}(\pi)$

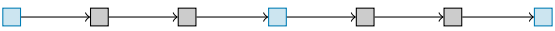
TOSCA abstract test case

```
step 1:
 command X1;
 within 100 cycles:
  assume Y1;
 within 100 cycles:
  assert Z1;
```

```
step 2:
 command X2;
 within 100 cycles:
  assume Y2;
 within 100 cycles:
  assert Z2;
```

```
step 3:
 command X3;
 within 100 cycles:
  assume Y3;
 within 100 cycles:
  assert Z3;
```

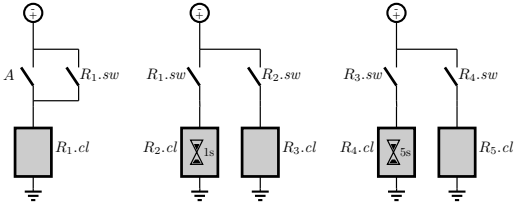
SwRIS execution

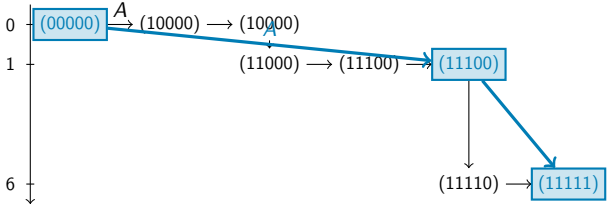


Mapping

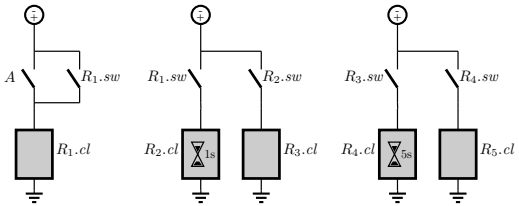
	A	B	C	D	E	F	G
1	Espressione TOSCA per settare	Espressione TOSCA per assumere	Espressione TOSCA per verificare	SdP	Tipo	Condizione	Espressione SdP (true=eccitato;false=diseccitato)
2	comando manuale Confermaton a dev1			SDe02	Comando	da Banco	!SDP_X_RIGHT & SDP_X_LEFT & !SDP_Tb_LEFT & !SDP_Tc_LEF
3	comando manuale Confermator a dev1			SDe02	Comando	da Banco	SDP_X_RIGHT & !SDP_X_LEFT & !SDP_Tb_LEFT & !SDP_Tc_LEF
4	comando manuale Tbn a dev1			SDe02	Comando	da Banco	!SDP_X_RIGHT & SDP_X_LEFT & SDP_Tb_LEFT & !SDP_Tc_LEF
5	comando manuale Tbr a dev1			SDe02	Comando	da Banco	SDP_X_RIGHT & !SDP_X_LEFT & SDP_Tb_LEFT & !SDP_Tc_LEF
6	comando manuale Tcn a dev1			SDe02	Comando	da Banco	!SDP_X_RIGHT & SDP_X_LEFT & !SDP_Tb_LEFT & SDP_Tc_LEF
7	comando manuale Tcr a dev1			SDe02	Comando	da Banco	SDP_X_RIGHT & !SDP_X_LEFT & !SDP_Tb_LEFT & SDP_Tc_LEF
8	comando manuale Tbtcn a dev1			SDe02	Comando	da Banco	!SDP_X_RIGHT & SDP_X_LEFT & SDP_Tb_LEFT & SDP_Tc_LEF
9	comando manuale Tbtcr a dev1			SDe02	Comando	da Banco	SDP_X_RIGHT & !SDP_X_LEFT & SDP_Tb_LEFT & SDP_Tc_LEF
10	comando manuale Funzionamento_automatico a dev1			SDe02	Comando	da Banco	!SDP_X_RIGHT & !SDP_X_LEFT
11	comando manuale Rte a rte1			SDe02	Comando	da Banco	SDP_Te_LEFT
12	comando manuale Alimentazio a dev1			SDe02	Comando	impulsivo d	SDP_ALIM_impulso
13	comando manuale Disalimentazione a dev1			SDe02	Comando	impulsivo d	SDP_DISAL_impulso
14	comando manuale Attivazione_pesante a iti1			SDe02	Comando	impulsivo d	SDP_Attivazione_iti_N_impulso
15	comando manuale Attivazione_pesante a iti2			SDe02	Comando	impulsivo d	SDP_Attivazione_iti_R_impulso
16	comando manuale Distruzione_pesante a po1			SDe02	Comando	impulsivo d	SDP_Distruzione_iti_N_impulso
17	comando manuale Distruzione_pesante a po2			SDe02	Comando	impulsivo d	SDP_Distruzione_iti_R_impulso
18	applica forzatura frz_fuori_controllo_elettrico a dev1	assumi che posizione di dev1 sia uguale a nope		SDe02	Comando	Stub	SDP_frz_fuori_controllo_elettrico
19	rimuovi forzatura frz_fuori_controllo_elettrico da dev1			SDe02	Comando	Stub	!SDP_frz_fuori_controllo_elettrico
20	applica forzatura frz_no_pos_normale a dev1	assumi che posizione di dev1 sia diverso da sinistra		SDe02	Comando	Stub	SDP_frz_no_pos_normale
21	rimuovi forzatura frz_no_pos_normale da dev1			SDe02	Comando	Stub	!SDP_frz_no_pos_normale
22	applica forzatura frz_no_pos_rovescio a dev1	assumi che posizione di dev1 sia diverso da destra		SDe02	Comando	Stub	SDP_frz_no_pos_rovescio
23	rimuovi forzatura frz_no_pos_rovescio da dev1			SDe02	Comando	Stub	!SDP_frz_no_pos_rovescio
24	applica forzatura frz_non_cambia_posiz a dev1			SDe02	Comando	Stub	SDP_frz_non_cambia_posiz
25	rimuovi forzatura frz_non_cambia_posiz da dev1			SDe02	Comando	Stub	!SDP_frz_non_cambia_posiz
26	applica forzatura frz_non_si_diseccita a ele1	assumi che basso di ele1 sia uguale a false		SDe02	Comando	Stub	SDP_frz_non_si_diseccita
27	rimuovi forzatura frz_non_si_diseccita da ele1			SDe02	Comando	Stub	!SDP_frz_non_si_diseccita
28	applica forzatura frz_non_si_eccita a ele1	assumi che basso di ele1 sia uguale a true		SDe02	Comando	Stub	SDP_frz_non_si_eccita
29	rimuovi forzatura frz_non_si_eccita da ele1			SDe02	Comando	Stub	!SDP_frz_non_si_eccita
30			verifica che statocontrollo di dev1 sia a	SDe02	Elemento	interno osse	!SDP_CDX_SX & !SDP_CDX_DX
31			verifica che statocontrollo di dev1 sia a	SDe02	Elemento	interno osse	SDP_CDX_SX & !SDP_CDX_DX
32			verifica che statocontrollo di dev1 sia a	SDe02	Elemento	interno osse	!SDP_CDX_SX & SDP_CDX_DX
33			verifica che statologico di dev1 sia ugu	SDe02	Elemento	interno osse	SDP_M_normale
34			verifica che statologico di dev1 sia ugu	SDe02	Elemento	interno osse	!SDP_M_normale
35		assumi che bloccatop di dev1 sia diverso da 0		SDe02	Input da	altre tavole	!SDP_bc | !SDP_bp
36		assumi che bloccatop di dev1 sia uguale a 0		SDe02	Input da	altre tavole	SDP_bc & SDP_bp
37		assumi che conta_prenotazioni_n di dev1 sia diverso da 0		SDe02	Input da	altre tavole	SDP_Cn
38		assumi che conta_prenotazioni_n di dev1 sia uguale a 0		SDe02	Input da	altre tavole	!SDP_Cn
39		assumi che conta_prenotazioni_r di dev1 sia diverso da 0		SDe02	Input da	altre tavole	SDP_Cr
40		assumi che conta_prenotazioni_r di dev1 sia uguale a 0		SDe02	Input da	altre tavole	!SDP_Cr
41	rimuovi forzatura frz_occupazione da cdb1	assumi che libertaBinario di cdb1 sia uguale a true		SDe02	Input da	altre tavole/SP	!SDP_CB
42	applica forzatura frz_occupazione a cdb1	assumi che libertaBinario di cdb1 sia uguale a false		SDe02	Input da	altre tavole/SP	!SDP_CB
43		assumi che alimentazione di dev1 sia uguale a true		SDe02	Input esterno		SDP_IX

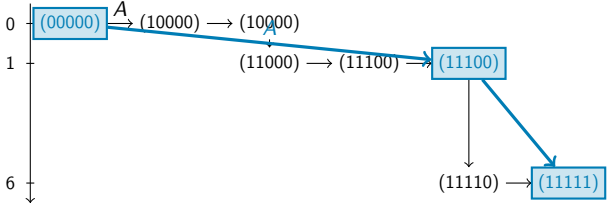
Mapping example





Mapping example





A	: "command A";
$R_3.cl$	: "assume Gate up";
$\neg R_3.cl$	: "assume Gate down";
$R_5.cl$	: "assert Light On";
$\neg R_5.cl$	: "assert Light Off";

`mapping.json`

```
step 0:  // initial state
assume Gate down
assert Light Off

step 1:  command A
within 100 cycles:  assume Gate up
within 100 cycles:  assert Light Off

step 2:
within 100 cycles:  assume Gate down
within 100 cycles:  assert Light On
```

`test.atosca`

Paths generator

- Random simulations
- subset-wise coverage of relays
- coverage of "negative and positive proofs" for each circuits:

Paths generator

- Random simulations
- subset-wise coverage of relays
- coverage of "negative and positive proofs" for each circuits: use AMS

Paths generator

- Random simulations
- subset-wise coverage of relays
- coverage of "negative and positive proofs" for each circuits: use AMS

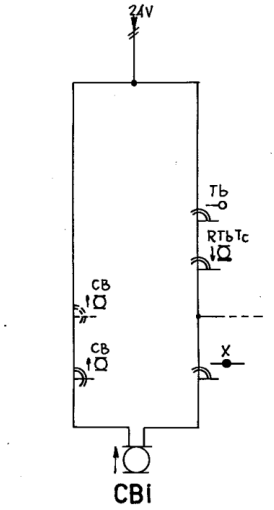
For $p_1, p_2 \in 2^P$ and $in \in 2^I$:

$$\mathrm{witness}(p_1, in, p_2) := \left\{ \pi \in \Pi_{\mathcal{M}} \;\middle|\; \pi \models \mathrm{F}\left((\phi_\sigma \wedge p_1) \wedge \left(\mathrm{G}\, in \wedge \mathrm{X}(\neg\phi_\sigma \;\mathrm{U}\; (\phi_\sigma \wedge p_2))\right)\right) \right\}$$

For $|P| = 1$:

- Positive proofs : $\{\pi \mid \pi \in \mathrm{witness}(p, in, \neg p)\}$
- Negative proofs : $\{\pi \mid \pi \in \mathrm{witness}(p, in, p)\}$

Positive and negative proofs



46 paths extracted. Covering:

- 24 input combinations for which CBi does not change status
- 22 input combinations for which CBi changes status

Wrapping up

Plan of the talk

1. Making available the RRIS models
2. Abstraction Modulo Stability
3. Testing $\mathrm{RRIS} \subseteq \mathrm{SwRIS}$
4. Remaining problems

Future directions

- Apply Abstraction Modulo Stability to LdS: symmetric checks
- What coverage criterion for $\mathrm{Simulate}$ guarantees to consider all RRIS behaviors?
- Use Abstraction Modulo Stability to generalize from instances to abstract concepts

Future directions

- Apply Abstraction Modulo Stability to LdS: symmetric checks
- What coverage criterion for $\mathrm{Simulate}$ guarantees to consider all RRIS behaviors?
- Use Abstraction Modulo Stability to generalize from instances to abstract concepts