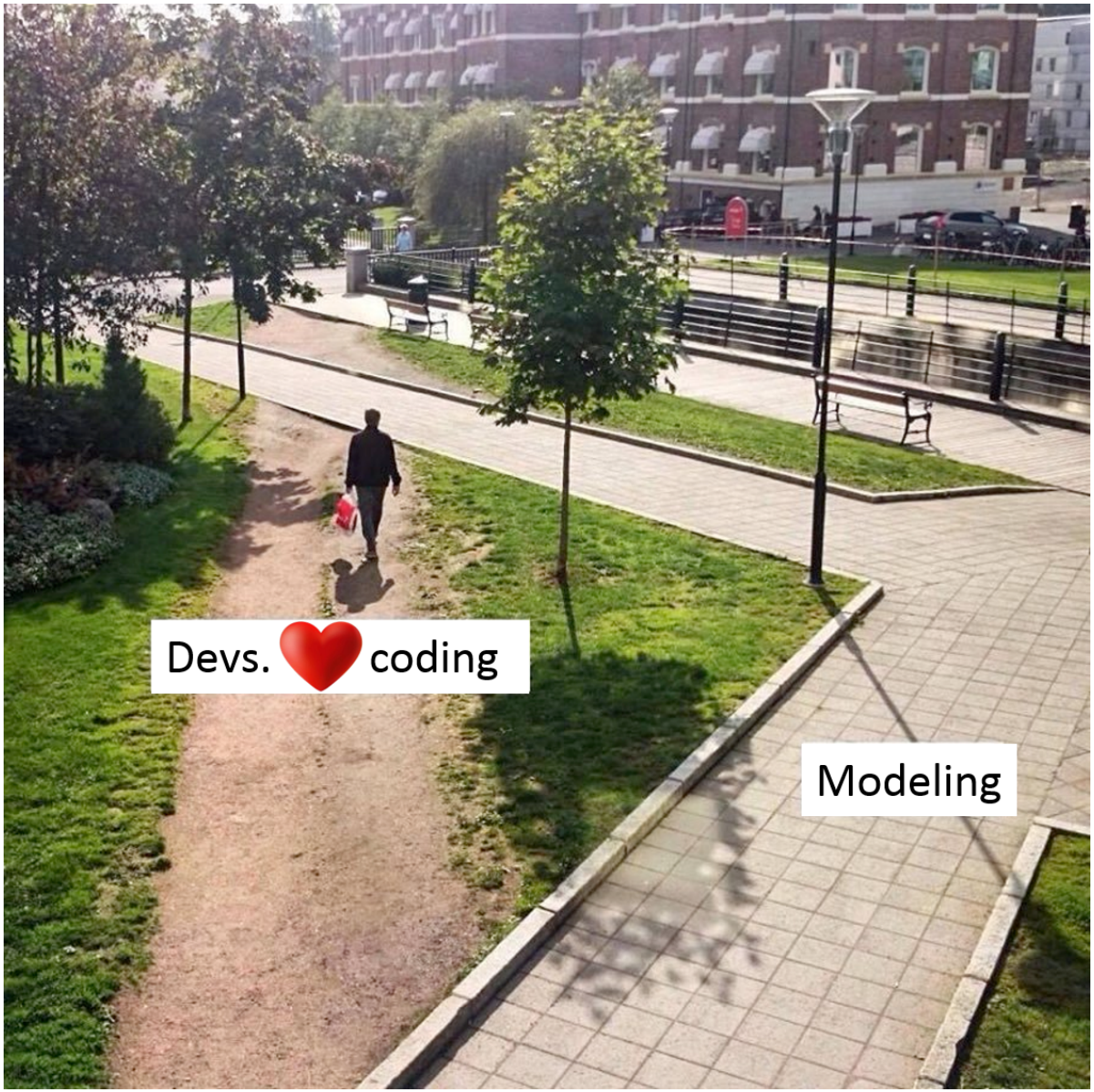
COnfECt: An Approach to Learn Models of Component-based Systems.

Sébastien Salva, Elliott Blot

LIMOS, Clermont Auvergne University, France

ICSOFT 2018: 13th International Conference on Software Technologies
28 July 2018


Devs. ❤️ coding
Modeling

Model Learning

Generation of behavioral models from a black box application. (by retro-engineering it).

- Models : for documentation, analysis, auto generation of test cases.
- Several limitations:
  Working app. or execution traces
  Modeling behaviors of components of an application.

COnfECt

Model learning method, generates 1 model /component from execution traces.

Assumptions

- No knowledge of the components
- Components have different behaviors

Goal:

Use data mining to detect different behaviors in traces $\rightarrow$ models
Provide strategies to synchronize models together

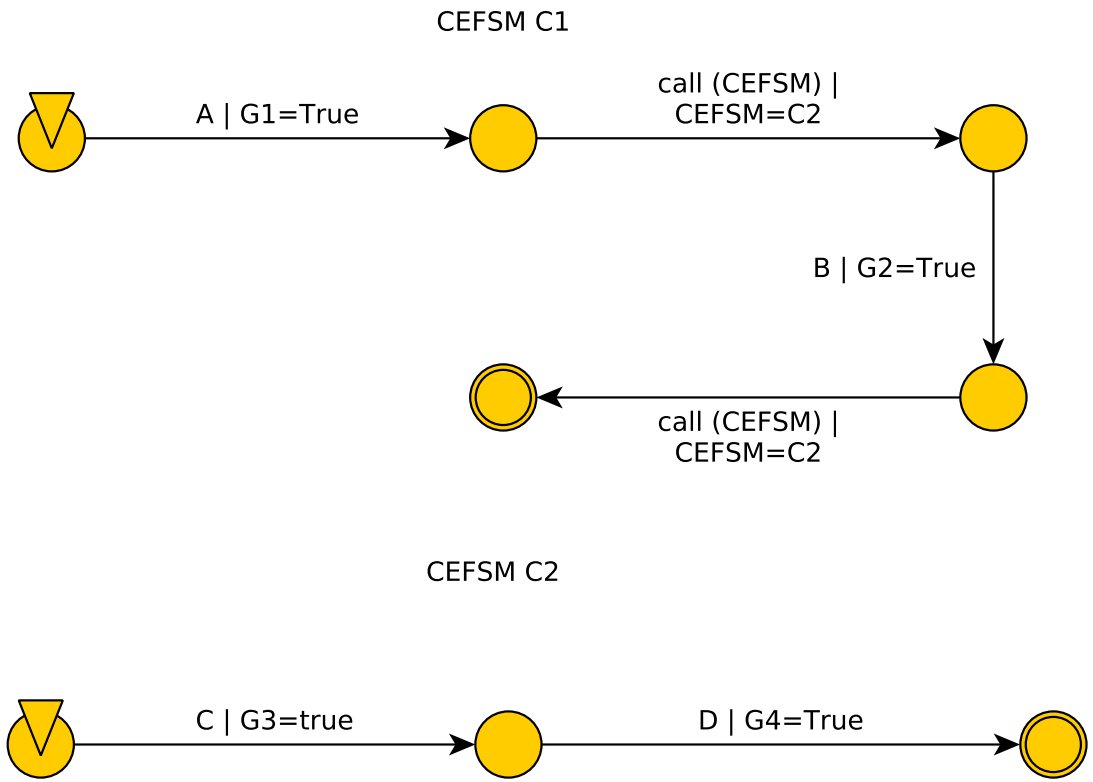
Contents

1. Approach & Overview
2. Steps details
3. Conclusion

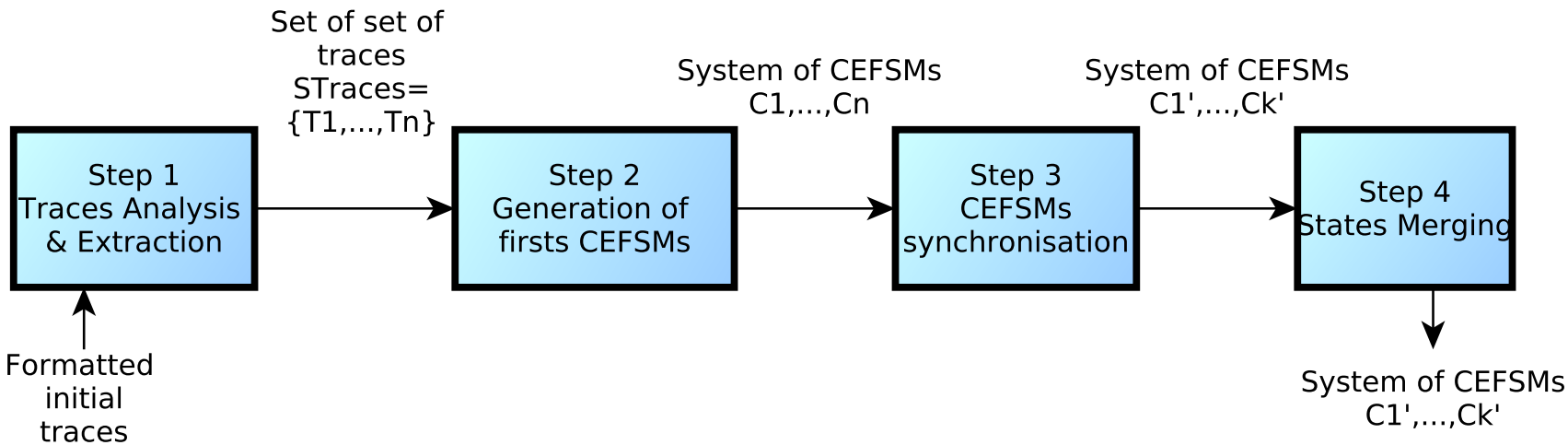
CEFSM (Callable Extended Finite State Machine)

Generation of systems of CEFSMs

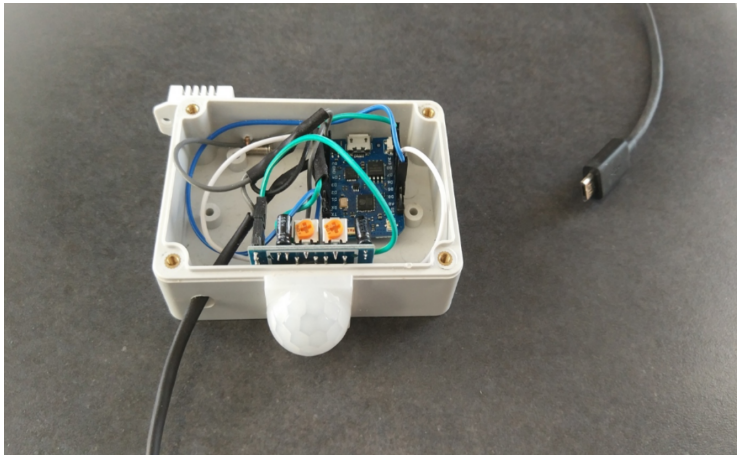
CEFSM = FSM + variables + special internal event: *call(CEFSM)*.



Split / Merge algorithm:



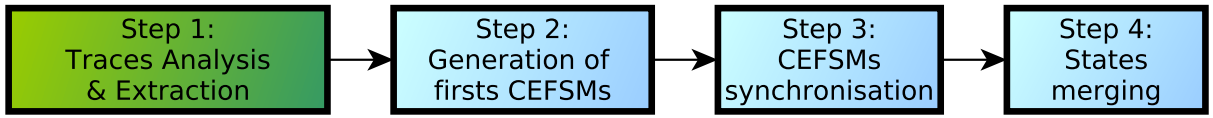
Case study :
smart connected thermostat.



2 components in the system:

- Web interface
- Sensors (motion detector, temperature sensor)

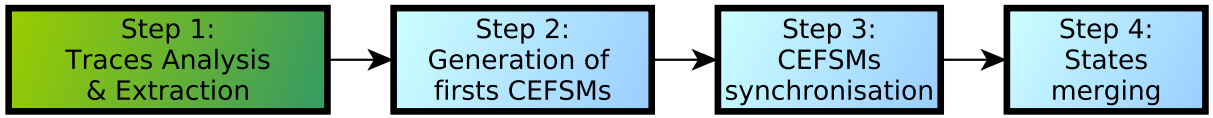
provides HTTP traces



Trace analysis with data mining (string similarities, correlation factors, etc.)

```
e1:  ?/devices (Verb=GET, Uri=devices)
e2:  !/json.htm (param=udevice, 15<=svalue<=16)
e3:  ?Response (response=ok)
e4:  !Response (response=ok, data=[1])
e5:  !/json.htm (param=udevice, 15<=svalue<=16)
e6:  ?Response (response=ok)
e7:  ?/devices (Verb=GET, Uri=devices)
e8:  !Response (response=ok, data=[2])
e9:  ?/hardware (Verb=GET, Uri=hardware)
e10: !Response (response=ok, data=[3])
e11: ?/config (Verb=GET, Uri=config)
e12: !/json.htm (param=switchlight, switchcmd=on)
e13: ?Response (response=ok)
e14: !Response (response=ok, data=[4])
e15: ?/tools (Verb=GET, Uri=tools)
e16: !Response (response=ok, data=[5])
```

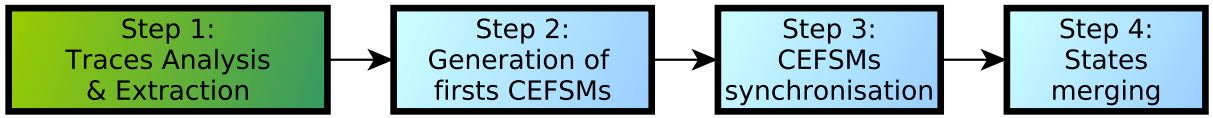
Traces Analysis & Extraction



- Extract of the red events $\rightarrow$ new trace sets
- Red events replaced by event call.

(
In reality not so simple :
Algorithm based on several definitions for finding red blocks with correlation factors + detection of hierarchical calls.
)

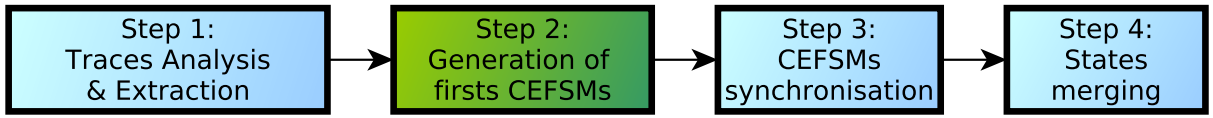
Traces Analysis & Extraction

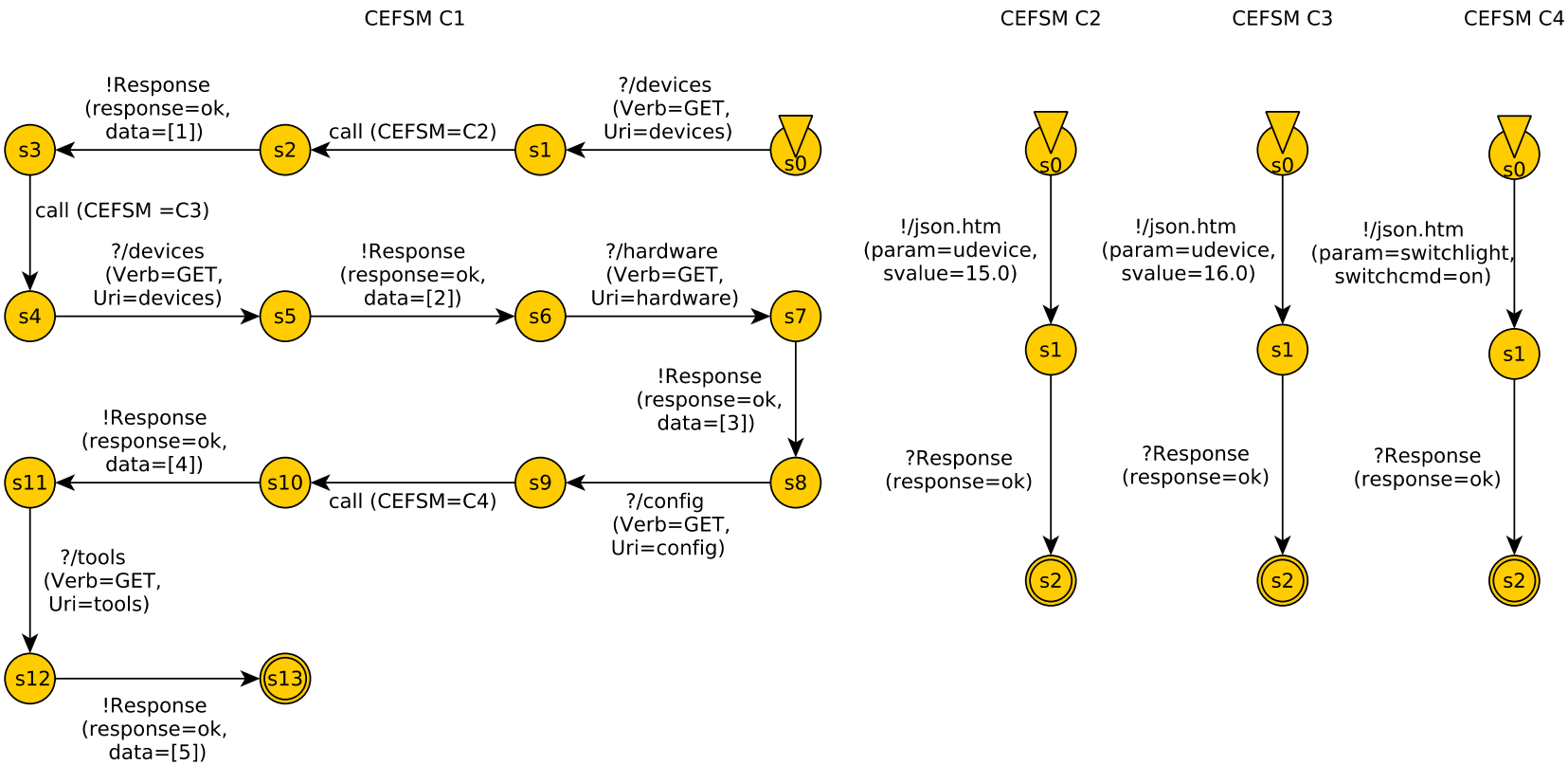


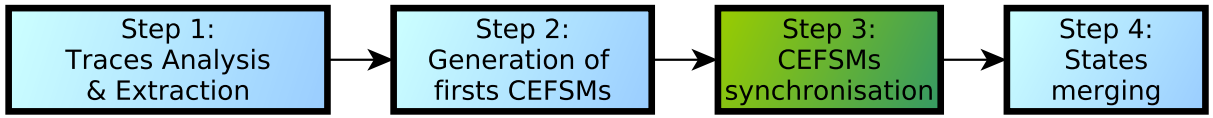

```
Trace T1:
e1:  ?/devices (Verb=GET, Uri=devices)
     call (CEFSM:=C2)
e4:  !Response (response=ok, data=[1])
     call (CEFSM:=C3)
e7:  ?/devices (Verb=GET, Uri=devices)
e8:  !Response (response=ok, data=[2])
e9:  ?/hardware (Verb=GET, Uri=hardware)
e10: !Response (response=ok, data=[3])
e11: ?/config (Verb=GET, Uri=config)
     call (CEFSM:=C4)
e14: !Response (response=ok, data=[4])
e15: ?/tools (Verb=GET, Uri=tools)
e16: !Response (response=ok, data=[5])
```

```
Trace T2:
e2:  !/json.htm (param=udevice, 15<=svalue<=16)
e3:  ?Response (response=ok)

Trace T3:
e5:  !/json.htm (param=udevice, 15<=svalue<=16)
e6:  ?Response (response=ok)

Trace T4:
e12: !/json.htm (param=switchlight, switchcmd=on)
e13: ?Response (response=ok)
```

Generation of CEFSMs

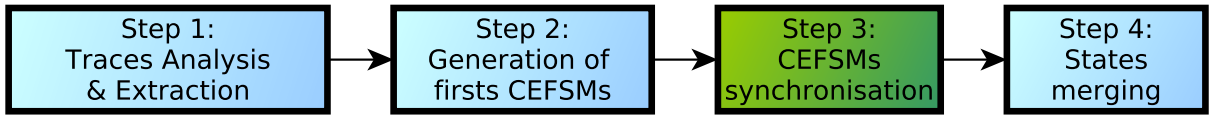




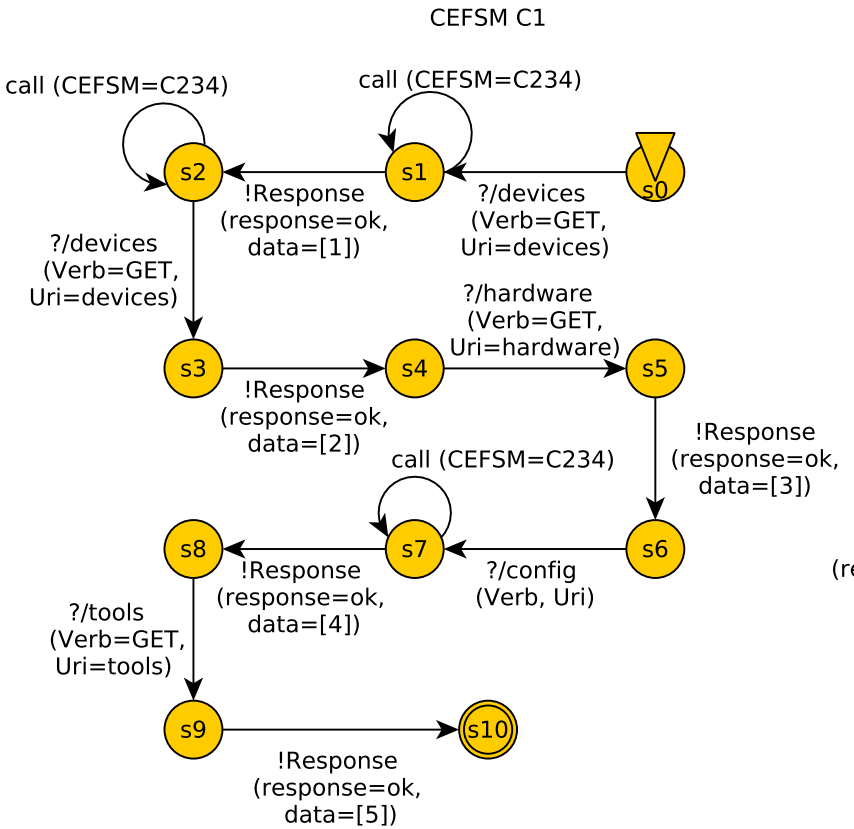


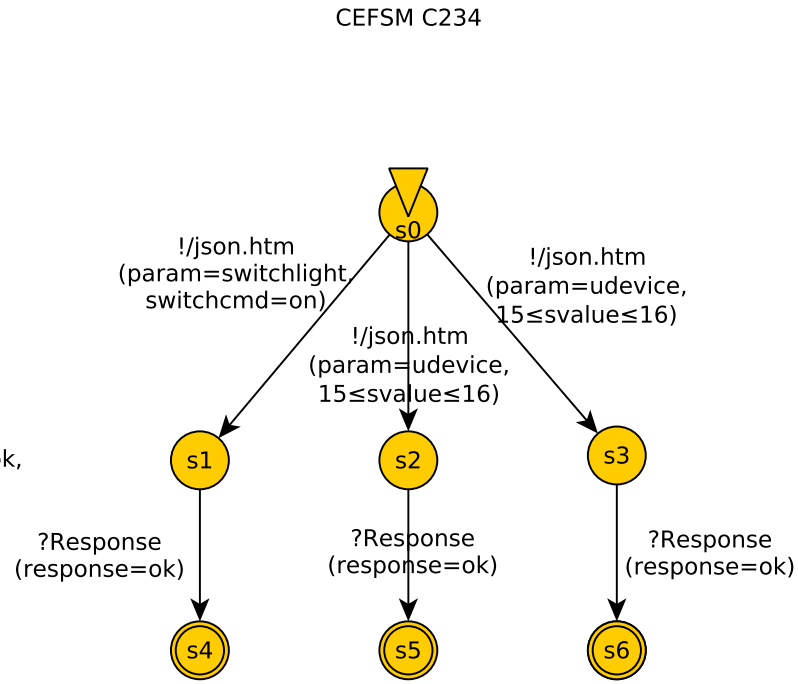
Synchronisation strategies:

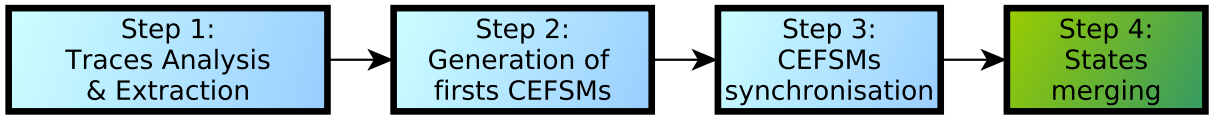
- **Strict Synchronisation:** we do not group CEFSMs, each CEFSM can be called once.
- **Weak Synchronisation:** we group similar CEFSMs, each CEFSM can be called many times.
- **Strong Synchronisation:** we group similar CEFSMs, each CEFSM can call any CEFSM from any state.



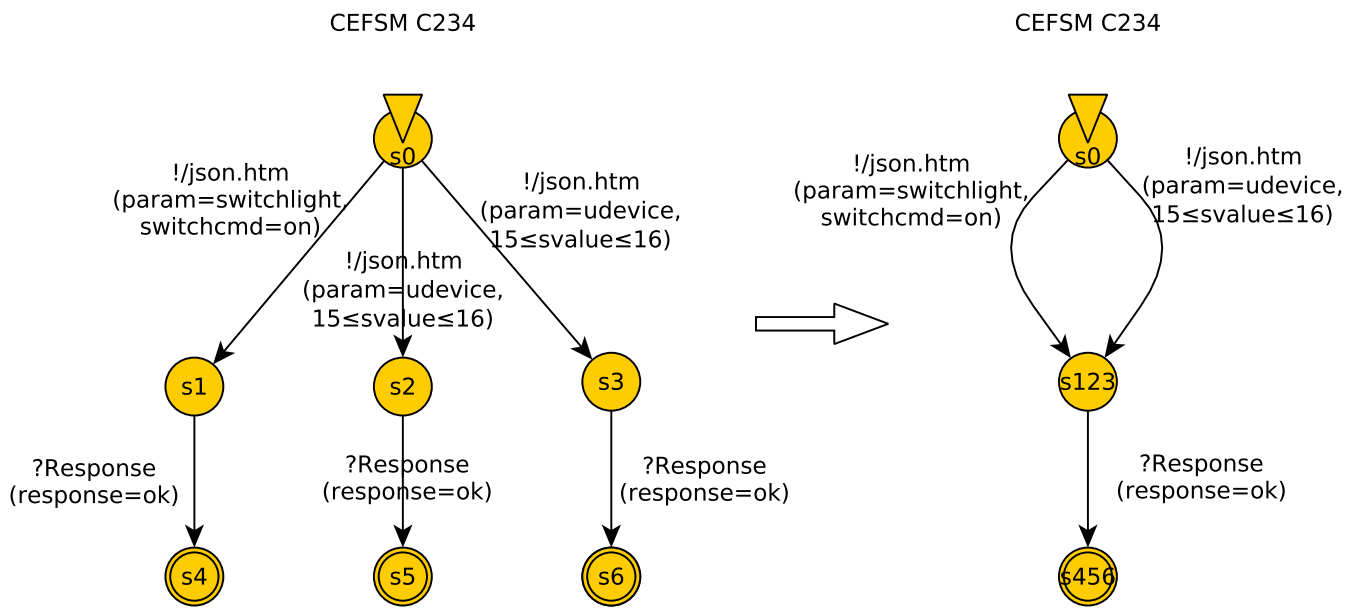
Example of CEFSMs obtained after Weak synchronisation :





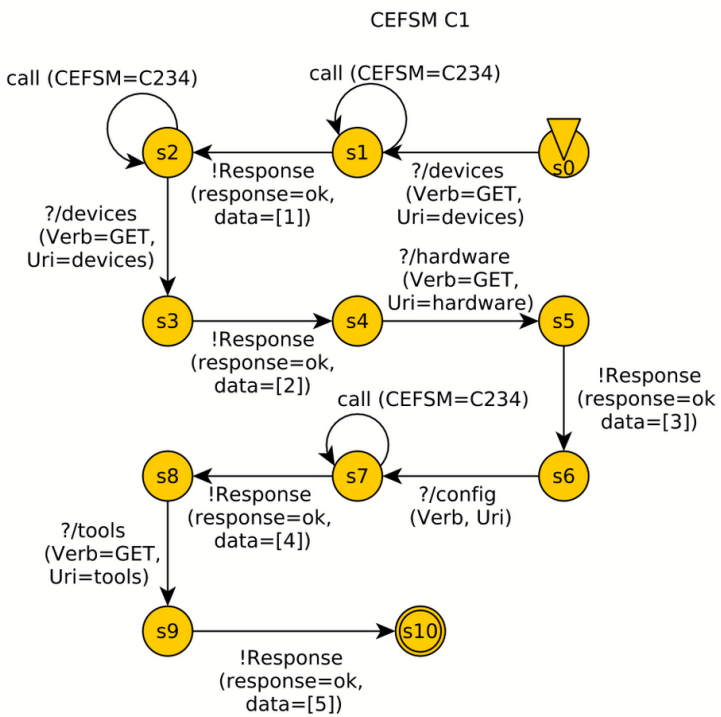


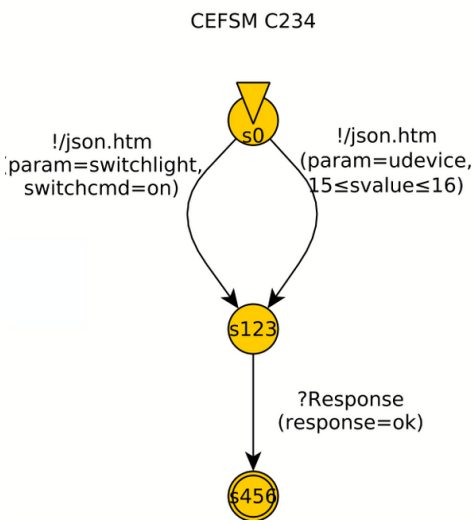
Merge equivalents states, with kTail, that merges states with the same future of length k or lower.



```
?/devices (Verb=GET, Uri=devices) !/json.htm (param=udevice, 15<=svalue<=16)
?Response (response=ok)!Response (response=ok, data=[1])
!/json.htm (param=udevice, 15<=svalue<=16) ?Response (response=ok)
?/devices (Verb=GET, Uri=devices) !Response (response=ok, data=[2])
?/hardware (Verb=GET, Uri=hardware) !Response (response=ok, data=[3])
?/config (Verb=GET, Uri=config)!/json.htm (param=switchlight, switchcmd=on)
?Response (response=ok) !Response (response=ok, data=[4])
?/tools (Verb=GET, Uri=tools) !Response (response=ok, data=[5])
```

2 models, which over-generalise the initial trace here





COnfECt approach

Generation of systems of CEFSMs from execution traces
Component behaviours modelling (hierarchical component calls)

Future work

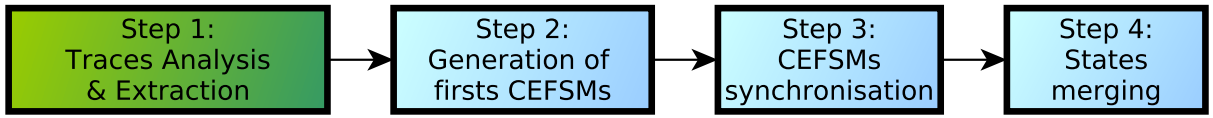
- Method evaluation on real cases.
- Using models for security testing.

Thank you
Questions?

A Callable Extended Finite State Machine (CEFSM) is a 5-tuple $\langle S, s0, \Sigma, P, T \rangle$ where :

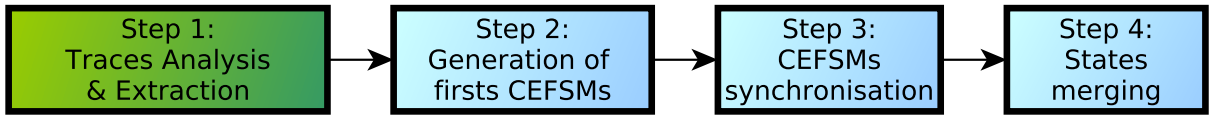
- S is a finite set of states, $S_F \subseteq S$ is the non-empty set of final states, $s0$ is the initial state,
- $\Sigma \subseteq \mathcal{E} = \Sigma_I \cup \Sigma_O \cup \{call\}$ is the finite set of symbols, with Σ_I the set of input symbols, Σ_O the set of output symbols and $call$ an internal action,
- P is a finite set of parameters, which can be assigned to values of D_P,
- T is a finite set of transitions. A transition $(s_1, e(p), G, s_2)$ is a 4-tuple also denoted $s_1 \xrightarrow{e(p),G} s_2$ where :
  - $s_1, s_2 \in S$ are the source and destination states,
  - $e(p)$ is an event with $e \in \Sigma$ and $p = \langle p_1, ..., p_k \rangle$ a finite tuple of parameters in $P^k (k \in \mathbb{N})$,
  - $G : D_P \rightarrow \{true, false\}$ is a guard that restricts the firing of the transition.

Traces Analysis & Extraction



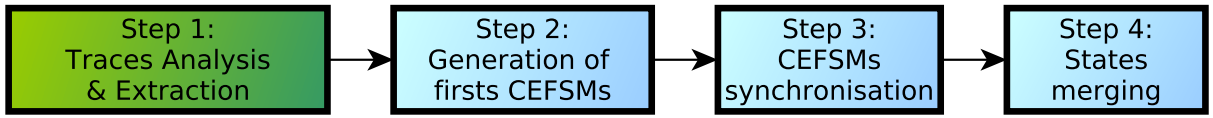
$$\sigma = \quad \sigma 1 \quad \sigma 2 \quad \sigma 3 \quad \sigma 4 \quad \sigma 5 \quad \sigma 6$$

Traces Analysis & Extraction



$\sigma = \quad \sigma 1 \quad \sigma 2 \quad \sigma 3 \quad \sigma 4 \quad \sigma 5 \quad \sigma 6$

Traces Analysis & Extraction

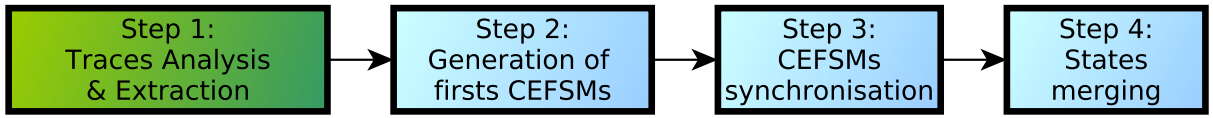


$$\sigma = \quad \sigma 1 \; call(CEFSM := C2) \; \sigma 5 \quad \sigma 6$$

$$\downarrow$$

$$T2 = \quad \sigma 2 \quad \sigma 3 \quad \sigma 4$$

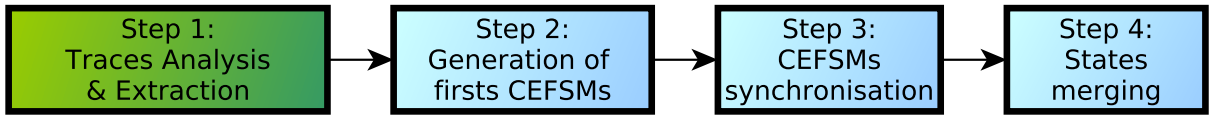
Traces Analysis & Extraction



$$\sigma = \quad \sigma 1 \; call(CEFSM := C2) \; \sigma 5 \quad \sigma 6$$

$$T2 = \quad \sigma 2 \qquad \sigma 3 \qquad \sigma 4$$

Traces Analysis & Extraction

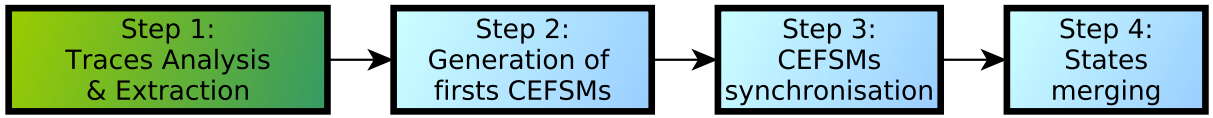


$$\sigma = \quad \sigma 1 \; call(CEFSM := C2) \; \sigma 5 \quad \sigma 6$$

$$T2 = \quad \sigma 2 \; call(CEFSM := C3) \; \sigma 4$$

$$T3 = \quad \sigma 3$$

Traces Analysis & Extraction

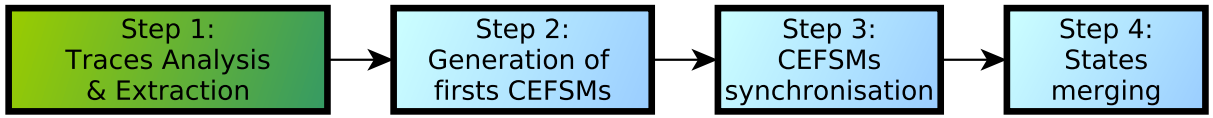


$$\sigma = \quad \sigma 1 \; call(CEFSM := C2) \; \sigma 5 \quad \sigma 6$$

$$T2 = \quad \sigma 2 \; call(CEFSM := C3) \; \sigma 4$$

$$T3 = \quad \sigma 3$$

Traces Analysis & Extraction



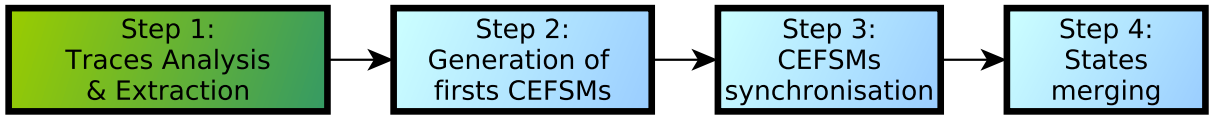
$\sigma = \quad \sigma 1 \; call(CEFSM := C2) \; \sigma 5 \; call(CEFSM := C4)$

$T2 = \quad \sigma 2 \; call(CEFSM := C3) \; \sigma 4$

$T3 = \quad \sigma 3$

$T4 = \quad \sigma 6$

Traces Analysis & Extraction



$$T1 = \quad \sigma 1 \ \mathit{call}(\mathit{CEFSM} := C2) \ \sigma 5 \ \mathit{call}(\mathit{CEFSM} := C4)$$

$$T2 = \quad \sigma 2 \ \mathit{call}(\mathit{CEFSM} := C3) \ \sigma 4$$

$$T3 = \quad \sigma 3$$

$$T4 = \quad \sigma 6$$

Algorithm 1: Procedures Inspect, Extract and Separate

1 **Procedure** *Inspect*(σ) : $\sigma'_1\sigma'_2\ldots\sigma'_k$ **is**
2 Find the no-empty sequences $\sigma'_1\sigma'_2\ldots\sigma'_k$ such that: $\sigma = \sigma'_1\sigma'_2\ldots\sigma'_k$, strong-corr$(\sigma'_i)_{(1\leq i\leq k)}$, $(\sigma'_i$ mismatch $\sigma'_{i+1})_{(1\leq i\leq k-1)}$;
3 **Procedure** *Extract*($\sigma = \sigma_1\sigma_2\ldots\sigma_k, T, STraces$): *STraces* **is**
4 $id := 1$;
5 **while** $id < k$ **do**
6 $n := |STraces| + 1$;
7 $T_n := \{\}$;
8 $STraces := STraces \cup \{T_n\}$;
9 σ_p is the prefix of σ up to σ_{id};
10 **if** $\exists i > id$: *strong-corr*$(\sigma_{id}\sigma_i)$ **then**
11 σ_i is the first sequence in $\sigma_{id}\ldots\sigma_k$ such that strong-corr$(\sigma_{id}\sigma_i)$;
12 $\sigma := \sigma_p\sigma_{id}call(CEFSM := C_n)\sigma_i\ldots\sigma_k$;
13 **if** $(i - id) > 2$ **then**
14 Extract$(\sigma_{id+1}\ldots\sigma_{i-1}, T_n)$;
15 **else**
16 $T_n := T_n \cup \{\sigma_{id+1}\}$;
17 $id := i$;
18 **else**
19 $\sigma := \sigma_p\sigma_{id}call(CEFSM := C_n)$;
20 **if** $(k - id) > 1$ **then**
21 Extract$(\sigma_{id+1}\ldots\sigma_k, T_n)$;
22 **else**
23 $T_n := T_n \cup \{\sigma_k\}$;
24 $id := k$;
25 $T := T \cup \{\sigma\}$;
26 return *STraces*;
27 **Procedure** *Separate*(T, *STraces*): *STraces* **is**
28 $\forall(\sigma_i, \sigma_j) \in T^2$ Compute $Similarity_{Trace}(\sigma_i, \sigma_j)$;
29 Build a similarity matrix;
30 Group the similar traces into clusters $\{T_{11}, \ldots T_{1k}\}$;
31 $STraces = STraces \setminus \{T_1\} \cup \{T^S_{11}, \ldots, T^S_{1k}\}$;

Algorithm 2: CEFSM synchronisation strategies

input : System of CEFSMs $SC = \langle C, S0 \rangle$, strategy
output: System of CEFSMs $SC_f = \langle C_f, S0_f \rangle$

1 **if** *strategy* $=$ *Strict synchronisation* **then**
2 return SC;
3 **else**
4 $\forall (C_i, C_j) \in C^2$ Compute $Similarity_{CEFSM}(C_i, C_j)$;
5 Build a similarity matrix;
6 Group the similar CEFSMs into clusters $\{Cl_1, \ldots Cl_k\}$;
7 **foreach** *cluster* $Cl = \{C_1, \ldots, C_l\}$ **do**
8 C_{Cl}:=Disjoint Union of the CEFSMs $C_1, \ldots, C_l$;
9 **if** $s0_i \in S0 (1 \leq i \leq l)$ **then**
10 $S0_f = S0_f \cup s0_{Cl}$;
11 $C_f = C_f \cup \{C_{Cl}\}$;
12 **foreach** $C_i = \langle S, s0, \Sigma, P, V, T \rangle \in C_f$ **do**
13 **foreach** $s_1 \xrightarrow{call(CEFSM), G} s_2 \in T$ *with* $G : CEFSM = C_m$ **do**
14 Find the Cluster Cl such that $C_m \in Cl$;
15 Replace G by $G : CEFSM = C_{Cl}$;
16 Merge (s_1, s_2);
17 **if** *strategy* $=$ *Strong synchronisation* **then**
18 **foreach** $C_i = \langle S, s0, \Sigma, P, T \rangle \in C_f$ **do**
19 Complete the states of S with self-loop transitions so that C_i is callable-complete;
20 return SC_f

Merge equivalents states, with the same future of length k.

2-futur equivalents Example: