UNIVERSITÀ DEGLI STUDI DI UDINE
hic sunt futura

Modeling Collective Adaptive Systems with Attribute-Based Events: Recent Trends and Open Problems

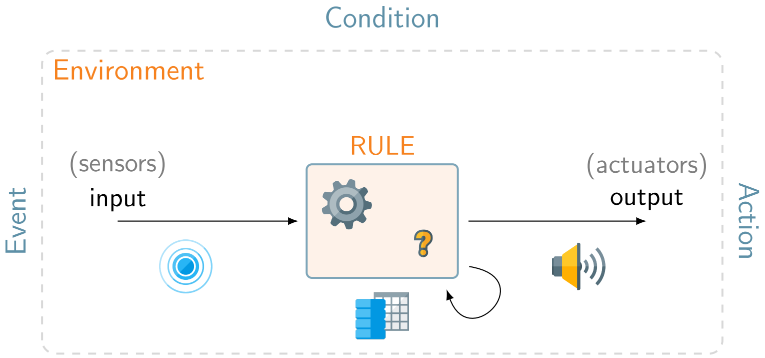
based on joint work with M. Pasqua (U. Verona), M. Paier (IMT Lucca), and others

Marino Miculan
marino.miculan@uniud.it

June 29, 2023

OPCT 2023 - Bertinoro

Event-driven programming of smart systems

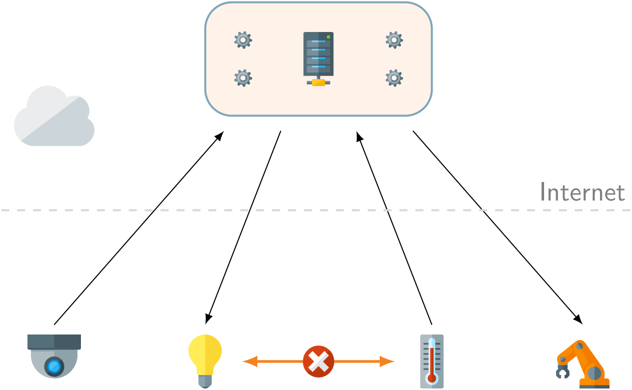


State-based ECA rules: "**on** movement **if** alarm = "active" **then** siren ← on"

variables can be internal, or connected to sensors or to actuators

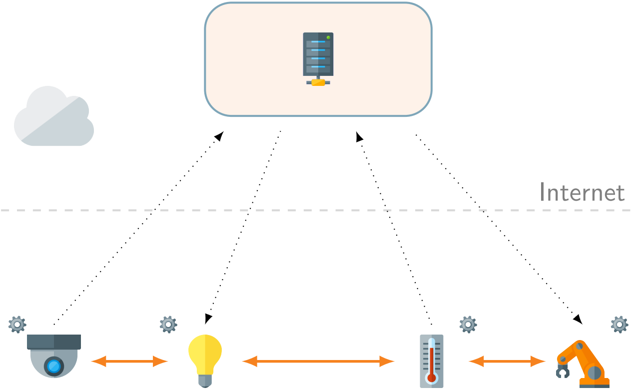
Common IoT architecture

- Centralized
- No intra-nodes communication
- Cloud and Internet-dependent
- Very popular:



Next (ECA) IoT architecture: *edge computing*

- Fully distributed
- Communication between nodes
- Cloud-agnostic
- Identity decoupled, for scalability
- *Collective Adaptive Systems*



Programming model for edge CAS?

We need programming abstractions and models for edge computing with:

- peer-to-peer, decentralised control
- identity decoupling, for scalability (no point-to-point communication)
- open and flexible (nodes can join and leave dynamically)
- which integrate neatly within the ECA paradigm

Programming model for edge CAS?

Alrahman et al. (2015): *attribute-based communication*, a new form of broadcast for coordinating large numbers of components: the actual receivers are selected "on the fly" by means of predicates.

Proposed the *AbC calculus*, which has two communication actions:

- $(E)@\Pi.P$: send the values of E to those components whose attributes satisfy Π;
- $\Pi(x).P$: receive from any component whose attributes (and possibly transmitted values) satisfy Π.

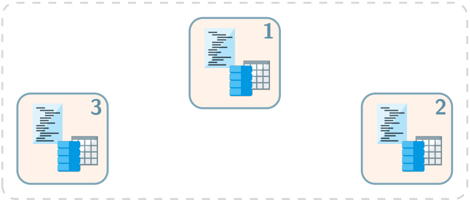
But message-passing is not proper of state-based declarative ECA programming: Interaction is on shared memory and modified variables.

Attribute-based Memory Updates

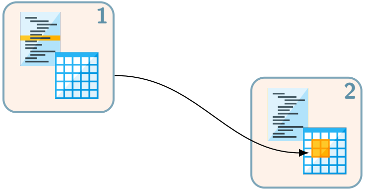
[M., Pasqua, ICTAC 2021]

Nodes behavior: defined by ECA rules like "on z for all Π : $x \leftarrow e$"

Nodes state: local memory



Interaction: remote updates



Attribute-based interaction: on all nodes satisfying Π, update the remote x with e

AbC-AbU correspondence

	AbC	AbU
Communication	message-passing	memory updates
Output	$(E)@\Pi$	$@\ \Pi$
Input	$\Pi(x)$	nodes invariant

- In AbU there are no explicit input primitives, to filter incoming updates
- But we can specify *admissible* states by means of state invariants

The AbU language

- An **AbU system** S is an **AbU node** $R, \iota\langle\Sigma, \Theta\rangle$ or the parallel of systems $\mathsf{S}_1 \,\|\, \mathsf{S}_2$
- Each node is equipped with a list R of **AbU rules** and an **invariant** ι

event default task forall @ $\overline{x} \leftarrow \varepsilon$ (remote)

$\underbrace{\mathsf{evt}}_{\text{list of resources}} \gtrdot \mathsf{act}_1 \;,\; \overbrace{\mathsf{cnd} : \mathsf{act}_2}^{\text{task}}$ assignments

$x \leftarrow \varepsilon$ (local)

“on all nodes with (remote) x greater than the current (local) x”

for all: $@(x < \bar{x}) : \bar{x} \leftarrow x, \bar{y} \leftarrow \bar{y} + 1$

“assign the (remote) x with the current (local) x, and increment remote y”

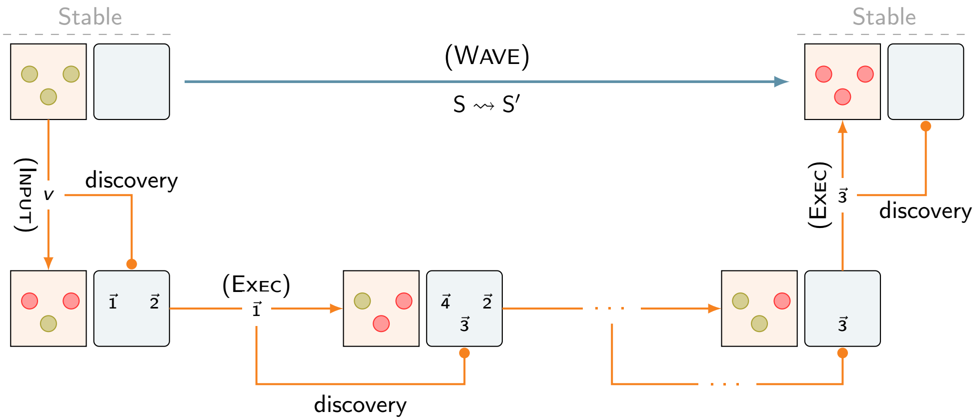
The AbU Language: a Domain Specific Language for the IoT

```
# AbU devices definition.

hvac : "An HVAC control system" {
   physical output boolean heating = fals
   physical output boolean condit = false
   logical integer temp = 0
   logical integer humidity = 0
   physical input boolean airButton
   logical string node = "hvac"
   where not (condit and heating == true)
} has cool warm dry stopAir

tempSens : "A temperature sensor" {
   physical input integer temp
   logical string node = "tempSens"
} has notifyTemp

humSens : "A humidity sensor" {
   physical input integer humidity
   logical string node = "humSens"
} has notifyHum
```

```
\%
AbU (ECA) rules definition.
Rules can be referenced by multiple devices.
%\

rule cool on temp
for (this.temp < 18) do this.heating = true

rule warm on temp
for (this.temp > 27) do this.heating = false

rule dry on humidity; temp
for (this.temp * 0.14 < this.humidity)
do this.condit = true

rule stopAir on airButton
for (this.airButton) do this.condit = false

rule notifyTemp on temp
for all (ext.node == "hvac")
do ext.temp = this.temp
```

See paper on IEEE Access 2022

AbU execution model



AbU operational semantics

[Pasqua, M., TCS 2023]

LTS semantics, with judgments:

$$R, \iota\langle \Sigma, \Theta \rangle \xrightarrow{\alpha} R, \iota\langle \Sigma', \Theta' \rangle$$

A label α can be:

- an **input** label, upd $\blacktriangleright$ T
- an **execution** label, upd $\triangleright$ T
- a **discovery** label, T

AbU operational semantics: rules

$$
(\textsc{Exec})\ \frac{\begin{array}{c}\mathsf{upd} \in \Theta \quad \mathsf{upd} = (x_1, v_1) \dots (x_k, v_k) \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad \Sigma' \models \iota \\ \Theta'' = \Theta \setminus \{\mathsf{upd}\} \quad X = \{x_i \mid i \in [1..k] \land \Sigma(x_i) \neq \Sigma'(x_i)\} \\ \Theta' = \Theta'' \cup \mathsf{DefUpds}(R, X, \Sigma') \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle \Sigma, \Theta \rangle \xrightarrow{\mathsf{upd} \rhd T} R, \iota\langle \Sigma', \Theta' \rangle}
$$

$$
(\textsc{Exec-Fail})\ \frac{\mathsf{upd} \in \Theta \quad \mathsf{upd} = (x_1, v_1) \dots (x_k, v_k) \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad \Sigma' \not\models \iota \quad \Theta' = \Theta \setminus \{\mathsf{upd}\}}{R, \iota\langle \Sigma, \Theta \rangle \xrightarrow{\mathsf{upd} \rhd T} R, \iota\langle \Sigma, \Theta' \rangle}
$$

$$
(\textsc{Input})\ \frac{\begin{array}{c}v_1, \dots, v_k \in \mathbb{V} \quad \Sigma' = \Sigma[v_1/x_1 \dots v_k/x_k] \quad X = \{x_1, \dots, x_k\} \\ \Theta' = \Theta \cup \mathsf{DefUpds}(R, X, \Sigma') \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle \Sigma, \Theta \rangle \xrightarrow{(x_1, v_1) \dots (x_k, v_k) \blacktriangleright T} R, \iota\langle \Sigma', \Theta' \rangle}
$$

$$
(\textsc{Disc})\ \frac{\Theta'' = \{[\![\mathsf{act}]\!]\Sigma \mid \exists i \in [1..n] \,.\, \mathsf{task}_i = \varphi : \mathsf{act} \land \Sigma \models \varphi\} \quad \Theta' = \Theta \cup \Theta''}{R, \iota\langle \Sigma, \Theta \rangle \xrightarrow{\mathsf{task}_1 \dots \mathsf{task}_n} R, \iota\langle \Sigma, \Theta' \rangle}
$$

$$
(\textsc{StepL})\ \frac{\mathsf{S}_1 \xrightarrow{\alpha} \mathsf{S}_1' \quad \mathsf{S}_2 \xrightarrow{T} \mathsf{S}_2'}{\mathsf{S}_1 \parallel \mathsf{S}_2 \xrightarrow{\alpha} \mathsf{S}_1' \parallel \mathsf{S}_2'}\ \alpha \in \{\mathsf{upd} \rhd T, \mathsf{upd} \blacktriangleright T\}
\qquad
(\textsc{StepR})\ \frac{\mathsf{S}_1 \xrightarrow{T} \mathsf{S}_1' \quad \mathsf{S}_2 \xrightarrow{\alpha} \mathsf{S}_2'}{\mathsf{S}_1 \parallel \mathsf{S}_2 \xrightarrow{\alpha} \mathsf{S}_1' \parallel \mathsf{S}_2'}\ \alpha \in \{\mathsf{upd} \rhd T, \mathsf{upd} \blacktriangleright T\}
$$

Research questions and problems

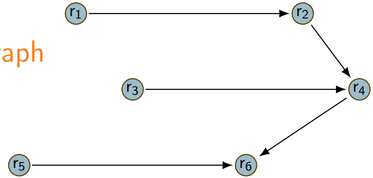
1. Stability: after an input, does a wave computation always terminates?
2. Confluence: will different executions end up with the same state(s)?
3. Global invariants: how to guarantee that trajectories will not invalidate a given global property?
4. Security: how to avoid information leakages?
5. Safety: how to avoid unintended interactions?
6. Implementation: how to make it efficient, portable and scalable?
7. ...

Q1: Stabilization

The wave semantics may exhibit internal divergence, namely $S \xrightarrow{\alpha_0} S^0 \xrightarrow{\alpha_1} \ldots$

Question: how to guarantee that a program will always stabilize, after an input?

ECA
dependency graph



rule A $r_4 \gtrdot (\Box) : r_6 \leftarrow \Box$

rule B $r_3\, r_2 \gtrdot (\Box) : r_4 \leftarrow \Box$

rule C $r_5 \gtrdot (\Box) : r_6 \leftarrow \Box$

rule D $r_1 \gtrdot (\Box) : r_2 \leftarrow \Box$

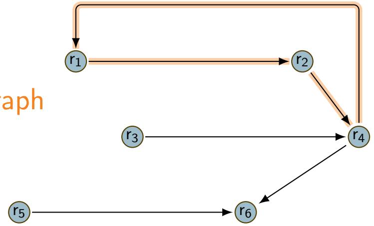
Theorem (AbU stabilization)

If the ECA dependency graph of an AbU system S is acyclic, then S is stabilizing.

Q1: Stabilization

The wave semantics may exhibit internal divergence, namely $S \xrightarrow{\alpha_0} S^0 \xrightarrow{\alpha_1} \dots$
Question: how to guarantee that a program will always stabilize, after an input?

ECA
dependency graph



rule A $r_4 \gtrdot (\square) : r_6 \leftarrow \square \; r_1 \leftarrow \square$

rule B $r_3 \; r_2 \gtrdot (\square) : r_4 \leftarrow \square$

rule C $r_5 \gtrdot (\square) : r_6 \leftarrow \square$

rule D $r_1 \gtrdot (\square) : r_2 \leftarrow \square$

Theorem (AbU stabilization)

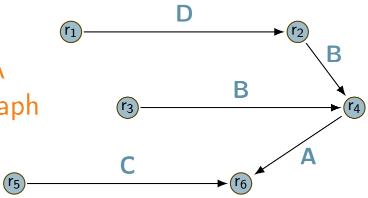
If the ECA dependency graph of an AbU system S is acyclic, then S is stabilizing.

Can we do better? E.g., including (some) loops? (Control theory may be useful here?)

Q2: Confluence

We may want the semantics not to be influenced by scheduler decisions:
for all S_1, S_2 s.t. $S \rightarrowtriangle^* S_1$ and $S \rightarrowtriangle^* S_2$, there exists S' s.t. $S_1 \rightarrowtriangle^* S'$ and $S_2 \rightarrowtriangle^* S'$

labeled ECA dependency graph



rule A $r_4 \gtrdot (\square) : r_6 \leftarrow \square$

rule B $r_3\ r_2 \gtrdot (\square) : r_4 \leftarrow \square$

rule C $r_5 \gtrdot (\square) : r_6 \leftarrow \square$

rule D $r_1 \gtrdot (\square) : r_2 \leftarrow \square$

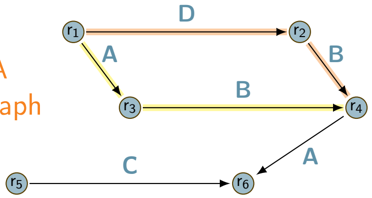
Theorem (AbU confluence)

If for each pair (x, y) of nodes in the labeled ECA dependency graph of an AbU system S we have that $|\text{walks}(x, y)| \leq 1$, then S is confluent.

Q2: Confluence

We may want the semantics not to be influenced by scheduler decisions:
for all S_1, S_2 s.t. $S \twoheadrightarrow^* S_1$ and $S \twoheadrightarrow^* S_2$, there exists S' s.t. $S_1 \twoheadrightarrow^* S'$ and $S_2 \twoheadrightarrow^* S'$

labeled ECA dependency graph



rule A $r_4 \gtrdot (\square) : r_6 \leftarrow \square\ r_3 \leftarrow \square$

rule B $r_3\ r_2 \gtrdot (\square) : r_4 \leftarrow \square$

rule C $r_5 \gtrdot (\square) : r_6 \leftarrow \square$

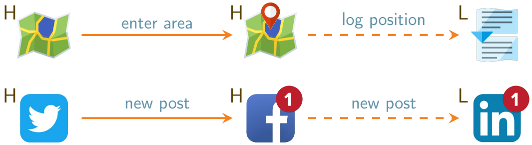
rule D $r_1 \gtrdot (\square) : r_2 \leftarrow \square$

Theorem (AbU confluence)

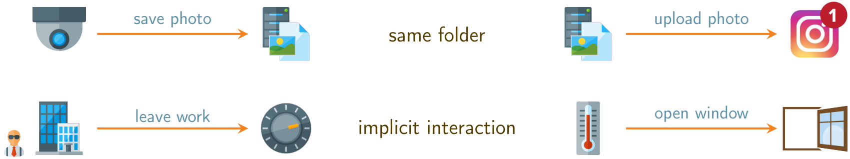
If for each pair (x, y) of nodes in the labeled ECA dependency graph of an AbU system S we have that $|\text{walks}(x, y)| \leq 1$, then S is confluent.

Q4: Security, and Q5: Safety [Pasqua, M., SEFM 2021]

Security



Safety



Hiding bisimulation

- Weak bisimulation hiding labels not related to the requirements
- Parametric on a function h making non-observable labels α such that $h(\alpha) = \diamond$

$$\text{if } h(\alpha_2) \neq \diamond \qquad \begin{array}{c} \downarrow \alpha_1 \\ \downarrow \alpha_2 \\ \downarrow \alpha_3 \end{array} \quad \approx_h \quad \begin{array}{c} \downarrow \alpha_1 \\ \vdots \\ \downarrow \alpha_2 \\ \vdots \\ \downarrow \alpha_3 \end{array}$$

$$\text{if } h(\alpha_2) = \diamond \qquad \begin{array}{c} \downarrow \alpha_1 \\ \downarrow \alpha_2 \\ \downarrow \alpha_3 \end{array} \quad \approx_h \quad \begin{array}{c} \downarrow \alpha_1 \\ \vdots \\ \downarrow \alpha_2' \\ \vdots \\ \downarrow \alpha_3 \end{array} \quad \text{with } h(\alpha_2') = \diamond$$

Security h_{L} hides:

- discovery labels
- execution labels with H resources

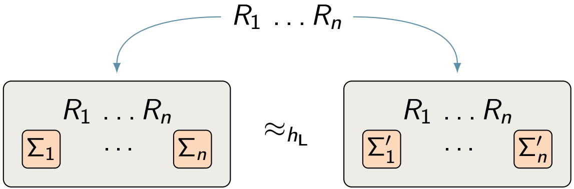
Safety h_{S} hides:

- discovery labels
- execution labels produced by S

Q4: Security: behavioral equivalence

Protection of confidential data (noninterference)

- Security policy: L (public) and H (confidential) resources
- No flows from H to L allowed
- Bisimulation $\approx_{h_L}$ that hides H-level updates
- $R_1 \ldots R_n$ is *interference-free* if it "behaves the same" for L-equivalent states



Hiding bisimulation:

execution labels with H resources

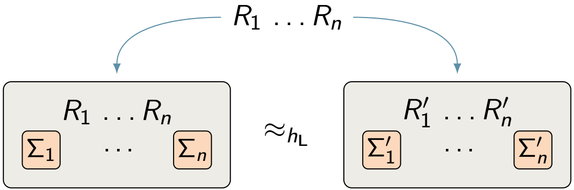
for all L-equivalent states $\Sigma_1 \equiv_L \Sigma'_1 \ldots \Sigma_n \equiv_L \Sigma'_n$

Q4: Security: behavioral equivalence

- This definition captures leaks due to internal resources modifications, but not leaks originated by external changes (i.e., inputs) on high-level variables. E.g.:

$$\mathit{motion} \gtrdot (00\!:\!00 < \mathit{time} \wedge \mathit{time} < 06\!:\!00) : \mathit{light} \leftarrow \text{`on'}$$

  (where *motion* is H and *light* is L) is interference-free as defined above, but it actually leaks confidential information.
- $R_1 \ldots R_n$ is *presence-sensitive interference-free* if it "behaves the same" for L-equivalent states *and under renaming of rule triggers of level H*

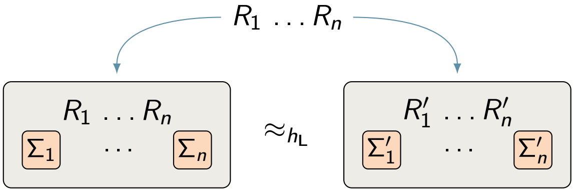


Q4: Security: behavioral equivalence

- This definition captures leaks due to internal resources modifications, but not leaks originated by external changes (i.e., inputs) on high-level variables. E.g.:

  $$motion \gtrdot (00\!:\!00 < time \wedge time < 06\!:\!00) : light \leftarrow \text{`on'}$$

  (where *motion* is H and *light* is L) is interference-free as defined above, but it actually leaks confidential information.
- $R_1 \dots R_n$ is *presence-sensitive interference-free* if it "behaves the same" for L-equivalent states *and under renaming of rule triggers of level H*



for all L-equivalent states $\Sigma_1 \equiv_L \Sigma'_1 \dots \Sigma_n \equiv_L \Sigma'_n$

Q4: Security: verification algorithm

Algorithm `IFRules` for computing information flows:

context is L

explicit

$x \leftarrow \varepsilon$

$x : \mathsf{L}$ $\qquad$ ε not constant

context is H

implicit

$x \leftarrow \varepsilon$

$x : \mathsf{L}$ $\qquad$ ε constant

- Compute a constancy analysis for conditions and expressions
- Check explicit flows for the default action
- Check explicit and implicit flows for the task action

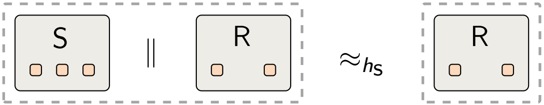
Theorem (Soundness for Security)

If $\texttt{IFRules}(R) = \textit{false}$ *then* R *is noninterferent, hence* R *is secure.*

Q5: Safety: behavioral equivalence

Prevention of unintended interactions

- The systems S and R are known to be safe
- Is the ensemble of all nodes in S and R still safe?
- Bisimulation $\approx_{hs}$ that hides the updates of S



Hiding bisimulation:

execution labels produced by S

S does not interact with, or is transparent for, R

Q5: Safety: verification algorithm

- Compute sinks: resources that rules may update
- Compute sources: resources that may influence rules behavior

Check that the sinks of S does not overlap with the sources of R

$$\overbrace{\{y_1,\ldots,y_n\}}^{\text{LHS}} \cup \overbrace{\{y_{n+1},\ldots,y_{n+m}\}}^{\text{LHS}}$$

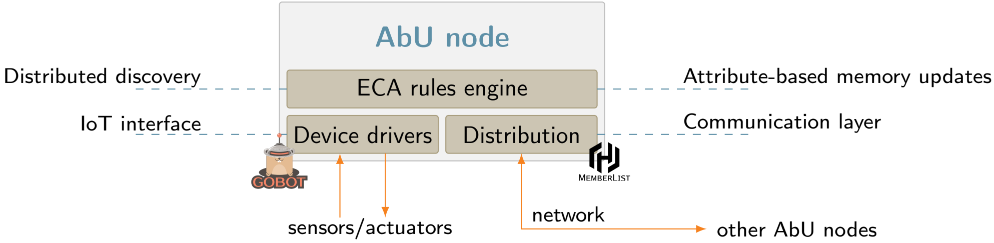
$$\overset{\text{event}}{\boxed{x_1 \ldots x_k}} > \overset{\text{default}}{\boxed{y_1 \leftarrow \varepsilon_1 \ldots y_n \leftarrow \varepsilon_n}} , \overset{\text{task}}{\boxed{(\text{cnd}) : y_{n+1} \leftarrow \varepsilon_{n+1} \ldots y_{n+m} \leftarrow \varepsilon_{n+m}}}$$

$$\{x_1,\ldots,x_k\} \cup \underbrace{\text{Vars}(\varepsilon_1) \cup \ldots \cup \text{Vars}(\varepsilon_n)}_{\text{RHS}} \cup \text{Vars}(\text{cnd}) \cup \underbrace{\text{Vars}(\varepsilon_{n+1}) \cup \ldots \cup \text{Vars}(\varepsilon_{n+m})}_{\text{RHS}}$$

Theorem (Soundness for Safety)

If $\text{sinks}(S) \cap \text{sources}(R) = \varnothing$ *then* S *is transparent for* R.

Q5: A (modular) distributed implementation



- ECA rules engine module: AbU semantics
- Device drivers module: abstraction of physical resources
- Distribution module: abstraction of send/receive and cluster nodes join/leave
- Available at `https://github.com/abu-lang`

Conclusion and Future Work

AbU: a new ECA programming paradigm for smart devices

Open Problems:

1. Stability: after an input, does a wave computation always terminates?
2. Confluence: will different executions end up with the same state(s)?
3. Global invariants: how to guarantee that trajectories will not invalidate a given global property?
4. Security: how to avoid information leakage?
5. Safety: how to avoid unintended interactions?
6. Implementation: how to make it efficient, portable and scalable?
7. Abstract model: what is the *bedrock* of decentralised event-driven programming?

Thanks for the attention

- M Miculan, M Pasqua, *A Calculus for Attribute-based Memory Updates*, Proc. ICTAC 2021 - LNCS 12819;
- M Pasqua, M Miculan, *On the Security and Safety of AbU Systems*, International Conference on Software Engineering and Formal Methods, LNCS 13085, 2021.
- M Pasqua, M Miculan, *Distributed Programming of Smart Systems with Event-Condition-Action Rules*, ICTCS 2022: 201-206
- M Pasqua, M Comuzzo, M Miculan, *The AbU Language: IoT Distributed Programming Made Easy*, IEEE Access 10: 132763-132776 (2022)
- M Pasqua, M Miculan, *AbU: A calculus for distributed event-driven programming with attribute-based interaction*. TCS 958: 113841 (2023)
- `https://github.com/abu-lang`