UNIVERSITÀ DEGLI STUDI DI UDINE

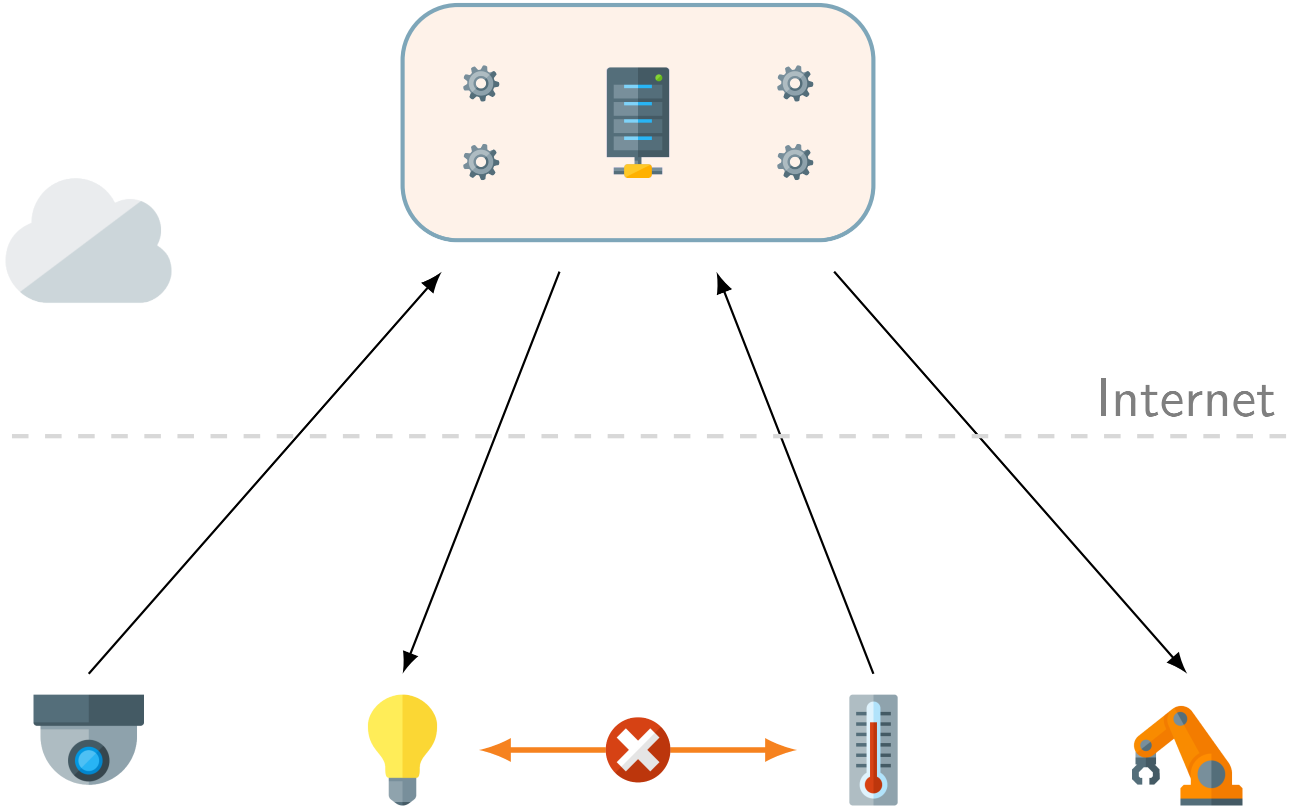
Enforcing Global Invariants with Local Reasoning in AbU Collective Adaptive Systems

Marino Miculan (75% UniUD, 25% UniVE)
(Joint work with M. Pasqua, UniVR)

SWOPS 2 — IMT Alti Studi, Lucca — June 13, 2024

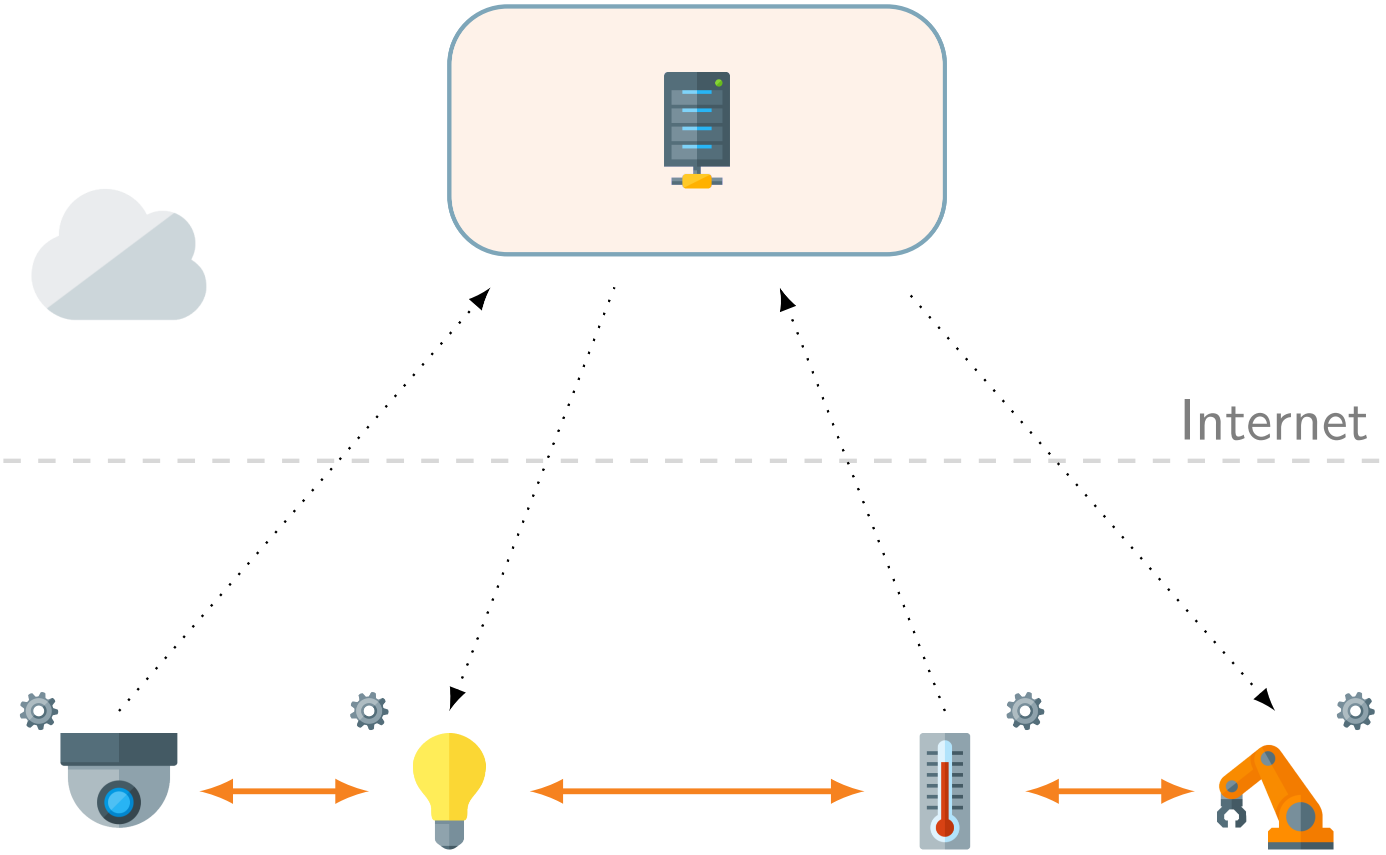
Actual IoT/*smart* architecture

- Centralized
- No inter-nodes communication
- Cloud-dependent
- Very popular as *Trigger-Action Platforms* (TAP)
    - Google Home
    - IFTTT
    - Samsung SmartThings
    - …



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?

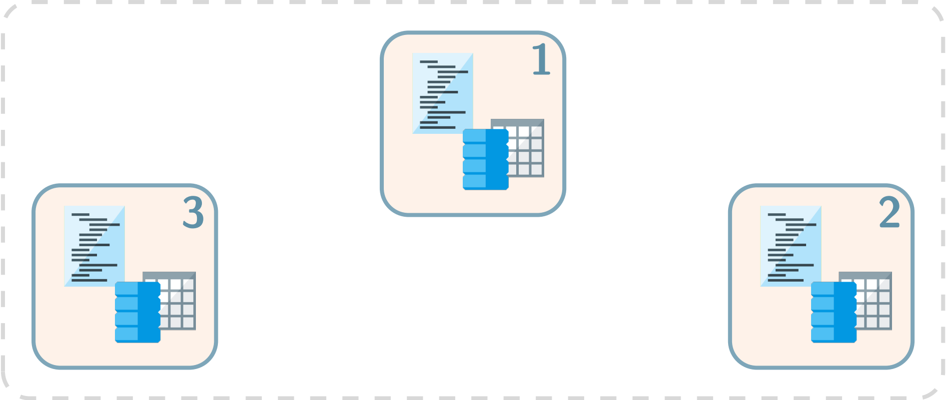
- We need programming abstractions and models for edge computing with:
  - peer-to-peer, decentralised control
  - identity decoupling, for scalability (no point-to-point communication)
  - open and flexible (nodes can join and leave dynamically)
  - which integrate neatly within the ECA paradigm
- Our proposal:
  Attribute-based distributed declarative programming
  (rooted in Attribute-based Communication)

Attribute-based Memory Updates

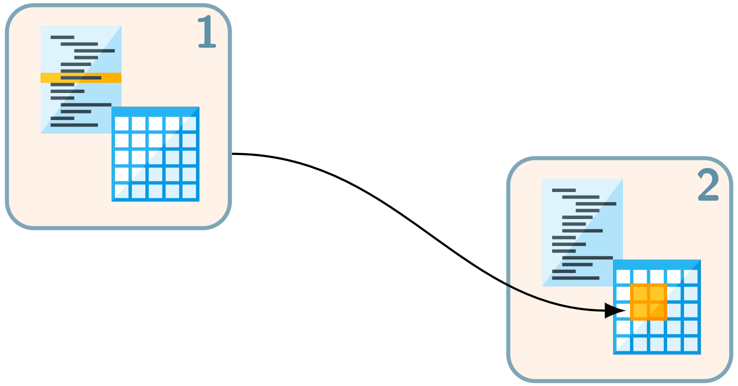
- Nodes behavior: defined by ECA rules like

$$\text{“}\mathbf{on}\ z_1 \ldots z_m\ \mathbf{for\ all}\ \Pi : x_1 \leftarrow e_1 \ldots x_n \leftarrow e_n\text{”}$$

Nodes state: local memory



Interaction: remote updates



- **Attribute-based interaction**: on all nodes satisfying Π, update the remote $x_1, \ldots, x_n$ with the values of $e_1, \ldots, e_n$

AbU syntax

- AbU systems: $S ::= R, \iota\langle\Sigma, \Theta\rangle \mid S_1 \parallel S_2$
  - R = list of AbU rules
  - Σ = state of the node (local variables, attributes)
  - ι = invariant over local variables (specifies *admissible* states)
  - Θ = set of pending updates
- Form of the rules:

event default task forall @

$$\mathsf{evt} \;\gtrdot\; \mathsf{act}_1 \;,\; \mathsf{cnd} : \mathsf{act}_2$$

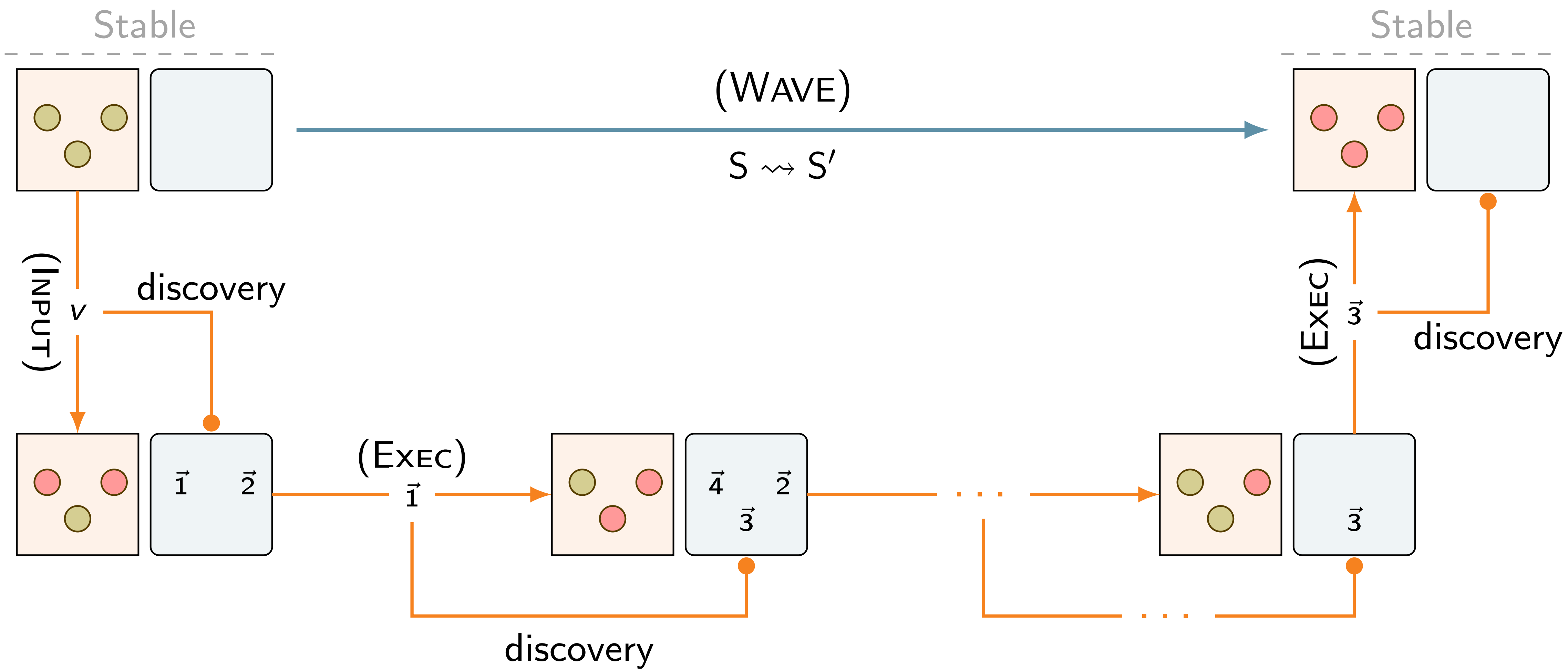
list of resources

assignments

$\overline{x} \leftarrow \varepsilon$ (remote)

$x \leftarrow \varepsilon$ (local)

AbU execution model



AbU operational semantics

- LTS semantics, with judgments of the form

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

- Labels:
  - Input (from devices): $upd \blacktriangleright T$
  - Internal execution: $upd \triangleright T$
  - “Discovery” (receive): T
  - (T is a list of updates generated by a rule’s firing)
- Interleaving semantics: communications are atomic transactions

AbU semantics: SOS rules

$$
(\textsc{Exec})\ \frac{\begin{array}{c}\mathsf{upd} \in \Theta \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \quad \Sigma' \models \iota \\ \Theta'' = \Theta \setminus \{\mathsf{upd}\} \quad X = \{\mathrm{x}_i \mid i \in [1..k] \wedge \Sigma(\mathrm{x}_i) \neq \Sigma'(\mathrm{x}_i)\} \\ \Theta' = \Theta'' \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle\Sigma, \Theta\rangle \xrightarrow{\mathsf{upd} \triangleright T} R, \iota\langle\Sigma', \Theta'\rangle}
$$

$$
(\textsc{Exec-F})\ \frac{\mathsf{upd} \in \Theta \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \quad \Sigma' \not\models \iota \quad \Theta' = \Theta \setminus \{\mathsf{upd}\}}{R, \iota\langle\Sigma, \Theta\rangle \xrightarrow{\mathsf{upd} \triangleright \epsilon} R, \iota\langle\Sigma, \Theta'\rangle}
$$

$$
(\textsc{Input})\ \frac{\begin{array}{c}v_1, \dots, v_k \in \mathbb{V} \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \quad X = \{\mathrm{x}_1, \dots, \mathrm{x}_k\} \\ \Theta' = \Theta \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle\Sigma, \Theta\rangle \xrightarrow{(\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \blacktriangleright T} R, \iota\langle\Sigma', \Theta'\rangle}
$$

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

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

(Some) Research questions and problems

- **Stability**: after an input, does a wave computation always terminate?

[ICTAC 2021, TCS 2023]

- **Confluence**: will different executions end up with the same state(s)?
- **Security**: how to avoid information leakages?

[SEFM 2021, TCS 2024]

- **Safety**: how to avoid unintended interactions?
- **Implementation**: how to make it efficient, portable and scalable?

[IEEE ACCESS 2023]

- **Global invariants**: how to guarantee that executions will not invalidate a given global property?

This talk!

(None of these problems is definitely solved. Still a lot to do!)

Example: smart HVAC system

$$R_s, \iota_s\langle \Sigma_s, \varnothing\rangle \parallel R_t\langle \Sigma_t, \varnothing\rangle \parallel R_h\langle \Sigma_h, \varnothing\rangle$$

- Three kinds of devices: ‘system’, ‘tempSens’, ‘humSens’
- Control system’s state:

$$\Sigma_s = [\,\mathrm{heating} \mapsto \mathtt{ff} \;\; \mathrm{conditioning} \mapsto \mathtt{ff} \;\; \mathrm{temperature} \mapsto 0 \\ \mathrm{humidity} \mapsto 0 \;\; \mathrm{airButton} \mapsto \mathtt{ff} \;\; \mathrm{node} \mapsto \text{‘}\mathit{system}\text{’}]$$

- ...and rules:

$$\mathrm{temperature} \gtrdot (\mathrm{temperature} < 18) : \mathrm{heating} \leftarrow \mathtt{tt}$$
$$\mathrm{temperature} \gtrdot (\mathrm{temperature} > 27) : \mathrm{heating} \leftarrow \mathtt{ff}$$
$$\mathrm{airButton} \gtrdot (\mathrm{airButton} = \mathtt{tt}) : \mathrm{conditioning} \leftarrow \mathtt{ff}$$
$$\mathrm{humidity}\ \mathrm{temperature} \gtrdot \\ (2 + 0.5 * \mathrm{temperature} < \mathrm{humidity} \wedge 38 - \mathrm{temperature} < \mathrm{humidity}) : \\ \mathrm{conditioning} \leftarrow \mathtt{tt}$$

Example: smart HVAC system (cont.)

- Temperature sensor:

$$\Sigma_t = [\,\mathrm{temperature} \mapsto 19 \quad \mathrm{node} \mapsto \text{'}\mathit{tempSens}\text{'}]$$
$$R_t \triangleq \mathrm{temperature} \gg @(\mathrm{node} = \text{'}\mathit{system}\text{'}) : \overline{\mathrm{temperature}} \leftarrow \mathrm{temperature}$$

- Humidity sensor:

$$\Sigma_h = [\,\mathrm{humidity} \mapsto 40 \quad \mathrm{node} \mapsto \text{'}\mathit{humSens}\text{'}]$$
$$R_h \triangleq \mathrm{humidity} \gg @(\mathrm{node} = \text{'}\mathit{system}\text{'}) : \overline{\mathrm{humidity}} \leftarrow \mathrm{humidity}$$

- Invariant on control system node:

$$\iota_s = \neg(\mathrm{conditioning} \land \mathrm{heating})$$

Smart HVAC revisited: without system node

- Heating and conditioning controllers are moved to temperature and humidity sensor nodes

$$R_t\langle \Sigma_t, \varnothing\rangle \parallel R_h\langle \Sigma_h, \varnothing\rangle$$

- Temperature node:

$$\Sigma_t = [\,\text{temperature} \mapsto 19 \quad \text{heating} \mapsto \mathtt{ff}\,]$$

$$\text{temperature} \gtrdot (\mathtt{tt}) : \overline{\text{temperature}} \leftarrow \text{temperature}$$

$$\text{temperature} \gtrdot (\text{temperature} < 18) : \text{heating} \leftarrow \mathtt{tt}$$

$$\text{temperature} \gtrdot (\text{temperature} > 27) : \text{heating} \leftarrow \mathtt{ff}$$

- Humidity node

$$\Sigma_h = [\,\text{humidity} \mapsto 40 \quad \text{conditioning} \mapsto \mathtt{ff} \quad \text{airButton} \mapsto \mathtt{ff}\,]$$

$$\text{airButton} \gtrdot (\text{airButton} = \mathtt{tt}) : \text{conditioning} \leftarrow \mathtt{ff}$$

$$\begin{aligned}&\text{humidity temperature} \gtrdot \\ &\quad (2 + 0.5 * \text{temperature} < \text{humidity} \wedge 38 - \text{temperature} < \text{humidity}) : \\ &\qquad \text{conditioning} \leftarrow \mathtt{tt}\end{aligned}$$

And what about the invariant?

- The invariant which was *local* to control system, now becomes a *global invariant*, predicating on variables of different nodes:

$$I = \neg(\text{conditioning}_h \wedge \text{heating}_t)$$

- How can we enforce this invariant without a central node?

From Global to Local Invariants

- We can guarantee global invariants in CASs by projecting a *global* invariant to many node-level, *local* invariants
- The fulfillment of local invariants, under specific assumptions, guarantees the fulfillment of the corresponding global invariant
- Requires the replication of invariant on all nodes having at least a resource appearing in it
- AbU nodes do not have knowledge about other node’s resources, hence have to propagate modifications to resources in the scope of global invariants to all interested nodes.
- Such synchronization is achieved by adding suitable AbU remote updates for each resource in the scope of global invariants

DecentralizeInvariant(S,I)

Algorithm `DecentralizeInvariant`(S, I)

```
/* the AbU system S is of the form R₁, î₁⟨Σ₁, Θ₁⟩ ‖ ... ‖ Rₙ, îₙ⟨Σₙ, Θₙ⟩   */
/* the global invariant I is of the form ι₁ ∧ ... ∧ ιₘ                   */
for i from 1 to n do
    for j from 1 to m do
        if vars(ιⱼ) ∩ vars(Σᵢ) ≠ ∅ then
            îᵢ := îᵢ ∧ ιⱼ
            for all x in vars(ιⱼ) \ vars(Σᵢ) do
                Σᵢ := Σᵢ ⊎ [x ↦ v]        // here ⊎ denotes state join and v ∈ type(x)
            end
            for all x in vars(ιⱼ) ∩ vars(Σᵢ) do
                Rᵢ := Rᵢ :: x ≫ @(tt) : x̄ ← x        // here :: denotes list concat
            end
        end
    end
end
return S
```

Revisited Smart HVAC system

- After the execution of DecentralizeInvariant(S,I):

$$\Sigma_t = [\,\text{temperature} \mapsto 19 \quad \text{heating} \mapsto \texttt{ff} \quad \text{conditioning} \mapsto \texttt{ff}\,]$$
$$\Sigma_h = [\,\text{humidity} \mapsto 40 \quad \text{conditioning} \mapsto \texttt{ff} \quad \text{airButton} \mapsto \texttt{ff} \quad \text{heating} \mapsto \texttt{ff}\,]$$

- Rule added to temperature node:

$$\text{heating} \gg @(\texttt{tt}) : \overline{\text{heating}} \leftarrow \text{heating}$$

- Rule added to the heating node:

$$\text{conditioning} \gg @(\texttt{tt}) : \overline{\text{conditioning}} \leftarrow \text{conditioning}$$

But does it work?

- Yes, if update execution in nodes respect some order
- Recall (Exec) rule: There is no *a priori* fixed scheduling policy

$$(\textsc{Exec})\ \frac{\begin{array}{c}\mathsf{upd} \in \Theta \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\ldots(\mathrm{x}_k, v_k) \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \ldots v_k/\mathrm{x}_k] \quad \Sigma' \models \iota \\ \Theta'' = \Theta \setminus \{\mathsf{upd}\} \quad X = \{\mathrm{x}_i \mid i \in [1..k] \wedge \Sigma(\mathrm{x}_i) \neq \Sigma'(\mathrm{x}_i)\} \\ \Theta' = \Theta'' \cup \mathsf{LocalUpds}(R, X, \Sigma') \quad T = \mathsf{ExtTasks}(R, X, \Sigma')\end{array}}{R, \iota\langle\Sigma, \Theta\rangle \xrightarrow{\mathsf{upd} \triangleright T} R, \iota\langle\Sigma', \Theta'\rangle}$$

- But synchronization updates must be executed **before** any other pending update in pool, otherwise they can be dropped due to invariant invalidation
- This calls for **priority scheduling**

LTS semantics with priority

- Labels: (P,T); upd $\triangleright (P,T)$; upd $\blacktriangleright (P,T)$
  where P is a list of high priority task
- (Exec) rules are modified accordingly

$$
(\textsc{ExecP})\ \frac{\begin{array}{c}\mathsf{upd} \in \hat{\Theta} \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \quad \Sigma' \models \iota \\ X = \{\mathrm{x}_i \mid i \in [1..k] \wedge \Sigma(\mathrm{x}_i) \neq \Sigma'(\mathrm{x}_i)\} \quad \mathsf{LocalUpds}(R, X, \Sigma') = (\hat{\Theta}'', \Theta'') \\ \hat{\Theta}' = (\hat{\Theta} \setminus \{\mathsf{upd}\}) \cup \hat{\Theta}'' \quad \Theta' = \Theta \cup \Theta'' \quad \mathsf{ExtTasks}(R, X, \Sigma') = (P, T)\end{array}}{R, \iota\langle \Sigma, (\hat{\Theta}, \Theta)\rangle \xrightarrow{\mathsf{upd} \triangleright (P,T)}_{\bullet} R, \iota\langle \Sigma', (\hat{\Theta}', \Theta')\rangle}
$$

$$
(\textsc{Exec})\ \frac{\begin{array}{c}\hat{\Theta} = \varnothing \quad \mathsf{upd} \in \Theta \quad \mathsf{upd} = (\mathrm{x}_1, v_1)\dots(\mathrm{x}_k, v_k) \quad \Sigma' = \Sigma[v_1/\mathrm{x}_1 \dots v_k/\mathrm{x}_k] \quad \Sigma' \models \iota \\ X = \{\mathrm{x}_i \mid i \in [1..k] \wedge \Sigma(\mathrm{x}_i) \neq \Sigma'(\mathrm{x}_i)\} \quad \mathsf{LocalUpds}(R, X, \Sigma') = (\hat{\Theta}'', \Theta'') \\ \hat{\Theta}' = \hat{\Theta} \cup \hat{\Theta}'' \quad \Theta' = (\Theta \setminus \{\mathsf{upd}\}) \cup \Theta'' \quad \mathsf{ExtTasks}(R, X, \Sigma') = (P, T)\end{array}}{R, \iota\langle \Sigma, (\hat{\Theta}, \Theta)\rangle \xrightarrow{\mathsf{upd} \triangleright (P,T)}_{\bullet} R, \iota\langle \Sigma', (\hat{\Theta}', \Theta')\rangle}
$$

Soundness of Decentralized invariants

- Priority semantics guarantees that local invariants are enough for enforcing global invariants

Theorem 1 (Local Invariants Soundness). *Let* S_ℓ *be a system obtained from an AbU system* S *by decentralizing the invariant* I *as per Algorithm 1. If* S_ℓ *satisfies* I*, then for all* S' *reachable from* S_ℓ*,* S' *satisfies* I*.*

Conclusions

- Global invariants can be implemented by means of local invariants, provided that the local execution of updates respect priority of synchronization messages
- Future work:
    - Other kinds of properties, e.g. liveness, fairness, etc.
    - Temporal properties
    - Non-interference
    - *Resilience*: how to recover an invariant when it fails?

Thanks for your attention!

Questions?

https://github.com/abu-lang