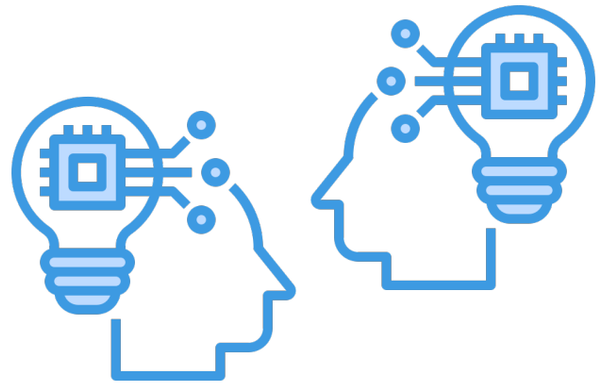
SECONDO WORKSHOP SU MODELLAZIONE E VERIFICA FORMALE DI SISTEMI DI DIALOGO, ARGUMENTATION PER CYBERCRIME E CIVIL SECURITY APPLICATIONS

Argumentative agents with local store

CARLO TATICCHI

RIMINI - 05/02/2026

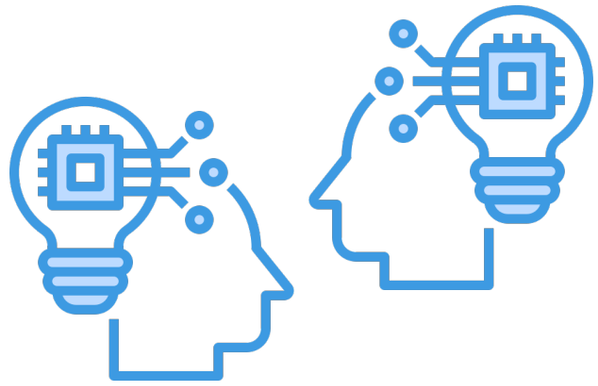
Context and Goal



- Concurrent language based on computational argumentation (TCLA)
- Agents interact and reason asynchronously

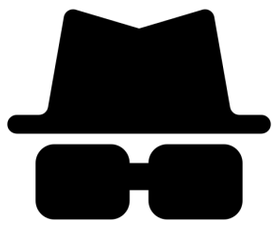
$$P ::= \mathit{let}\ C\ \mathit{in}\ A$$
$$C ::= p(x) :: A \mid C, C$$
$$A ::= \mathit{success} \mid \mathit{failure} \mid \mathit{add}(Arg, R) \to A \mid \mathit{rmv}(Arg, R) \to A \mid E \mid A \parallel A \mid \mathit{new}\ S\ \mathit{in}\ A \mid p(x)$$
$$E ::= \mathit{check}_t(Arg, R) \to A \mid \mathit{c\text{-}test}_t(a, l, \sigma) \to A \mid \mathit{s\text{-}test}_t(a, l, \sigma) \to A \mid E + E \mid E +_P E$$

Context and Goal



- Concurrent language based on computational argumentation (TCLA)
- Agents interact and reason asynchronously

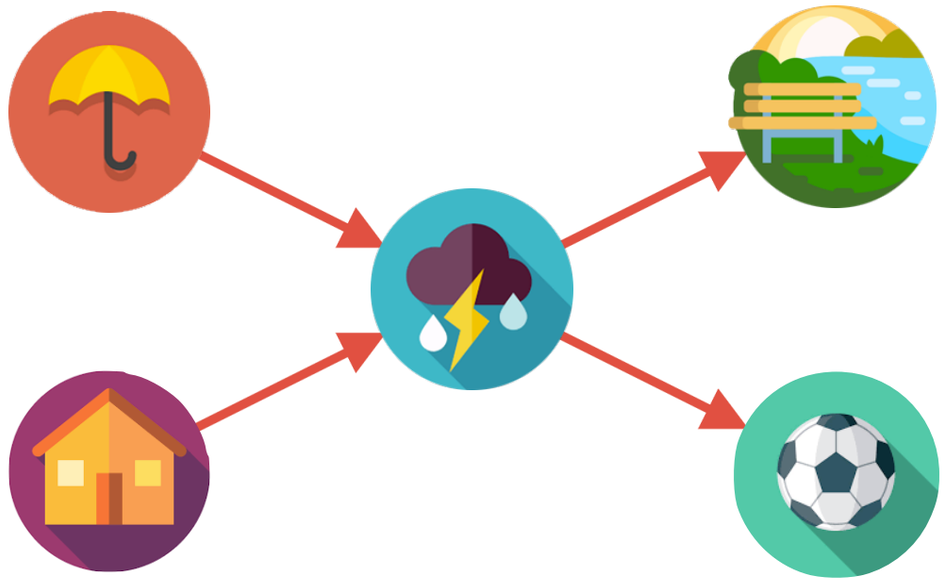
$$P ::= \mathit{let}\ C\ \mathit{in}\ A$$
$$C ::= p(x) :: A \mid C, C$$
$$A ::= \mathit{success} \mid \mathit{failure} \mid \mathit{add}(Arg, R) \to A \mid \mathit{rmv}(Arg, R) \to A \mid E \mid A \| A \mid \mathit{new}\ S\ \mathit{in}\ A \mid p(x)$$
$$E ::= \mathit{check}_t(Arg, R) \to A \mid \mathit{c\text{-}test}_t(a, l, \sigma) \to A \mid \mathit{s\text{-}test}_t(a, l, \sigma) \to A \mid E + E \mid E +_P E$$



- Introduce a notion of locality
- Enable private and strategic reasoning via local stores

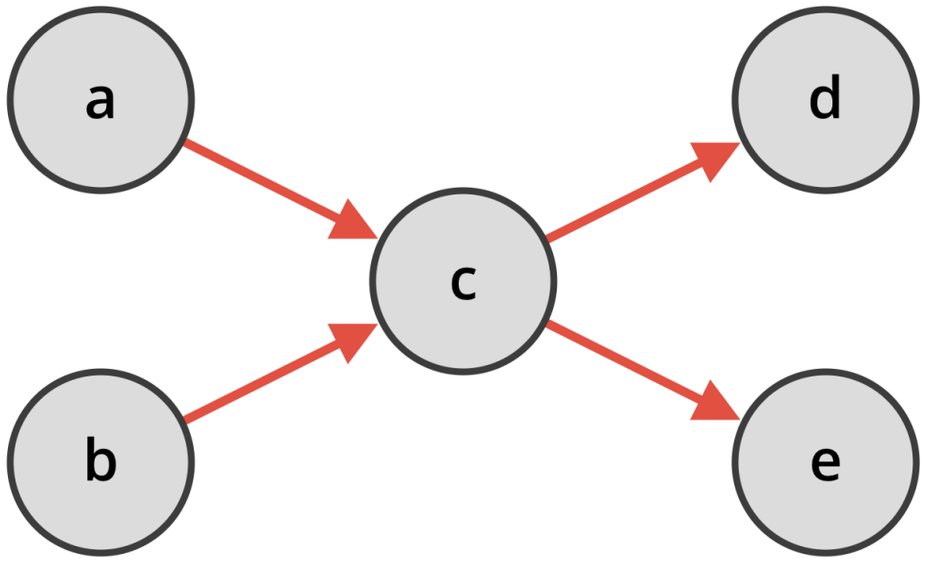
Computational Argumentation

- Abstract Argumentation Framework $\langle \text{Arg}, R \rangle$
- Represents and evaluates arguments
- Conflict-freeness and admissibility
- Acceptance Semantics



Computational Argumentation

- Abstract Argumentation Framework $\langle \text{Arg}, R \rangle$
- Represents and evaluates arguments
- Conflict-freeness and admissibility
- Acceptance Semantics

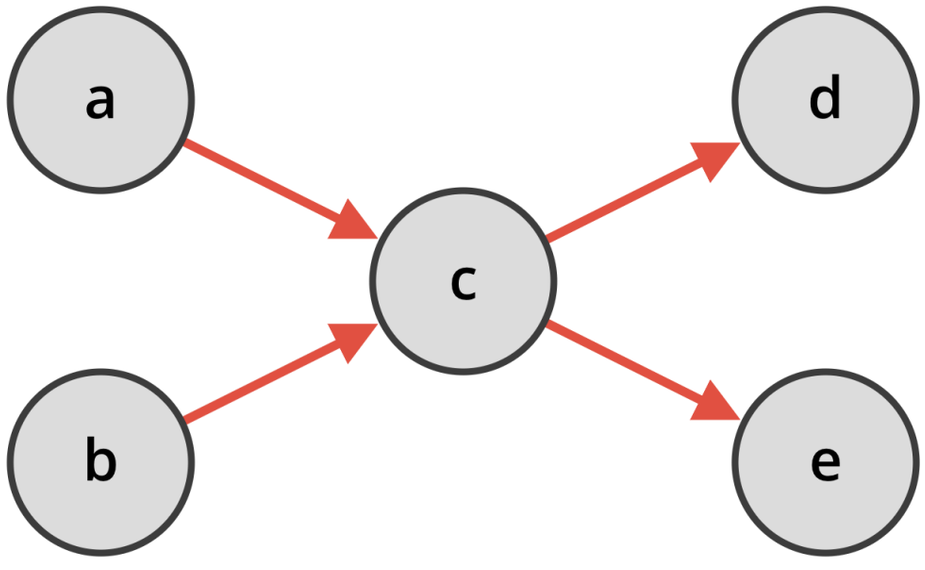


Computational Argumentation

- Abstract Argumentation Framework $\langle \text{Arg}, R \rangle$
- Represents and evaluates arguments
- Conflict-freeness and admissibility
- Acceptance Semantics

$a \in \text{Arg}$ is **defended** iff

$\forall b \in \text{Arg}$ s.t. $(b, a) \in R \,.\, \exists c \in \text{Arg}$ s.t. $(c, b) \in R$

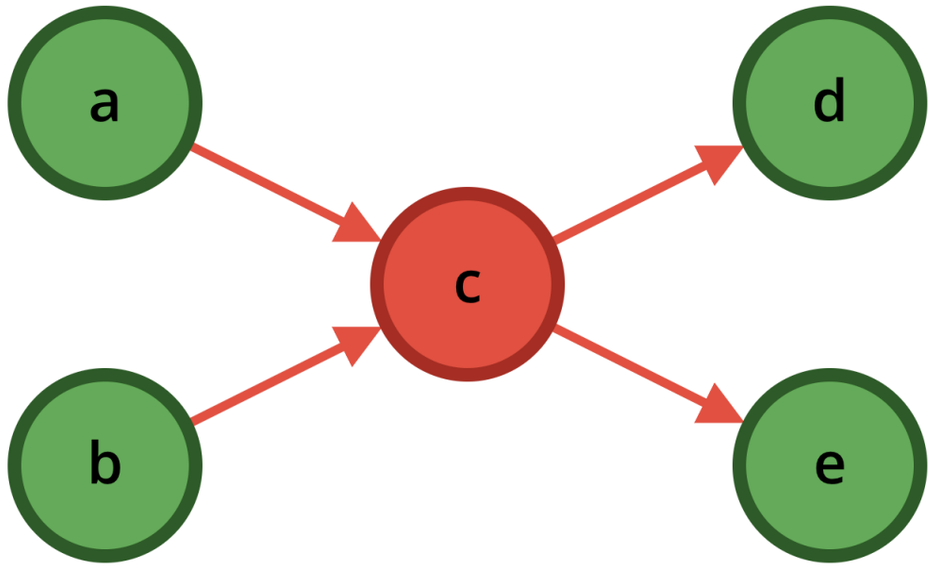


Computational Argumentation

- Abstract Argumentation Framework $\langle \text{Arg}, R \rangle$
- Represents and evaluates arguments
- Conflict-freeness and admissibility
- Acceptance Semantics

$a \in \text{Arg}$ is **defended** iff

$\forall b \in \text{Arg}$ s.t. $(b, a) \in R \,.\, \exists c \in \text{Arg}$ s.t. $(c, b) \in R$



Add and remove

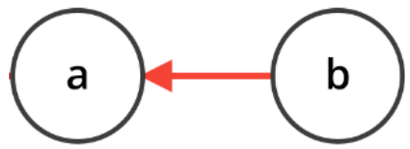
$$\langle \textit{add}(Arg', R') \to A, \langle Arg, R \rangle, T \rangle \longrightarrow \langle A, \langle Arg \cup Arg', (R \cup R')_{\|(Arg \cup Arg')} \rangle, \textit{dec}(T) \rangle \qquad \textbf{Ad}$$

$$\langle \textit{rmv}(Arg', R') \to A, \langle Arg, R \rangle, T \rangle \longrightarrow \langle A, \langle Arg \setminus Arg', (R \setminus R')_{\|(Arg \setminus Arg')} \rangle, \textit{dec}(T) \rangle \qquad \textbf{Re}$$

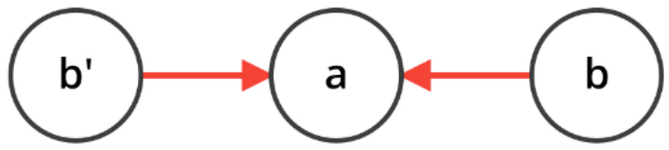
Add and remove

$$\langle \textit{add}(Arg', R') \to A, \langle Arg, R \rangle, T \rangle \longrightarrow \langle A, \langle Arg \cup Arg', (R \cup R')_{\|(Arg \cup Arg')} \rangle, \textit{dec}(T) \rangle \qquad \textbf{Ad}$$

$$\langle \textit{rmv}(Arg', R') \to A, \langle Arg, R \rangle, T \rangle \longrightarrow \langle A, \langle Arg \setminus Arg', (R \setminus R')_{\|(Arg \setminus Arg')} \rangle, \textit{dec}(T) \rangle \qquad \textbf{Re}$$



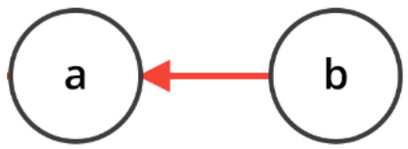
- Example 1: ***add({b'}, {(b',a)})*** → ***success***



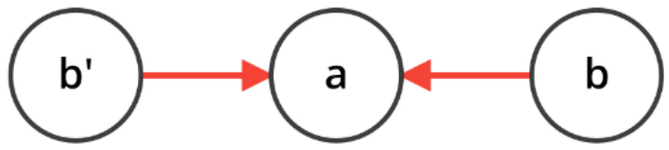
Add and remove

$$\langle \textit{add}(Arg', R') \to A, \langle Arg, R \rangle, T \rangle \longrightarrow \langle A, \langle Arg \cup Arg', (R \cup R')_{\|(Arg \cup Arg')} \rangle, \textit{dec}(T) \rangle \qquad \textbf{Ad}$$

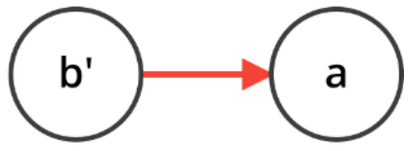
$$\langle \textit{rmv}(Arg', R') \to A, \langle Arg, R \rangle, T \rangle \longrightarrow \langle A, \langle Arg \setminus Arg', (R \setminus R')_{\|(Arg \setminus Arg')} \rangle, \textit{dec}(T) \rangle \qquad \textbf{Re}$$



- Example 1: ***add({b'}, {(b',a)})*** → ***success***



- Example 2: ***rmv({b}, {})*** → ***success***



Timeout Environments with TC

- The passing of time is handled on a **global clock** via a timeout environment $T : I \rightharpoonup \mathbb{N} \cup \{\infty\}$
- We can update/decrement timers in T

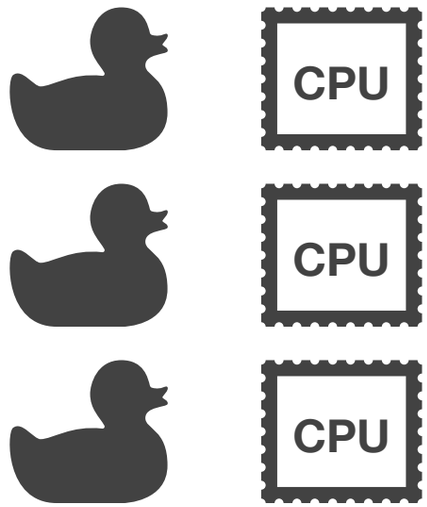
$$T[\bar{I} : \bar{t}](I) = \begin{cases} T(I) & \text{if } I \neq \bar{I} \\ \bar{t} & \text{otherwise} \end{cases}$$

$$dec(T)(I) = \begin{cases} T(I) - 1 & \text{if } 0 < T(I) \in \mathbb{N} \\ T(I) & \text{if } T(I) = 0 \text{ or } T(I) = \infty \end{cases}$$

I	T(I)
Agent A	4
Agent B	1
Agent C	2

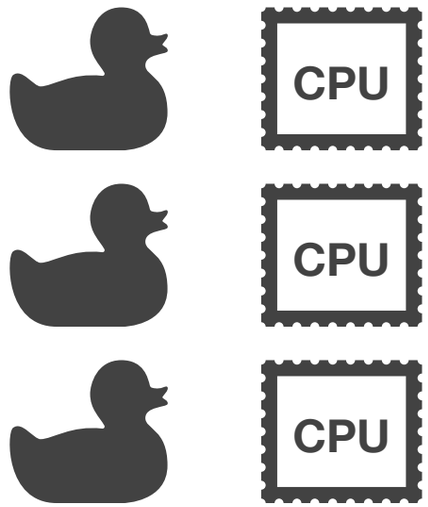
Handling Parallelism

- True Concurrency (TC)
  - Infinite number of processors
  - All concurrent agents executed simultaneously

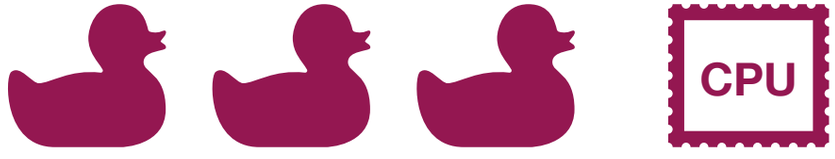


Handling Parallelism

- True Concurrency (TC)
  - Infinite number of processors
  - All concurrent agents executed simultaneously



- Alternative: interleaving
  - 1 processor
  - 1 operation at a time



True Concurrency

$$\frac{\langle A_1, F, T\rangle \longrightarrow \langle A_1', F', T_1\rangle,\ \ \langle A_2, F, T\rangle \longrightarrow \langle A_2', F'', T_2\rangle}{\langle A_1 \parallel A_2, F, T\rangle \longrightarrow \langle A_1' \parallel A_2', *(F, F', F''), T_1 \cup T_2\rangle}$$

TC

$$*\,(F, F', F'') := (F' \cap F'') \cup ((F' \cup F'')\ F)$$

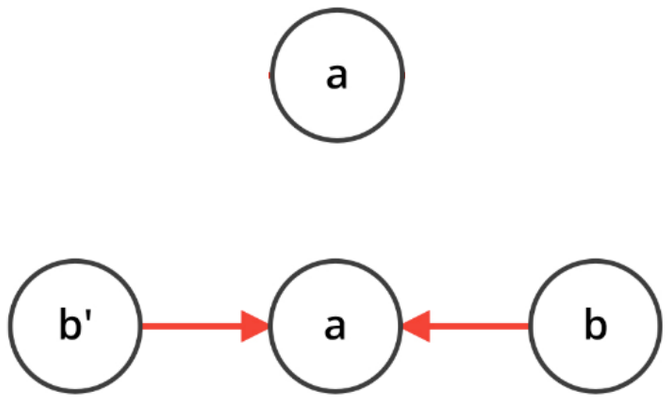
True Concurrency

$$\frac{\langle A_1, F, T\rangle \longrightarrow \langle A_1', F', T_1\rangle,\ \ \langle A_2, F, T\rangle \longrightarrow \langle A_2', F'', T_2\rangle}{\langle A_1 \parallel A_2, F, T\rangle \longrightarrow \langle A_1' \parallel A_2', *(F, F', F''), T_1 \cup T_2\rangle}$$

TC

$$*\,(F, F', F'') := (F' \cap F'') \cup ((F' \cup F'')\ F)$$

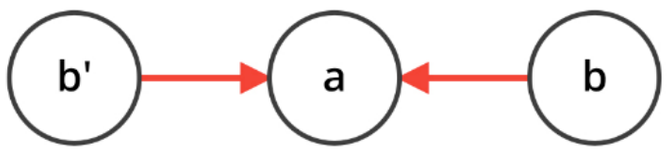
- Example: ***add({b}, {(b,a)}) → success ll add({b'}, {b',a}) → success***



Check

$$\frac{Arg' \subseteq Arg \wedge R' \subseteq R,\ t > 0}{\langle check_t(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle A, F, dec(T)\rangle} \quad \textbf{Ch1}$$

- Example 1: ***check*$_5$*({a}, {}) → success***



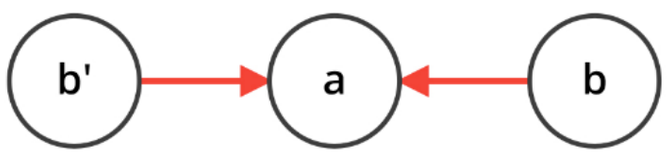
Check

$$\frac{Arg' \subseteq Arg \wedge R' \subseteq R,\ t > 0}{\langle check_t(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle A, F, dec(T)\rangle} \quad \textbf{Ch1}$$

$$\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R),\ t > 0}{\langle check_t(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle check_I(Arg', R') \rightarrow A, F, dec(T[I : t])\rangle} \quad \textbf{Ch2}$$

where I is a fresh timeout identifier

- Example 2: ***check*$_5$*({c}, {}) → success***
  - Next step: ***check*$_A$*({c}, {}) → success***



I	T(I)
A	4

Check

$$\frac{Arg' \subseteq Arg \wedge R' \subseteq R,\ t > 0}{\langle check_t(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle A, F, dec(T)\rangle} \quad \textbf{Ch1}$$

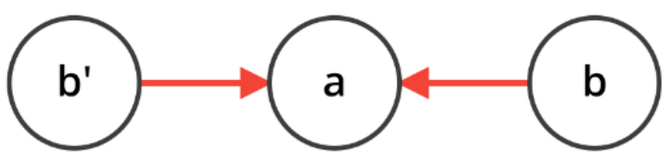
$$\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R),\ t > 0}{\langle check_t(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle check_I(Arg', R') \rightarrow A, F, dec(T[I : t])\rangle} \quad \textbf{Ch2}$$

where I is a fresh timeout identifier

$$\frac{Arg' \subseteq Arg \wedge R' \subseteq R,\ T(I) > 0}{\langle check_I(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle A, F, dec(T)\rangle} \quad \textbf{Ch3}$$

$$\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R),\ T(I) > 0}{\langle check_I(Arg', R') \rightarrow A, F, T\rangle \longrightarrow \langle check_I(Arg', R') \rightarrow A, F, dec(T)\rangle} \quad \textbf{Ch4}$$

- Example 3: ***check$_A$({c}, {}) → success***



I	T(I)
A	4

Check

$$\frac{Arg' \subseteq Arg \wedge R' \subseteq R,\ t > 0}{\langle check_t(Arg', R') \to A, F, T\rangle \longrightarrow \langle A, F, dec(T)\rangle} \quad \textbf{Ch1}$$

$$\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R),\ t > 0}{\langle check_t(Arg', R') \to A, F, T\rangle \longrightarrow \langle check_I(Arg', R') \to A, F, dec(T[I : t])\rangle} \quad \textbf{Ch2}$$

where I is a fresh timeout identifier

$$\frac{Arg' \subseteq Arg \wedge R' \subseteq R,\ T(I) > 0}{\langle check_I(Arg', R') \to A, F, T\rangle \longrightarrow \langle A, F, dec(T)\rangle} \quad \textbf{Ch3}$$

$$\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R),\ T(I) > 0}{\langle check_I(Arg', R') \to A, F, T\rangle \longrightarrow \langle check_I(Arg', R') \to A, F, dec(T)\rangle} \quad \textbf{Ch4}$$

$$\langle check_0(Arg', R') \to A, F, T\rangle \longrightarrow \langle failure, F, dec(T)\rangle \quad \textbf{Ch5}$$

$$\frac{T(I) = 0}{\langle check_I(Arg', R') \to A, F, T\rangle \longrightarrow \langle failure, F, dec(T)\rangle} \quad \textbf{Ch6}$$

Locality with the *new* operator

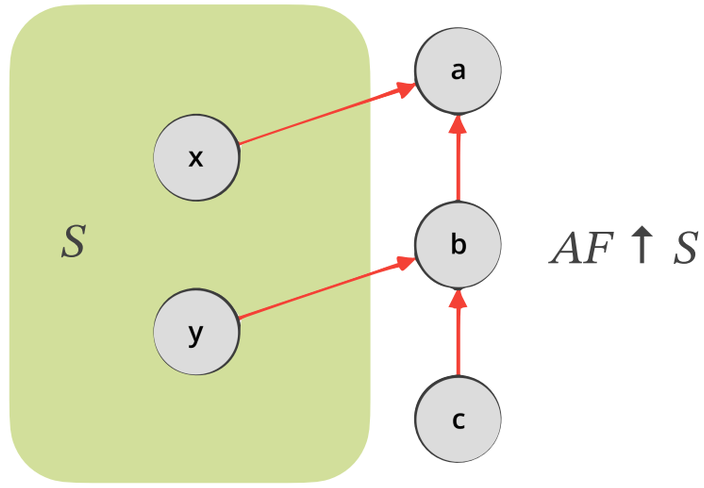
$$\frac{\langle A, (AF \uparrow S) \cup AF_{loc}, T\rangle \longrightarrow \langle \mathsf{B}, AF', T'\rangle}{\langle \textit{new } S \textit{ in } A^{AF_{loc}}, AF, T\rangle \longrightarrow \langle \textit{new } S \textit{ in } B^{AF' \downarrow S}, (AF' \uparrow S) \cup (AF'' \downarrow S), T'\rangle} \quad \textbf{Loc}$$

$R_{|S} = \{(a,b) \in R \mid a \in S \vee b \in S\}$

$F \downarrow S = \langle \mathsf{Arg} \cap S', R_{|S}\rangle$ where $S' = S \cup \{b \mid (b,c) \in R_{|S} \vee (c,b) \in R_{|S}\}$

$F \uparrow S = \langle \mathsf{Arg} \backslash S, R \backslash R_{|S}\rangle$

- *new* S *in* A behaves like A where arguments in S are local to A
- AF_{loc} contains information on S which is hidden from the external *AF*



New - some issues

- No exit from a local context
  - Ad-hoc mechanisms needed to propagate information to the global level
  - Parallel processes needs to be re-synchronised

New - some issues

- No exit from a local context
  - Ad-hoc mechanisms needed to propagate information to the global level
  - Parallel processes needs to be re-synchronised

- Locality is temporary
  - No persistent storage
  - Opening a new *new* loses all previously written information

Local stores

- **Proposal: introduce operators with restricted context that work in permanent local stores**
- We looked at Spatial CCP[1]

$$\frac{\langle P, c^i \rangle \longrightarrow \langle P', c' \rangle}{\langle [P]_i, c \rangle \longrightarrow \langle [P']_i, c \sqcup s_i(c') \rangle} \quad \text{Loc}$$

- Process P executes over the local store c_i (the portion of the global store visible to i)
- When P produces new information locally, the spatial process $[P]_i$ propagates it to the global store via the space function s_i

1. Knight, S., Palamidessi, C., Panangaden, P., Valencia, F.D.: Spatial and epistemic modalities in constraint-based process calculi. In CONCUR 2012 - Concurrency Theory - 23rd International Conference, 4-7, 2012

Local Add in TCLA

- $[add(\{a\}, \{\})]_i \rightarrow add(\{a\}, \{\})$: the agent i cannot detect whether a already exists in another local store, so a is added to the global store by default

Local Add in TCLA

- $[add(\{a\}, \{\})]_i \rightarrow add(\{a\}, \{\})$: the agent i cannot detect whether a already exists in another local store, so a is added to the global store by default

- $add(\{a\}, \{\}) \rightarrow [add(\{a\}, \{\})]_i$: adds a to the global store and then to the local one, even if agent i already knows that a is present globally

Local Add in TCLA

- $[add(\{a\}, \{\})]_i \rightarrow add(\{a\}, \{\})$: the agent i cannot detect whether a already exists in another local store, so a is added to the global store by default

- $add(\{a\}, \{\}) \rightarrow [add(\{a\}, \{\})]_i$: adds a to the global store and then to the local one, even if agent i already knows that a is present globally

- **We define an agent's *view* as the union of its local store and the global one.** Therefore, the private "copy" of a disappears under *check* and *test*

Local Remove in TCLA

- Global actions do not affect local stores

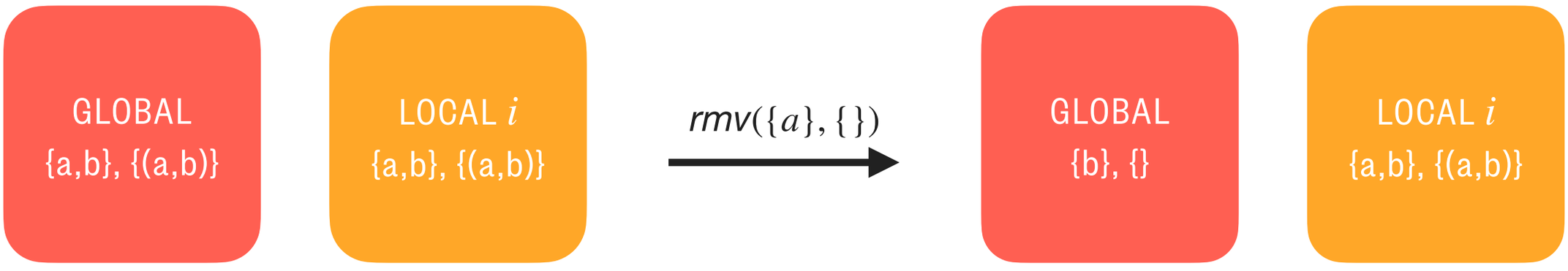
Local Remove in TCLA

- Global actions do not affect local stores

- If an argument a is present both in the global store and in some agent's local store, executing *rmv*$(\{a\}, \{\})$ removes a only from the global store and the local copies remain

Local Remove in TCLA

- **Global actions do not affect local stores**
- If an argument a is present both in the global store and in some agent's local store, executing $rmv(\{a\}, \{\})$ removes a only from the global store and the local copies remain
- The same reasoning applies to attacks: removing the attack globally does not affect its local occurrences (local stores are self-contained AFs, so if an attack is stored locally, the involved arguments are also stored locally)



Local Check/Test in TCLA

- *check*$(\{a\}, \{\})$ verifies whether a is present in the global store GLOBAL
- $[$*check*$(\{a\}, \{\})]_i$ verifies whether a is present in agent i's view (local store ∪ global store) GLOBAL LOCAL i_1
- Same for *test*

Local Check/Test in TCLA

- *check*$(\{a\}, \{\})$ verifies whether a is present in the global store GLOBAL
- $[$*check*$(\{a\}, \{\})]_i$ verifies whether a is present in agent i's view (local store ∪ global store) GLOBAL LOCAL i_1
- Same for *test*

- Do we need a local-only *check*? What for? LOCAL i_1
- What about local-only *tests?* Probably not

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner
- $add_i(\{a\}, \{(a, b)\})$ produces two outcomes:

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner
- $add_i(\{a\}, \{(a, b)\})$ produces two outcomes:
  1. adds a and (a, b) to the store (subject to usual constraints)

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner
- $add_i(\{a\}, \{(a, b)\})$ produces two outcomes:

1. adds a and (a, b) to the store (subject to usual constraints)
2. records ownerships $own(a, i)$, $own((a, b), i)$

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner
- $add_i(\{a\}, \{(a, b)\})$ produces two outcomes:
  1. adds a and (a, b) to the store (subject to usual constraints)
  2. records ownerships $own(a, i)$, $own((a, b), i)$
- $rmv_i(\{a\}, \{\})$ succeeds only if $own(a, i)$

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner
- $add_i(\{a\}, \{(a, b)\})$ produces two outcomes:
  1. adds a and (a, b) to the store (subject to usual constraints)
  2. records ownerships $own(a, i)$, $own((a, b), i)$

- $rmv_i(\{a\}, \{\})$ succeeds only if $own(a, i)$
- All arguments and attacks should have an owner (so $add(\{a\}, \{(a, b)\})$ not allowed)

Ownership

- Problem: agent i adds an argument a to the global store, and then agent j removes it
- Proposal: associate arguments and attacks with an owner
- $add_i(\{a\}, \{(a, b)\})$ produces two outcomes:
  1. adds a and (a, b) to the store (subject to usual constraints)
  2. records ownerships $own(a, i)$, $own((a, b), i)$

- $rmv_i(\{a\}, \{\})$ succeeds only if $own(a, i)$
- All arguments and attacks should have an owner (so $add(\{a\}, \{(a, b)\})$ not allowed)
- Local add/remove have implicit owner (so $[add_i(\{a\}, \{(a, b)\})]_j$ not allowed)

Conclusion

- **TCLA** prototype available at conarg.dmi.unipg.it/cla

Conclusion

- **TCLA** prototype available at conarg.dmi.unipg.it/cla

- Next steps:
  - Extend **TCLA** to support **local stores** and **ownership**

Conclusion

- **TCLA** prototype available at conarg.dmi.unipg.it/cla

- Next steps:
  - Extend **TCLA** to support **local stores** and **ownership**
  - Validate the approach with concrete examples of real-life interaction scenarios

Conclusion

- **TCLA** prototype available at conarg.dmi.unipg.it/cla

- Next steps:
  - Extend **TCLA** to support **local stores** and **ownership**
  - Validate the approach with concrete examples of real-life interaction scenarios
  - Improve interoperability with existing frameworks beyond classical AFs (e.g., ASPIC+, ABA)

Conclusion

- **TCLA** prototype available at conarg.dmi.unipg.it/cla

- Next steps:
  - Extend **TCLA** to support **local stores** and **ownership**
  - Validate the approach with concrete examples of real-life interaction scenarios
  - Improve interoperability with existing frameworks beyond classical AFs (e.g., ASPIC+, ABA)
  - Apply **TCLA** to chatbot-based interactions for persuasion and decision making (NLP layer to map natural language text into arguments + computational argumentation to guide system's responses)

Thank you for your attention!

Argumentative agents with local store

CARLO TATICCHI