An Abstraction Technique for the Verification of Artifact-Centric Multi-Agent Systems

Francesco Belardinelli

Department of Computing
Imperial College London, UK

joint work with F. Patrizi and A. Lomuscio
within the EU Project ACSI (Artifact-Centric Service Interoperation)

Department of Computing, University of Liverpool
February 14, 2012
Overview

1. **Motivation:** Artifact Systems as *data-aware systems*
2. **Main task:** *formal* verification of (infinite-state) Artifact Systems
3. **Key contribution:** verification of *bounded* AS is decidable
4. Conclusion and future directions
Artifacts and Artifact Systems

Recent paradigm for Business Process modeling and development [CH09]

- Artifact: data model + lifecycle
  - (Nested) records equipped with actions
  - Actions may affect several artifacts

- Artifact System: set of interacting artifacts

- Data and processes are given same emphasis
Artifact Systems

Order-to-Cash Scenario

Customer

<table>
<thead>
<tr>
<th>Purchase Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desk 1</td>
</tr>
<tr>
<td>Chair 4</td>
</tr>
</tbody>
</table>

Manufacturer

<table>
<thead>
<tr>
<th>Material Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desk Legs 4</td>
</tr>
<tr>
<td>Chair Legs 16</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Supplier

<table>
<thead>
<tr>
<th>Material Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hammer Nails 1000</td>
</tr>
<tr>
<td>Glue 10</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Accept/reject

Accept/reject
Artifact Systems
Data Model

**PO**

<table>
<thead>
<tr>
<th>id</th>
<th>prod_code</th>
<th>offer</th>
<th>status</th>
</tr>
</thead>
</table>

- `createPO(prod_code)`
- `deletePO(id)`
- `addLinePO(id, prod_code, offer)`
- `...`

**WO**

<table>
<thead>
<tr>
<th>id</th>
<th>po_id</th>
<th>price</th>
<th>status</th>
</tr>
</thead>
</table>

- `createWO(po_id)`
- `deleteWO(id)`
- `addLineWO(id, po_id, price)`
- `...`

**MO**

<table>
<thead>
<tr>
<th>id</th>
<th>wo_id</th>
<th>cost</th>
<th>status</th>
</tr>
</thead>
</table>

- `createMO(wo_id)`
- `deleteMO(id)`
- `addLineMO(id, wo_id, status)`
- `...`

**Products**

<table>
<thead>
<tr>
<th>prod_code</th>
<th>budget</th>
</tr>
</thead>
</table>

**Materials**

<table>
<thead>
<tr>
<th>mat_code</th>
<th>cost</th>
</tr>
</thead>
</table>
As the process goes on, artifact actions are executed.
  
  e.g., the Purchase Order is sent to the Manufacturer.
As the process goes on, artifact actions are executed.
  - e.g., the Purchase Order is sent to the Manufacturer.

Actions add/remove artifacts or change artifact attributes.
  - e.g., the PO status changes from *created* to *submitted*.
As the process goes on, artifact actions are executed.
  ▶ e.g., the Purchase Order is sent to the Manufacturer.
Actions add/remove artifacts or change artifact attributes.
  ▶ e.g., the PO status changes from created to submitted.
The whole system can be seen as a *data-aware* dynamic system.
  ▶ At every step, an action yields a change in the current state.
We can give a (partial) representation of AS as FSM.

(a) Purchase Order lifecycle

(b) Work Order Lifecycle

(c) Material Order lifecycle
Databases and Artifact Systems

We introduce some (basic) notions on databases to formalise data models.

A database schema is a finite set \( D = \{ P_1/\alpha_1, \ldots, P_n/\alpha_n \} \) of predicate symbols \( P_i \) with their arity \( \alpha_i \in \mathbb{N} \).

In the order-to-cash scenario \( D = \{ \text{Products}/2, \text{PO}/4, \text{WO}/4, \text{Materials}/2, \text{MO}/4 \} \).

A \( D \)-interpretation on a (possibly infinite) domain \( U \) is a mapping \( D \) associating each predicate symbol \( P_i \) with a finite \( \alpha_i \)-ary relation on \( U \).

The active domain \( \text{adom}(D) \) of each \( D \)-instance \( D \) is finite.

<table>
<thead>
<tr>
<th>PO id</th>
<th>prod code</th>
<th>offer</th>
<th>status</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#12</td>
<td>$50</td>
<td>prepared</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>#24</td>
<td>$120</td>
<td>shipped</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>#45</td>
<td>$80</td>
<td>paid</td>
<td></td>
</tr>
</tbody>
</table>
Databases and Artifact Systems

We introduce some (basic) notions on databases to formalise data models.

A **database schema** is a *finite* set \( \mathcal{D} = \{ P_1/a_1, \ldots, P_n/a_n \} \) of predicate symbols \( P_i \) with their arity \( a_i \in \mathbb{N} \).

- In the order-to-cash scenario
  \( \mathcal{D} = \{ \text{Products}/2, \text{PO}/4, \text{WO}/4, \text{Materials}/2, \text{MO}/4 \} \)
Databases and Artifact Systems

We introduce some (basic) notions on databases to formalise data models.

A *database schema* is a *finite* set $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$ of predicate symbols $P_i$ with their arity $a_i \in \mathbb{N}$.

- In the order-to-cash scenario
  $\mathcal{D} = \{\text{Products}/2, \text{PO}/4, \text{WO}/4, \text{Materials}/2, \text{MO}/4\}$

A $\mathcal{D}$-interpretation on a (possibly infinite) domain $U$ is a mapping $D$ associating each predicate symbol $P_i$ with a *finite* $a_i$-ary relation on $U$.

<table>
<thead>
<tr>
<th>id</th>
<th>prod_code</th>
<th>offer</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#12</td>
<td>$50</td>
<td>prepared</td>
</tr>
<tr>
<td>2</td>
<td>#24</td>
<td>$120</td>
<td>shipped</td>
</tr>
<tr>
<td>4</td>
<td>#45</td>
<td>$80</td>
<td>paid</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Databases and Artifact Systems

We introduce some (basic) notions on databases to formalise data models.

A **database schema** is a *finite* set $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$ of predicate symbols $P_i$ with their arity $a_i \in \mathbb{N}$.

- In the order-to-cash scenario
  \[ \mathcal{D} = \{ \text{Products}/2, \text{PO}/4, \text{WO}/4, \text{Materials}/2, \text{MO}/4 \} \]

A **$\mathcal{D}$-interpretation** on a (possibly infinite) domain $U$ is a mapping $D$ associating each predicate symbol $P_i$ with a *finite* $a_i$-ary relation on $U$.

<table>
<thead>
<tr>
<th>id</th>
<th>prod_code</th>
<th>offer</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#12</td>
<td>$50</td>
<td>prepared</td>
</tr>
<tr>
<td>2</td>
<td>#24</td>
<td>$120</td>
<td>shipped</td>
</tr>
<tr>
<td>4</td>
<td>#45</td>
<td>$80</td>
<td>paid</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The active domain $adom(D)$ of each $\mathcal{D}$-instance $D$ is *finite.*
Databases and Artifact Systems

Given

- a \(\mathcal{D}\)-interpretation \(D\)
- an assignment \(\sigma : \text{Var} \rightarrow U\)
- an FO-formula \(\varphi \in \mathcal{L}_D\)

we inductively define satisfaction:

\[(D, \sigma) \models P_i(t_1, \ldots, t_\ell) \iff \langle \sigma(t_1), \ldots, \sigma(t_\ell) \rangle \in D(P_i)\]

\[(D, \sigma) \models t = t' \iff \sigma(t) = \sigma(t')\]

\[(D, \sigma) \models \neg \varphi \iff (D, \sigma) \not\models \varphi\]

\[(D, \sigma) \models \varphi \rightarrow \psi \iff (D, \sigma) \not\models \varphi \text{ or } (D, \sigma) \models \psi\]

\[(D, \sigma) \models \forall x \varphi \iff \text{for every } u \in \text{adom}(D), (D, \sigma^x_u) \models \varphi\]

Notice that we adopt an active domain semantics.
Databases and Artifact Systems

Given

- a \( \mathcal{D} \)-interpretation \( D \)
- an assignment \( \sigma : \text{Var} \rightarrow U \)
- an FO-formula \( \varphi \in \mathcal{L}_\mathcal{D} \)

we inductively define \textit{satisfaction}:

\[
\begin{align*}
(D, \sigma) \models P_i(t_1, \ldots, t_\ell) & \iff \langle \sigma(t_1), \ldots, \sigma(t_\ell) \rangle \in D(P_i) \\
(D, \sigma) \models t = t' & \iff \sigma(t) = \sigma(t') \\
(D, \sigma) \models \neg \varphi & \iff (D, \sigma) \not\models \varphi \\
(D, \sigma) \models \varphi \rightarrow \psi & \iff (D, \sigma) \not\models \varphi \text{ or } (D, \sigma) \models \psi \\
(D, \sigma) \models \forall x \varphi & \iff \text{for every } u \in \text{adom}(D), (D, \sigma^x_u) \models \varphi
\end{align*}
\]

Notice that we adopt an \textit{active domain semantics}.

\textit{Composition:} \( D \oplus D' \) is the \((\mathcal{D} \cup \mathcal{D}')\)-interpretation s.t. \( D \oplus D'(P_i) = D(P_i) \) and \( D \oplus D'(P'_i) = D'(P_i) \).
Artifact-centric Multi-agent Systems

- Artifacts are manipulated by agents, e.g., customers, manufacturers, suppliers.
Artifacts are manipulated by agents, e.g., customers, manufacturers, suppliers.

We introduce an agent-based model for AS inspired to [FHMV95].

An \textit{agent} is a tuple \(i = \langle D_i, L_i, Act_i, Pr_i \rangle\) where:

- \(D_i\) is the \textit{local database schema}
- \(L_i \subseteq D_i(U)\) is the set of \textit{local states}
- \(Act_i\) is the set of \textit{local actions}
- \(Pr_i : L_i \mapsto 2^{Act_i}\) is the \textit{local protocol function}
Artifacts are manipulated by agents, e.g., customers, manufacturers, suppliers.

We introduce an agent-based model for AS inspired to [FHMV95].

An agent is a tuple $i = \langle D_i, L_i, Act_i, Pr_i \rangle$ where:

- $D_i$ is the local database schema
- $L_i \subseteq D_i(U)$ is the set of local states
- $Act_i$ is the set of local actions
- $Pr_i : L_i \mapsto 2^{Act_i}$ is the local protocol function

The global database schema is such that $D = D_1 \cup \cdots \cup D_n$. 
Artifact-centric Multi-agent Systems

- Artifacts are manipulated by agents, e.g., customers, manufacturers, suppliers.
- We introduce an agent-based model for AS inspired to [FHMV95].
- An agent is a tuple $i = \langle D_i, L_i, Act_i, Pr_i \rangle$ where:
  - $D_i$ is the local database schema
  - $L_i \subseteq D_i(U)$ is the set of local states
  - $Act_i$ is the set of local actions
  - $Pr_i : L_i \mapsto 2^{Act_i}$ is the local protocol function

- The global database schema is such that $D = D_1 \cup \cdots \cup D_n$.
- Agents manipulate artifacts and have (partial) access to the information contained therein.
Example 1: the Order-to-Cash Scenario

- Agents: Customer, Manufacturer, Supplier.
Example 1: the Order-to-Cash Scenario

- Agents: Customer, Manufacturer, Supplier.
- Local database schemas:
  - $\mathcal{D}_C = \{Products, PO\}$
  - $\mathcal{D}_M = \{WO\}$
  - $\mathcal{D}_S = \{Materials, MO\}$
Example 1: the Order-to-Cash Scenario

- Agents: Customer, Manufacturer, Supplier.
- Local database schemas:
  - $D_C = \{\text{Products, PO}\}$
  - $D_M = \{\text{WO}\}$
  - $D_S = \{\text{Materials, MO}\}$
- Then $D = \{\text{Products, PO, WO, Materials, MO}\}$. 
Example 1: the Order-to-Cash Scenario

- Agents: Customer, Manufacturer, Supplier.
- Local database schemas:
  - $D_C = \{\text{Products, PO}\}$
  - $D_M = \{\text{WO}\}$
  - $D_S = \{\text{Materials, MO}\}$
- Then $D = \{\text{Products, PO, WO, Materials, MO}\}$.
- Parametric actions can introduce values from an infinite domain $U$:
  - $\text{createPO}(id, \text{prod\_code}, offer)$ in $Act_C$
  - $\text{createWO}(id, po\_id, price)$ in $Act_M$
  - $\text{createMO}(id, wo\_id, cost)$ in $Act_S$
Artifact-centric Multi-agent Systems

Agents are modules that can be composed together to obtain AC-MAS.
Artifact-centric Multi-agent Systems

Agents are modules that can be composed together to obtain AC-MAS.

- An **AC-MAS** is a tuple $\mathcal{P} = \langle S, U, D_0, \tau \rangle$ where
  - $S \subseteq L_1 \times \cdots \times L_n$ is the set of *reachable global states*
  - $U$ is the interpretation domain
  - $D_0 \in S$ is the *initial global state*
  - $\tau : S \times \text{Act} \mapsto 2^S$ is the *global transition function*
Artifact-centric Multi-agent Systems

Agents are modules that can be composed together to obtain AC-MAS.

- An **AC-MAS** is a tuple $\mathcal{P} = \langle S, U, D_0, \tau \rangle$ where
  - $S \subseteq L_1 \times \cdots \times L_n$ is the set of *reachable global states*
  - $U$ is the interpretation domain
  - $D_0 \in S$ is the *initial global state*
  - $\tau : S \times \text{Act} \mapsto 2^S$ is the *global transition function*

- Temporal transition: $D \to D'$ iff there is $\alpha$ s.t. $\tau(D, \alpha(\bar{u})) = D'$.

- Epistemic relation: $D \sim_i D'$ iff $D_i = D'_i$ for agent $i$. 
Artifact-centric Multi-agent Systems

Agents are modules that can be composed together to obtain AC-MAS.

- An **AC-MAS** is a tuple $\mathcal{P} = \langle S, U, D_0, \tau \rangle$ where
  - $S \subseteq L_1 \times \cdots \times L_n$ is the set of reachable global states
  - $U$ is the interpretation domain
  - $D_0 \in S$ is the initial global state
  - $\tau : S \times \text{Act} \mapsto 2^S$ is the global transition function

- Temporal transition: $D \rightarrow D'$ iff there is $\alpha$ s.t. $\tau(D, \alpha(\vec{u})) = D'$.
- Epistemic relation: $D \sim_i D'$ iff $D_i = D'_i$ for agent $i$.

AC-MAS are FO temporal epistemic structures, so a flavour of FO temporal epistemic logic can be used as specification language for AC-MAS.
Artifact-centric Multi-agent Systems

Intuition

- A transition system where each state is a $\mathcal{D}$-instance.
- As actions are executed, new states are generated.
- Action parameters can introduce new values.
- An infinite domain $U$ yields potentially infinitely many distinct states.
- In general, infinite branching and infinite run-length.
The Problem

Intuition

Does the system satisfy a (branching-time) temporal epistemic specification? E.g.:

- It is always the case that every artifact can be deleted
- There exists a way to create a certain number of artifacts
- The manufacturer knows that a product can be shipped only after assemblage
The Problem

Intuition

Does the system satisfy a (branching-time) *temporal epistemic* specification? E.g.:  
- It is always the case that every artifact can be deleted  
- There exists a way to create a certain number of artifacts  
- The manufacturer knows that a product can be shipped only after assemblage

Flavour of Model Checking, but:  
- *relational* states (database instances)  
- infinite interpretation domain  
- infinite state space
Verification Formalism: FO-CTLK

How to specify system properties?

Definition (Syntax of FO-CTLK)

$$\phi ::= P(t) \mid t = t' \mid \neg \phi \mid \phi \to \phi \mid \forall x \phi \mid AX \phi \mid A\phi U \phi \mid E\phi U \phi \mid K_i \phi$$

We want to check FO-CTLK properties, e.g.: the manufacturer M knows that each WO has to match a corresponding PO:

$$AG \forall po id (\exists id, p, s WO (id, po id, p, s)) \to K_M \exists p, o, s PO (po id, p, o, s))$$

Difficulty: the infinite domain U gives raise to infinitely many states!

Investigated solution: can we simulate the concrete values with a finite set of abstract symbols?
Verification Formalism: FO-CTLK

How to specify system properties?

**Definition (Syntax of FO-CTLK)**

\[ \varphi ::= P(t) \mid t = t' \mid \neg \varphi \mid \varphi \to \varphi \mid \forall x \varphi \mid AX \varphi \mid A\varphi U \varphi \mid E\varphi U \varphi \mid Ki \varphi \]

We want to check FO-CTLK properties, e.g.:

- the manufacturer M knows that each WO has to match a corresponding PO:

\[
AG \forall po\_id(\exists id, p, s \ WO(id, po\_id, p, s) \to KM \exists p, o, s \ PO(po\_id, p, o, s))
\]
Verification Formalism: FO-CTLK

How to specify system properties?

**Definition (Syntax of FO-CTLK)**

\[ \varphi ::= P(t) \mid t = t' \mid \neg \varphi \mid \varphi \to \varphi \mid \forall x \varphi \mid AX \varphi \mid A\varphi U \varphi \mid E \varphi U \varphi \mid K_i \varphi \]

We want to check FO-CTLK properties, e.g.:

- the manufacturer M knows that each WO has to match a corresponding PO:

\[ AG \forall po_{-id}(\exists id, p, s \ WO(id, po_{-id}, p, s) \to K_M \exists p, o, s \ PO(po_{-id}, p, o, s)) \]

**Difficulty**: the infinite domain \( U \) gives raise to infinitely many states!

**Investigated solution**: can we *simulate* the concrete values with a finite set of *abstract* symbols?
A run $r$ is an infinite sequence $D^0 \to D^1 \to \ldots$ of states; $r(i) = D^i$.

**Definition (Semantics of FO-CTLK)**

\[
\begin{align*}
(P, D, \sigma) \models \varphi & \iff (D, \sigma) \models \varphi, \text{ if } \varphi \text{ is an FO-formula} \\
(P, D, \sigma) \models \neg \varphi & \iff (P, D, \sigma) \not\models \varphi \\
(P, D, \sigma) \models \varphi \rightarrow \psi & \iff (P, D, \sigma) \not\models \varphi \text{ or } (P, D, \sigma) \models \psi \\
(P, D, \sigma) \models \forall x \varphi & \iff \text{for all } u \in \text{adom}(D), (P, D, \sigma^x_u) \models \varphi \\
(P, D, \sigma) \models AX \varphi & \iff \text{for all runs } r, \text{ if } r(0) = D \text{ then } (P, r(1), \sigma) \models \varphi \\
(P, D, \sigma) \models A\varphi U \psi & \iff \text{for all runs } r, \text{ if } r(0) = D \text{ then there is } k \geq 0 \text{ s.t. } (P, r(k), \sigma) \models \psi, \text{ and for all } j, 0 \leq j < k \text{ implies } (P, r(j), \sigma) \models \varphi \\
(P, D, \sigma) \models E\varphi U \psi & \iff \text{for some run } r, \text{ if } r(0) = D \text{ and there is } k \geq 0 \text{ s.t. } (P, r(k), \sigma) \models \psi, \text{ and for all } j, 0 \leq j < k \text{ implies } (P, r(j), \sigma) \models \varphi \\
(P, D, \sigma) \models Ki \varphi & \iff \text{for all } D', D \sim_i D' \text{ implies } (P, D', \sigma) \models \varphi
\end{align*}
\]

A formula $\varphi$ is *true* in $D$, or $(P, D) \models \varphi$, if $(P, D, \sigma) \models \varphi$ for all $\sigma$.

A formula $\varphi$ is *true* in $P$, or $P \models \varphi$, if $(P, D_0) \models \varphi$. 
FO-CTL Semantics

Intuition

AXφ:

AφUψ:

EφUψ:
Verification of AC-MAS

The General Problem

- **Model Checking for AC-MAS**: 
  
  Given $\mathcal{P}$ and $\varphi$, does $(\mathcal{P}, D_0, \sigma) \models \varphi$ for some $\sigma$?
Verification of AC-MAS

The General Problem

- **Model Checking for AC-MAS:**
  
  Given $\mathcal{P}$ and $\varphi$, does $(\mathcal{P}, D_0, \sigma) \models \varphi$ for some $\sigma$?

- Similar to Model Checking but technically more challenging:
  - Relational states
  - Infinite state-space
Verification of AC-MAS

The General Problem

- **Model Checking for AC-MAS:**
  
  \[ \text{Given } P \text{ and } \varphi, \text{ does } (P, D_0, \sigma) \models \varphi \text{ for some } \sigma? \]

- Similar to Model Checking but technically more challenging:
  - Relational states
  - Infinite state-space

**Theorem**

*The MC problem for AC-MAS is undecidable.*

- BUT decidable over finite interpretation domains:
  - by reduction to standard propositional case (*propositionalise* FO facts).
Verification of Bounded AC-MAS

- Here we devise a notable case of decidability
Verification of Bounded AC-MAS

- Here we devise a notable case of decidability
- If all $\mathcal{D}$-instances of the AC-MAS are bounded, then, though infinite-state, model-checking is decidable.

**Definition (b-bounded (Artifact) System)**

Given a bound $b \in \mathbb{N}$ s.t. $b \geq |\text{adom}(D_0)|$, an AC-MAS $\mathcal{P}$ is $b$-bounded if for every $D \in \mathcal{P}$, $|\text{adom}(D)| \leq b$. 

Practical approach: verify implementation, rather than design. 
Idea: actual machines have bounded memory.
Here we devise a notable case of decidability.

If all $D$-instances of the AC-MAS are bounded, then, though infinite-state, model-checking is decidable.

**Definition ($b$-bounded (Artifact) System)**

Given a bound $b \in \mathbb{N}$ s.t. $b \geq |adom(D_0)|$, an AC-MAS $\mathcal{P}$ is $b$-bounded if for every $D \in \mathcal{P}$, $|adom(D)| \leq b$.

Practical approach: verify implementation, rather than design.
Here we devise a notable case of decidability
If all $\mathcal{D}$-instances of the AC-MAS are bounded, then, though infinite-state, model-checking is decidable.

**Definition ($b$-bounded (Artifact) System)**

Given a bound $b \in \mathbb{N}$ s.t. $b \geq |adom(D_0)|$, an AC-MAS $\mathcal{P}$ is $b$-bounded if for every $D \in \mathcal{P}$, $|adom(D)| \leq b$.

- Practical approach: verify implementation, rather than design.
- Idea: actual machines have bounded memory.
Verification of Bounded AC-MAS

As a consequence of the domain $U$ being infinite, we still have:

- infinite branching;
- infinite state-space.

E.g., with at most 2 tuples:

QUESTION:

- Can we model-check a bounded system?
  - Non-trivial! we cannot *construct* the (infinite) model.
The concrete AC-MAS is abstracted by replacing the infinite interpretation domain $\mathbb{N}$ with a finite one ($\{a, b, c, d, e, f, g, h\}$).
The cardinality of the new domain $U'$ depends on

- the (memory) bound $b$
- the AC-MAS $\mathcal{P}$
- the specification $\varphi$ to check
The cardinality of the new domain $U'$ depends on
- the (memory) bound $b$
- the AC-MAS $\mathcal{P}$
- the specification $\varphi$ to check

The resulting finite-state system can be model-checked by standard techniques
Abstraction

Results

- The cardinality of the new domain $U'$ depends on:
  - the (memory) bound $b$
  - the AC-MAS $\mathcal{P}$
  - the specification $\varphi$ to check

- The resulting finite-state system can be model-checked by standard techniques

- BUT how did we get rid of an infinite number of elements and transitions?
Abstraction

Results

- The cardinality of the new domain $U'$ depends on
  - the (memory) bound $b$
  - the AC-MAS $\mathcal{P}$
  - the specification $\varphi$ to check

- The resulting finite-state system can be model-checked by standard techniques

- BUT how did we get rid of an infinite number of elements and transitions?

- We apply an abstraction process based on two formal notions:
  1. Isomorphism between $\mathcal{D}$-instances;
  2. Bisimulation between AC-MAS.
Definition (Isomorphism)

Two $D$-instances $D$ and $D'$ are $C$-isomorphic, or $D \simeq_C D'$, iff there is a bijection $\iota : \text{adom}(D) \cup C \leftrightarrow \text{adom}(D') \cup C$ s.t.

(i) $\iota$ is the identity on $C$

(ii) for every $\bar{u} \in U^*$, $\bar{u} \in D(P_i)$ iff $\iota(\bar{u}) \in D'(P_i)$

In words: instances obtained by uniformly renaming the elements not in $C$.

E.g., for $C = \{1\}$, $\iota(1) = 1$, $\iota(2) = a$, $\iota(3) = b$, $\iota(4) = c$. 

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{D}$</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>b</td>
</tr>
</tbody>
</table>
Isomorphic instances have a notable (well-known) property:

**Lemma**

If $D \simeq D'$ then for every FO-formula $\varphi$ s.t. $\text{con}(\varphi) \subseteq C$,

$$D \models \varphi \iff D' \models \varphi$$

- The *coloured instance* satisfies $\varphi$ iff all the instances isomorphic to it do
- The *coloured instance* stands for infinitely many isomorphic instances (*isomorphism type*): same values iff same colours
- Observation: for a given bound $b$, there are only finitely many isomorphism types
Data Abstraction

Bisimilar AC-MAS

Definition (Bisimilarity)

Two AC-MAS $\mathcal{P}_1$ and $\mathcal{P}_2$ are **C-bisimilar**, or $\mathcal{P}_1 \approx_C \mathcal{P}_2$, iff there exists a \textit{bisimulation relation} $\approx_C$ s.t. $D_{10} \approx_C D_{20}$, and if $D_1 \approx_C D_2$ then

(i) $D_1 \approx_C D_2$

(ii) if $D_1 \rightarrow D_1'$ then there is $D_2'$ s.t. $D_2 \rightarrow D_2'$ and $D_1' \approx_C D_2'$

(iii) if $D_2 \rightarrow D_2'$ then there is $D_1'$ s.t. $D_1 \rightarrow D_1'$ and $D_1' \approx_C D_2'$

(iv) Similarly, (ii) and (iii) hold for the epistemic relation $\sim_i$ for every agent $i$
Definition (Bisimilarity)

Two AC-MAS $\mathcal{P}_1$ and $\mathcal{P}_2$ are **C-bisimilar**, or $\mathcal{P}_1 \approx_C \mathcal{P}_2$, iff there exists a *bisimulation relation* $\approx_C$ s.t. $D_{10} \approx_C D_{20}$, and if $D_1 \approx_C D_2$ then

(i) $D_1 \approx_C D_2$

(ii) if $D_1 \rightarrow D'_1$ then there is $D'_2$ s.t. $D_2 \rightarrow D'_2$ and $D'_1 \approx_C D'_2$

(iii) if $D_2 \rightarrow D'_2$ then there is $D'_1$ s.t. $D_1 \rightarrow D'_1$ and $D'_1 \approx_C D'_2$

(iv) Similarly, (ii) and (iii) hold for the epistemic relation $\sim_i$ for every agent $i$

Intuitively, the following diagrams commute:

\[
\begin{array}{ccc}
D_1 & \rightarrow & D'_1 \\
\downarrow & & \downarrow \\
D_2 & \rightarrow & D'_2
\end{array}
\quad
\begin{array}{ccc}
D_1 & \sim_i & D'_1 \\
\downarrow & & \downarrow \\
D_2 & \sim_i & D'_2
\end{array}
\]

However, bisimulation alone is not sufficient to preserve FO-CTLK formulas.
## Uniform AC-MAS

### Definition (Uniformity)

An AC-MAS $\mathcal{P}$ is **C-uniform** iff for $D, D', D'' \in S$ and $D''' \in \mathcal{D}(U)$:

1. $D \rightarrow D'$ and $D \oplus D' \simeq_c D'' \oplus D'''$ imply $D'' \rightarrow D'''$;
2. $D \sim_i D'$ and $D \oplus D' \simeq_c D'' \oplus D'''$ imply $D'' \sim_i D'''$.

Intuitively, the behaviour of uniform AC-MAS is independent from data not explicitly named.

Further, $\mathcal{P}(a) \rightarrow \mathcal{P}(b)$

Hence, $\mathcal{P}(c) \rightarrow \mathcal{P}(d)$

Uniform AC-MAS cover a vast number of interesting cases.
An AC-MAS \( \mathcal{P} \) is \textbf{C-uniform} iff for \( D, D', D'' \in S \) and \( D''' \in \mathcal{D}(U) \):

1. \( D \rightarrow D' \) and \( D \oplus D' \simeq_C D'' \oplus D''' \) imply \( D'' \rightarrow D''' \);
2. \( D \sim_i D' \) and \( D \oplus D' \simeq_C D'' \oplus D''' \) imply \( D'' \sim_i D''' \).

Intuitively, the behaviour of uniform AC-MAS is independent from data not explicitly named.

- Suppose that \( P(a) \rightarrow P(b) \)
- Further, \( P(a) \oplus P'(b) \simeq P(c) \oplus P'(d) \)
- Hence, \( P(c) \rightarrow P(d) \)
Definition (Uniformity)

An AC-MAS \( \mathcal{P} \) is **C-uniform** iff for \( D, D', D'' \in S \) and \( D''' \in \mathcal{D}(U) \):

1. \( D \rightarrow D' \) and \( D \oplus D' \cong_C D'' \oplus D''' \) imply \( D'' \rightarrow D''' \);

2. \( D \sim_i D' \) and \( D \oplus D' \cong_C D'' \oplus D''' \) imply \( D'' \sim_i D''' \).

Intuitively, the behaviour of uniform AC-MAS is independent from data not explicitly named.

- Suppose that \( P(a) \rightarrow P(b) \)
- Further, \( P(a) \oplus P'(b) \cong P(c) \oplus P'(d) \)
- Hence, \( P(c) \rightarrow P(d) \)

Uniform AC-MAS cover a vast number of interesting cases.
Bisimulation Results

Bisimilarity together with uniformity is sufficient to preserve FO-CTLK formulas.

**Theorem**

Consider two bisimilar uniform AC-MAS $P_1$ and $P_2$, and an FO-CTLK formula $\varphi$. If

1. $|U_2| \geq \max_{D \in P_1} |\text{adom}(D)| + |C| + |\text{var}(\varphi)|$
2. $|U_1| \geq \max_{D \in P_2} |\text{adom}(D)| + |C| + |\text{var}(\varphi)|$

then

$$P_1 \models \varphi \iff P_2 \models \varphi$$
We verify the actual, bounded implementations of AC-MAS.
Bounded Models and Finite Abstractions

We verify the actual, bounded implementations of AC-MAS.

Consider

- an AC-MAS $\mathcal{P}_1$ on a domain $U_1$ s.t.
  1. $U_1$ is infinite
  2. $\mathcal{P}_1$ is $b$-bounded, i.e., for all $D \in \mathcal{P}_1$, $|\text{adom}(D)| \leq b$
We verify the actual, bounded implementations of AC-MAS. Consider

- an AC-MAS $\mathcal{P}_1$ on a domain $U_1$ s.t.
  1. $U_1$ is infinite
  2. $\mathcal{P}_1$ is $b$-bounded, i.e., for all $D \in \mathcal{P}_1$, $|\text{dom}(D)| \leq b$
- an FO-CTLK formula $\varphi$. 

In particular, $\mathcal{P}_1 \models \varphi$ iff $\mathcal{P}_2 \models \varphi$. 

$\frac{59}{78}$
Bounded Models and Finite Abstractions

We verify the actual, bounded implementations of AC-MAS.

Consider

- an AC-MAS $\mathcal{P}_1$ on a domain $U_1$ s.t.
  1. $U_1$ is infinite
  2. $\mathcal{P}_1$ is $b$-bounded, i.e., for all $D \in \mathcal{P}_1$, $|adom(D)| \leq b$
- an FO-CTLK formula $\varphi$.

Then, there exists a finite abstraction $\mathcal{P}_2$ of $\mathcal{P}_1$ s.t.

- $\mathcal{P}_2$ is uniform and bisimilar to $\mathcal{P}_1$
- $|U_2| \geq 2b + |C| + |var(\varphi)|$
Bounded Models and Finite Abstractions

We verify the actual, bounded implementations of AC-MAS.

Consider

- an AC-MAS \( \mathcal{P}_1 \) on a domain \( U_1 \) s.t.
  1. \( U_1 \) is infinite
  2. \( \mathcal{P}_1 \) is \( b \)-bounded, i.e., for all \( D \in \mathcal{P}_1 \), \( |adom(D)| \leq b \)
- an FO-CTLK formula \( \varphi \).

Then, there exists a finite abstraction \( \mathcal{P}_2 \) of \( \mathcal{P}_1 \) s.t.

  1. \( \mathcal{P}_2 \) is uniform and bisimilar to \( \mathcal{P}_1 \)
  2. \( |U_2| \geq 2b + |C| + |\text{var}(\varphi)| \)

In particular,

\[
\mathcal{P}_1 \models \varphi \iff \mathcal{P}_2 \models \varphi
\]
**Problem:** the result in the previous slide assumes that $\mathcal{P}_1$ is given and then builds $\mathcal{P}_2$. 
Problem: the result in the previous slide assumes that $P_1$ is given and then builds $P_2$.

BUT $P_1$ is infinite, so we may not be able to construct $P_2$. 
Problem: the result in the previous slide assumes that $\mathcal{P}_1$ is given and then builds $\mathcal{P}_2$.

BUT $\mathcal{P}_1$ is infinite, so we may not be able to construct $\mathcal{P}_2$.

We need a methodology to obtain the abstract $\mathcal{P}_2$ without going through the concrete $\mathcal{P}_1$. 
Problem: the result in the previous slide assumes that $\mathcal{P}_1$ is given and then builds $\mathcal{P}_2$.

BUT $\mathcal{P}_1$ is infinite, so we may not be able to construct $\mathcal{P}_2$.

We need a methodology to obtain the abstract $\mathcal{P}_2$ without going through the concrete $\mathcal{P}_1$.

To do so, we need to specify the form of actions.
Verification of Artifact System Programs

We give an example of uniform AC-MAS consistent with GSM [HDD+11].

For each agent $i$ we define $\text{Act}_i$ as the set of local (parametric) actions of the form $\omega(\vec{x}) \doteq \langle \pi(\vec{y}), \psi(\vec{z}) \rangle$ s.t.

- $\omega(\vec{x})$ is the operation signature and $\vec{x} = \vec{y} \cup \vec{z}$ is the set of operation parameters
- $\pi(\vec{y})$ is the operation precondition, i.e., an FO-formula over $\mathcal{D}_i$
- $\psi(\vec{z})$ is the operation postcondition, i.e., an FO-formula over $\mathcal{D} \cup \mathcal{D}'$

We call the AC-MAS specified in this way Artifact System Programs.
Now, $D \rightarrow D'$ iff for some $\alpha(\vec{x}) \in \text{Act}$ there is an execution $\alpha(\vec{u}) = \langle \pi(\vec{v}), \psi(\vec{w}) \rangle$ and

- $\text{adom}(D') \subseteq \text{adom}(D) \cup \vec{w} \cup \text{con}(\psi)$
- $D \models \pi(\vec{v})$, i.e., the action is enabled
- $D \oplus D' \models \psi(\vec{w})$
Example 2: the Order-to-Cash Scenario

Specification of actions affecting the MO in the order-to-cash scenario:

- \( createMO(id, wo_id, cost) = \langle \pi(wo_id, cost), \psi(id, wo_id, cost) \rangle \) where:
  
  - \( \pi(wo_id, cost) \equiv \)
    
    \( \exists po_id, p(WO(wo_id, po_id, p, completed) \wedge \exists pr, o \ (PO(po_id, pr, o, pending) \wedge Materials(pr, cost))) \wedge \phi_{b-1} \)

  - \( \psi(id, wo_id, cost) \equiv \)
    
    \( MO'(id, wo_id, cost, preparation) \wedge \forall id', w, c, s \ (MO(id', w, c, s) \rightarrow id \neq id') \wedge \phi_{b} \)

where \( \phi_{k} \ := \ \forall x_1, \ldots, x_{k+1} \ \vee_{i \neq j}(x_i = x_j) \) says that there are at most \( k \) objects in the active domain.

The specification of \( createMO \) guarantees that the bound \( b \) is not violated by action execution.
Finite Abstraction of AS Programs

- AS programs are uniform.
Finite Abstraction of AS Programs

- AS programs are uniform.
- If
  - the AS program $P_{\text{Act}, U_1}$ is $b$-bounded
  - the finite domain $U_2$ is s.t. $|U_2| \geq 2b + |C_{AS}| + N_{AS}$
then
  - the induced AS program $P_{\text{Act}, U_2}$ is a finite abstraction of $P_{\text{Act}, U_1}$

**Lemma**

If $D \simeq_C \hat{D}$ then every concrete transition $D \rightarrow D'$ has an abstract counterpart $\hat{D} \rightarrow \hat{D}'$ s.t. $D' \simeq_C \hat{D}'$.
Given a concrete execution $\alpha(\vec{u}) = \langle \pi(\vec{v}), \psi(\vec{w}) \rangle$, there exist $\vec{v}, \vec{w}, \hat{D}$ s.t.

(i) $\hat{D} \models \pi(\vec{v})$

(ii) $\hat{D} \oplus \hat{D}' \models \psi(\vec{w})$

(iii) $D' \sim_C \hat{D}'$

- there exist $\hat{D}'$ and $\vec{u}$, and a $C$-isomorphism between

\[
\{ D, D', \vec{u} \} \text{ and } \{ \hat{D}, \hat{D}', \vec{u} \}
\]

- This is enough, as $\pi$ and $\varphi$ are invariant w.r.t. $C$-isomorphic instances.
Finite Abstraction of AS Programs

If-Part (Intuition) Cont.

How to define an $C$-isomorphism between $\{D, D', \vec{u}\}$ and $\{\hat{D}, \hat{D}', \hat{\vec{u}}\}$:

1. obtain $\hat{\vec{u}}$ by renaming the elements in $\vec{u}$ according to $\iota$, $k$, and preserving (in)equalities – $\hat{U}$ contains enough elements to do so;

2. obtain $\hat{D}'$ by renaming the elements in $D'$ according to $\iota$ and $j$. 
Verification of Artifact System Programs

If

- the AS program $P_{Act,U_1}$ is $b$-bounded
- the finite domain $U_2$ is s.t. $|U_2| \geq 2b + |C_{AS}| + N_{AS}$,

then

- the induced AS program $P_{Act,U_2}$ is a finite abstraction of $P_{Act,U_1}$. 
Verification of Artifact System Programs

- If
  - the AS program $\mathcal{P}_{\text{Act}, U_1}$ is $b$-bounded
  - the finite domain $U_2$ is s.t. $|U_2| \geq 2b + |C_{AS}| + N_{AS}$,
  then
    - the induced AS program $\mathcal{P}_{\text{Act}, U_2}$ is a finite abstraction of $\mathcal{P}_{\text{Act}, U_1}$.

- In particular, if $|U_2| \geq 2b + |C_{AS}| + \max\{N_{AS}, |\text{var}(\varphi)|\}$, then
  $$\mathcal{P}_{\text{Act}, U_1} \models \varphi \quad \text{iff} \quad \mathcal{P}_{\text{Act}, U_2} \models \varphi$$
Application to the General Case

Preservation Theorem

- What if $P$ is unbounded? (apart from undecidability)
Application to the General Case

Preservation Theorem

- What if $\mathcal{P}$ is unbounded? (apart from undecidability)

Observation: for fixed $b \in \mathbb{N}$, the $b$-abstraction $\hat{\mathcal{P}}_b$ corresponds to an (infinite) fragment of $\mathcal{P}$.

Preservation theorem for the existential fragment $\text{FO}^\exists\text{-ECTLK}$.

\[ \varphi ::= \phi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \exists x \varphi \mid \text{EX} \varphi \mid E \varphi \cup \varphi \mid \overline{K}_i \varphi \]

Theorem

Given an AS program $\mathcal{P}$, $b \geq |\text{dom}(D_0)|$, and an $\text{FO}^\exists\text{-ECTLK}$ formula $\varphi$,

\[ \hat{\mathcal{P}}_b \models \varphi \Rightarrow \mathcal{P} \models \varphi \]

Observe we can iterate on $b$. 
To conclude

Results...

- We are able to model check AC-MAS wrt full FO-CTLK...
- ...however, our results hold only for *uniform* systems.
- This class includes many interesting systems (AS programs).
To conclude

Results...

- We are able to model check AC-MAS wrt full FO-CTLK...
- ...however, our results hold only for *uniform* systems.
- This class includes many interesting systems (AS programs).

... and Future Work

- Techniques for finite abstraction.
- Abstraction techniques for finite-state systems are effective on the abstract system?
- How to perform the boundedness check.
D. Cohn and R. Hull.

Reasoning About Knowledge.

Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events.
To appear.