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

Imperial College London

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

Imperial College London

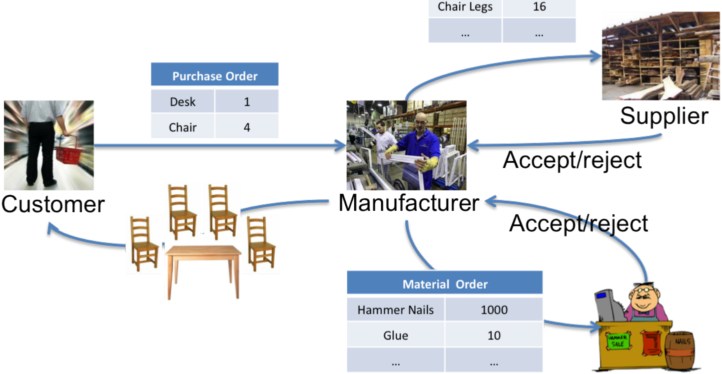
Recent paradigm for Business Process modeling and development [CH09]

- *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

Imperial College London



Purchase Order	
Desk	1
Chair	4

Material Order	
Desk Legs	4
Chair Legs	16
...	...

Material Order	
Hammer Nails	1000
Glue	10
...	...

Imperial College London

PO			
id	prod_code	offer	status

- $createPO(prod\_code)$
- $deletePO(id)$
- $addLinePO(id, prod\_code, offer)$
- ...

WO			
id	po_id	price	status

- $createWO(po\_id)$
- $deleteWO(id)$
- $addLineWO(id, po\_id, price)$
- ...

MO			
id	wo_id	cost	status

- $createMO(wo\_id)$
- $deleteMO(id)$
- $addLineMO(id, wo\_id, status)$
- ...

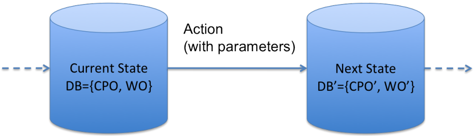
Products	
prod_code	budget

Materials	
mat_code	cost

Artifact Systems

Lifecycle

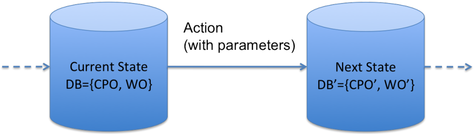
- As the process goes on, artifact actions are executed.
  - e.g., the Purchase Order is sent to the Manufacturer.



Artifact Systems

Lifecycle

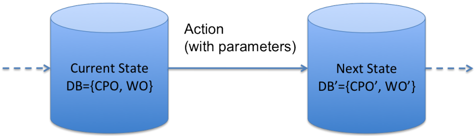
- As the process goes on, artifact actions are executed.
  - e.g., the Purchase Order is sent to the Manufacturer.
- Actions add/remove artifacts or change artifact attributes.
  - e.g., the PO status changes from *created* to *submitted*.



Artifact Systems

Lifecycle

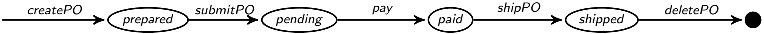
- As the process goes on, artifact actions are executed.
  - e.g., the Purchase Order is sent to the Manufacturer.
- Actions add/remove artifacts or change artifact attributes.
  - e.g., the PO status changes from *created* to *submitted*.
- The whole system can be seen as a *data-aware* dynamic system.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



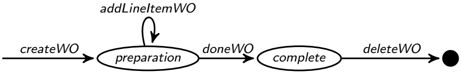
Artifact Systems

Lifecycle

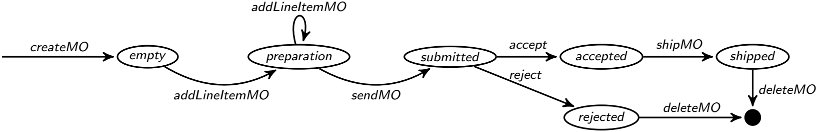
We can give a (partial) representation of AS as FSM.



(a) Purchase Order lifecyle



(b) Work Order Lifecyle



(c) Material Order lifecyle

Databases and Artifact Systems

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

Databases and Artifact Systems

Imperial College London

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
  $\mathcal{D} = \{Products/2, PO/4, WO/4, Materials/2, MO/4\}$

Databases and Artifact Systems

Imperial College London

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

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

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

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.

PO			
id	*prod_code*	*offer*	*status*
1	#12	\$50	prepared
2	#24	\$120	shipped
4	#45	\$80	paid
⋮	⋮	⋮	⋮

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
  $\mathcal{D} = \{Products/2, PO/4, WO/4, Materials/2, MO/4\}$

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.

PO			
id	*prod_code*	*offer*	*status*
1	#12	\$50	prepared
2	#24	\$120	shipped
4	#45	\$80	paid
⋮	⋮	⋮	⋮

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

Given

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

we inductively define satisfaction:

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

Notice that we adopt an *active domain semantics*.

Given

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

we inductively define satisfaction:

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

Notice that we adopt an *active domain semantics*.

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.

Artifact-centric Multi-agent Systems

Imperial College London

- 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 $i = \langle \mathcal{D}_i, L_i, Act_i, Pr_i \rangle$ where:
  - $\mathcal{D}_i$ is the *local database schema*
  - $L_i \subseteq \mathcal{D}_i(U)$ 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*

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 $i = \langle \mathcal{D}_i, L_i, Act_i, Pr_i \rangle$ where:
  - $\mathcal{D}_i$ is the *local database schema*
  - $L_i \subseteq \mathcal{D}_i(U)$ 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 $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n$.

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 $i = \langle \mathcal{D}_i, L_i, Act_i, Pr_i \rangle$ where:
  - $\mathcal{D}_i$ is the *local database schema*
  - $L_i \subseteq \mathcal{D}_i(U)$ 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 $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n$.
- Agents manipulate artifacts and have (partial) access to the information contained therein.

Example 1: the Order-to-Cash Scenario

- Agents: Customer, Manifacturer, Supplier.

Example 1: the Order-to-Cash Scenario

Imperial College London

- Agents: Customer, Manifacturer, 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, Manifacturer, Supplier.
- Local database schemas:
  - $\mathcal{D}_C = \{Products, PO\}$
  - $\mathcal{D}_M = \{WO\}$
  - $\mathcal{D}_S = \{Materials, MO\}$
- Then $\mathcal{D} = \{Products, PO, WO, Materials, MO\}$.

Example 1: the Order-to-Cash Scenario

Imperial College London

- Agents: Customer, Manifacturer, Supplier.
- Local database schemas:
  - $\mathcal{D}_C = \{Products, PO\}$
  - $\mathcal{D}_M = \{WO\}$
  - $\mathcal{D}_S = \{Materials, MO\}$
- Then $\mathcal{D} = \{Products, PO, WO, Materials, MO\}$.
- Parametric actions can introduce values from an infinite domain U:
  - $createPO(id, prod\_code, offer)$ in Act_C
  - $createWO(id, po\_id, price)$ in Act_M
  - $createMO(id, wo\_id, cost)$ in Act_S

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

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

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

Artifact-centric Multi-agent Systems

Imperial College London

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

- An *AC-MAS* is a tuple $\mathcal{P} = \langle \mathcal{S}, U, D_0, \tau \rangle$ where
  - $\mathcal{S} \subseteq L_1 \times \cdots \times L_n$ is the set of *reachable global states*
  - U is the interpretation domain
  - $D_0 \in \mathcal{S}$ is the *initial global state*
  - $\tau : \mathcal{S} \times Act \mapsto 2^{\mathcal{S}}$ is the *global transition function*
- Temporal transition: $D \rightarrow D'$ iff there is α s.t. $\tau(D, \alpha(\vec{u})) = D'$.
- 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.

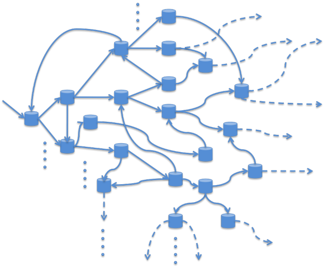
- An *AC-MAS* is a tuple $\mathcal{P} = \langle \mathcal{S}, U, D_0, \tau \rangle$ where
  - $\mathcal{S} \subseteq L_1 \times \cdots \times L_n$ is the set of *reachable global states*
  - U is the interpretation domain
  - $D_0 \in \mathcal{S}$ is the *initial global state*
  - $\tau : \mathcal{S}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

Imperial College London

- 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

Imperial College London

How to specify system properties?

Definition (Syntax of FO-CTLK)

$$\varphi ::= P(\vec{t}) \mid t = t' \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi \mid AX\varphi \mid A\varphi U\varphi \mid E\varphi U\varphi \mid K_i\varphi$$

Verification Formalism: FO-CTLK

How to specify system properties?

Definition (Syntax of FO-CTLK)

$$\varphi ::= P(\vec{t}) \mid t = t' \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi \mid AX\varphi \mid A\varphi U \varphi \mid E\varphi U \varphi \mid K_i \varphi$$

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

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

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

Verification Formalism: FO-CTLK

How to specify system properties?

Definition (Syntax of FO-CTLK)

$$\varphi ::= P(\vec{t}) \mid t = t' \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi \mid AX\varphi \mid A\varphi U\varphi \mid E\varphi U\varphi \mid K_i\varphi$$

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

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

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

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

Semantics

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)

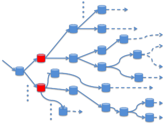
$(\mathcal{P}, D, \sigma) \models \varphi$	iff	$(D, \sigma) \models \varphi$, if φ is an FO-formula
$(\mathcal{P}, D, \sigma) \models \neg\varphi$	iff	$(\mathcal{P}, D, \sigma) \not\models \varphi$
$(\mathcal{P}, D, \sigma) \models \varphi \to \psi$	iff	$(\mathcal{P}, D, \sigma) \not\models \varphi$ or $(\mathcal{P}, D, \sigma) \models \psi$
$(\mathcal{P}, D, \sigma) \models \forall x \varphi$	iff	for all $u \in adom(D)$, $(\mathcal{P}, D, \sigma^x_u) \models \varphi$
$(\mathcal{P}, D, \sigma) \models AX\varphi$	iff	for all runs r, if $r(0) = D$ then $(\mathcal{P}, r(1), \sigma) \models \varphi$
$(\mathcal{P}, D, \sigma) \models A\varphi\mathcal{U}\psi$	iff	for all runs r, if $r(0) = D$ then there is $k \geq 0$ s.t. $(\mathcal{P}, r(k), \sigma) \models \psi$, and for all j, $0 \leq j < k$ implies $(\mathcal{P}, r(j), \sigma) \models \varphi$
$(\mathcal{P}, D, \sigma) \models E\varphi\mathcal{U}\psi$	iff	for some run r, $r(0) = D$ and there is $k \geq 0$ s.t. $(\mathcal{P}, r(k), \sigma) \models \psi$, and for all j, $0 \leq j < k$ implies $(\mathcal{P}, r(j), \sigma) \models \varphi$
$(\mathcal{P}, D, \sigma) \models K_i\varphi$	iff	for all D', $D \sim_i D'$ implies $(\mathcal{P}, D', \sigma) \models \varphi$

A formula φ is *true* in D, or $(\mathcal{P}, D) \models \varphi$, if $(\mathcal{P}, D, \sigma) \models \varphi$ for all σ.
A formula φ is *true* in $\mathcal{P}$, or $\mathcal{P} \models \varphi$, if $(\mathcal{P}, D_0) \models \varphi$.

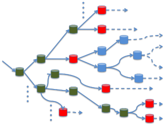
FO-CTL Semantics

Intuition

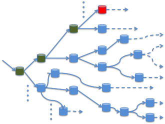
$AX\varphi$:



$A\varphi\mathcal{U}\psi$:



$E\varphi\mathcal{U}\psi$:



Imperial College London

- *Model Checking for AC-MAS*:

  Given $\mathcal{P}$ and φ, does $(\mathcal{P}, D_0, \sigma) \models \varphi$ for some σ?

Verification of AC-MAS

The General Problem

- *Model Checking for AC-MAS*:

  Given $\mathcal{P}$ and φ, does $(\mathcal{P}, D_0, \sigma) \models \varphi$ for some σ?

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

- *Model Checking for AC-MAS*:

  Given $\mathcal{P}$ and φ, does $(\mathcal{P}, D_0, \sigma) \models \varphi$ for some σ?

- 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

Imperial College London

- 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. $b \geq |adom(D_0)|$, an AC-MAS $\mathcal{P}$ is *b-bounded* if for every $D \in \mathcal{P}$, $|adom(D)| \leq b$.

Verification of Bounded AC-MAS

Imperial College London

- 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.

Verification of Bounded AC-MAS

Imperial College London

- 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 $D \in \mathcal{P}$, $|adom(D)| \leq b$.

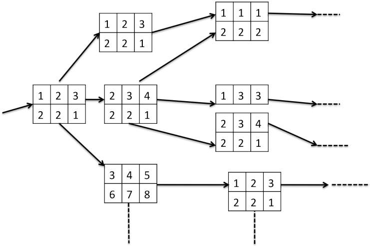
- Practical approach: verify implementation, rather than design.
- Idea: actual machines have bounded memory.

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:



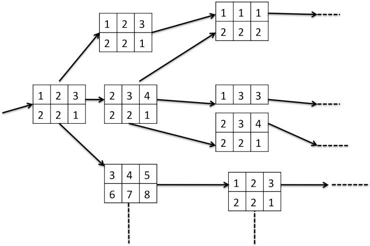
QUESTION:

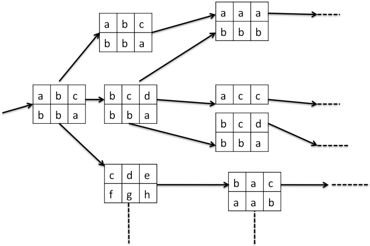
☞ Can we model-check a bounded system?

- Non-trivial! we cannot *construct* the (infinite) model.

Abstraction

Idea





- The concrete AC-MAS is abstracted by *replacing* the infinite interpretation domain $\mathbb{N}$ with a finite one ($\{a, b, c, d, e, f, g, h\}$).

Abstraction

Results

Imperial College London

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

Abstraction

Results

- The cardinality of the new domain U' depends on
  - the (memory) bound b
  - the AC-MAS $\mathcal{P}$
  - the specification φ to check
- The resulting finite-state system can be model-checked by standard techniques

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

Data Abstraction

Isomorphic instances

Imperial College London

Definition (Isomorphism)

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

(i) ι is the identity on C

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

In words: instances obtained by uniformly renaming the elements not in C.
E.g., for $C = \{1\}$, $\iota(1) = 1$, $\iota(2) = a$, $\iota(3) = b$, $\iota(4) = c$.

D

1	2	3
2	4	3

$\hat{D}$

1	a	b
a	c	b

Data Abstraction

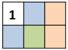
Isomorphic instances (cont.)

Isomorphic instances have a notable (well-known) property:

Lemma

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

$$D \models \varphi \Leftrightarrow D' \models \varphi$$



- The *coloured instance* satisfies φ 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

Imperial College London

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) $D_1 \simeq_C D_2$

(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

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 *bisimulation relation* $\approx_C$ s.t. $D_{10} \approx_C D_{20}$, and if $D_1 \approx_C D_2$ then

(i) $D_1 \simeq_C D_2$

(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:

$$\begin{array}{ccc} D_1 & \longrightarrow & D_1' \\ \wr & & \wr \\ D_2 & \longrightarrow & D_2' \end{array} \qquad \begin{array}{ccc} D_1 & \sim_i & D_1' \\ \wr & & \wr \\ D_2 & \sim_i & D_2' \end{array}$$

However, bisimulation alone is not sufficient to preserve FO-CTLK formulas.

Uniform AC-MAS

Imperial College London

Definition (Uniformity)

An AC-MAS $\mathcal{P}$ is *C-uniform* iff for $D, D', D'' \in \mathcal{S}$ 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 $D'' \sim_i D'''$.

Uniform AC-MAS

Imperial College London

Definition (Uniformity)

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

1. $D \to D'$ and $D \oplus D' \simeq_C D'' \oplus D'''$ imply $D'' \to D'''$;
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 $P(a) \to P(b)$
  - Further, $P(a) \oplus P'(b) \simeq P(c) \oplus P'(d)$
  - Hence, $P(c) \to P(d)$

Uniform AC-MAS

Imperial College London

Definition (Uniformity)

An AC-MAS $\mathcal{P}$ is *C-uniform* iff for $D, D', D'' \in \mathcal{S}$ 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, $P(c) \rightarrow P(d)$
- Uniform AC-MAS cover a vast number of interesting cases.

Bisimulation Results

Imperial College London

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

Theorem

Consider two bisimilar uniform *AC-MAS* $\mathcal{P}_1$ *and* $\mathcal{P}_2$, *and an FO-CTLK formula* φ. *If*

1. $|U_2| \geq \max_{D \in \mathcal{P}_1} |adom(D)| + |C| + |var(\varphi)|$
2. $|U_1| \geq \max_{D \in \mathcal{P}_2} |adom(D)| + |C| + |var(\varphi)|$

then

$$\mathcal{P}_1 \models \varphi \quad \textit{iff} \quad \mathcal{P}_2 \models \varphi$$

Bounded Models and Finite Abstractions

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 $D \in \mathcal{P}_1$, $|adom(D)| \leq b$

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 φ.

Bounded Models and Finite Abstractions

Imperial College London

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 φ.
- 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. $|U_2| \geq 2b + |C| + |var(\varphi)|$

Bounded Models and Finite Abstractions

Imperial College London

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 φ.
- 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. $|U_2| \geq 2b + |C| + |var(\varphi)|$
- In particular,

$$\mathcal{P}_1 \models \varphi \quad \text{iff} \quad \mathcal{P}_2 \models \varphi$$

- Problem: the result in the previous slide assumes that $\mathcal{P}_1$ is given and then builds $\mathcal{P}_2$.

Bounded Models and Finite Abstractions

- 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 $\mathcal{P}_2$.

Bounded Models and Finite Abstractions

- 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 $\mathcal{P}_1$.

Bounded Models and Finite Abstractions

- 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

Imperial College London

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

For each agent i we define Act_i as the set of 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*.

Artifact Systems: Semantics

Now, $D \rightarrow D'$ iff for some $\alpha(\vec{x}) \in Act$ there is an execution $\alpha(\vec{u}) = \langle \pi(\vec{v}), \psi(\vec{w}) \rangle$ and

- $adom(D') \subseteq adom(D) \cup \vec{w} \cup con(\psi)$
- $D \models \pi(\vec{v})$, i.e., the action is *enabled*
- $D \oplus D' \models \psi(\vec{w})$

Example 2: the Order-to-Cash Scenario

Imperial College London

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:
  - $\pi(wo\_id, cost) \equiv$
    $\exists po\_id, p(WO(wo\_id, po\_id, p, \text{completed}) \wedge$
    $\exists pr, o\ (PO(po\_id, pr, o, \text{pending}) \wedge$
    $Materials(pr, cost))) \wedge$
    ϕ_{b-1}
  - $\psi(id, wo\_id, cost) \equiv$
    $MO'(id, wo\_id, cost, \text{preparation}) \wedge$
    $\forall id', w, c, s\ (MO(id', w, c, s) \rightarrow id \neq id')) \wedge$
    ϕ_b

where $\phi_k ::= \forall x_1, \ldots, x_{k+1} \bigvee_{i \neq j}(x_i = x_j)$ 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

Imperial College London

- AS programs are uniform.

Finite Abstraction of AS Programs

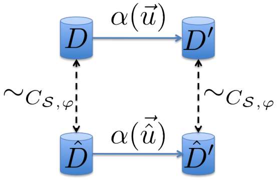
Imperial College London

- AS programs are uniform.
- If
  - the AS program $\mathcal{P}_{Act,U_1}$ is b-bounded
  - the finite domain U_2 is s.t. $|U_2| \geq 2b + |C_{AS}| + N_{AS}$

  then
  - the induced AS program $\mathcal{P}_{Act,U_2}$ is a finite abstraction of $\mathcal{P}_{Act,U_1}$

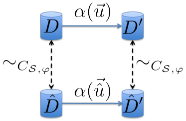
Lemma

If $D \simeq_C \hat{D}$ then every concrete transition $D \rightarrow D'$ has an abstract counterpart $\hat{D} \rightarrow \hat{D}'$ s.t. $D' \simeq_C \hat{D}'$.



Finite Abstraction of AS Programs

If-Part (Intuition)



Given a concrete execution $\alpha(\vec{u}) = \langle \pi(\vec{v}), \psi(\vec{w}) \rangle$, there exist $\hat{\vec{v}}, \hat{\vec{w}}, \hat{D}'$ s.t.

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

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

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

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

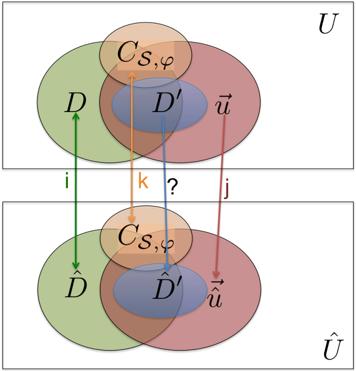
$$\{D, D', \vec{u}\} \text{ and } \{\hat{D}, \hat{D}', \hat{\vec{u}}\}$$

- This is enough, as π and φ 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 $\{\hat{D}, \hat{D}', \vec{\hat{u}}\}$:



1. obtain $\vec{\hat{u}}$ by renaming the elements in $\vec{u}$ according to ι, 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 ι and j.

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

  then
  - the induced AS program $\mathcal{P}_{Act,U_2}$ is a finite abstraction of $\mathcal{P}_{Act,U_1}$.

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

  then
  - the induced AS program $\mathcal{P}_{Act,U_2}$ is a finite abstraction of $\mathcal{P}_{Act,U_1}$.
- In particular, if $|U_2| \geq 2b + |C_{AS}| + \max\{N_{AS}, |var(\varphi)|\}$, then

$$\mathcal{P}_{Act,U_1} \models \varphi \quad \text{iff} \quad \mathcal{P}_{Act,U_2} \models \varphi$$

Application to the General Case

Preservation Theorem

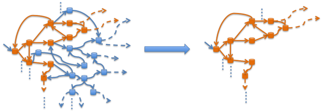
- What if $\mathcal{P}$ is unbounded? (apart from undecidability)

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* FO∃-ECTLK.

$$\varphi ::= \phi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x \varphi \mid EX\varphi \mid E\varphi\mathcal{U}\varphi \mid \overline{K}_i\varphi$$

Theorem

Given an AS program $\mathcal{P}$, $b \geq |adom(D_0)|$, and an FO∃-ECTLK formula φ,

$$\hat{\mathcal{P}}_b \models \varphi \quad \Rightarrow \quad \mathcal{P} \models \varphi$$

Observe we can iterate on b.

To conclude

Imperial College London

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 perfom the boundedness check.

Bibliography

Imperial College London

D. Cohn and R. Hull.
Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes.
IEEE Data Eng. Bull., 32(3):3–9, 2009.

R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi.
Reasoning About Knowledge.
The MIT Press, 1995.

Rick Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno (Therry) Heath III, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and Roman Vaculín.
Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events.
In *Proc. of DEBS*, 2011.
To appear.