Verification of Data-Centric Dynamic Systems with External Services

Diego Calvanese

Joint work with: B. Bagheri Hariri, G. De Giacomo, A. Deutsch, M. Montali

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Foundations and Challenges of Change in Ontologies and Databases
29–39/1/2014 – Bolzano, Italy

Outline

1. Motivations
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Verification of Data-Aware Processes
5. Incomplete Information in the Data Layer
6. References

Outline

1 Motivations

2 Artifact-Centric Approach

3 Data-Centric Dynamic Systems

4 Verification of Data-Aware Processes

5 Incomplete Information in the Data Layer

6 References

Why Formal Verification?

Errors in computerized systems can be **costly**.

Pentium chip (1994)

Bug found in FPU. Intel offers to replace faulty chips. Estimated loss: 475M US$

Ariane 5 (1996)

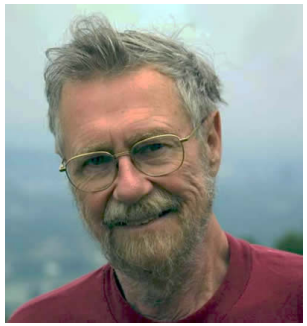
Esploded 37secs after launch. Cause: uncaught overflow exception.

Toyota Prius (2010)

Software "glitch" found in anti-lock braking system.
185,000 cars recalled.

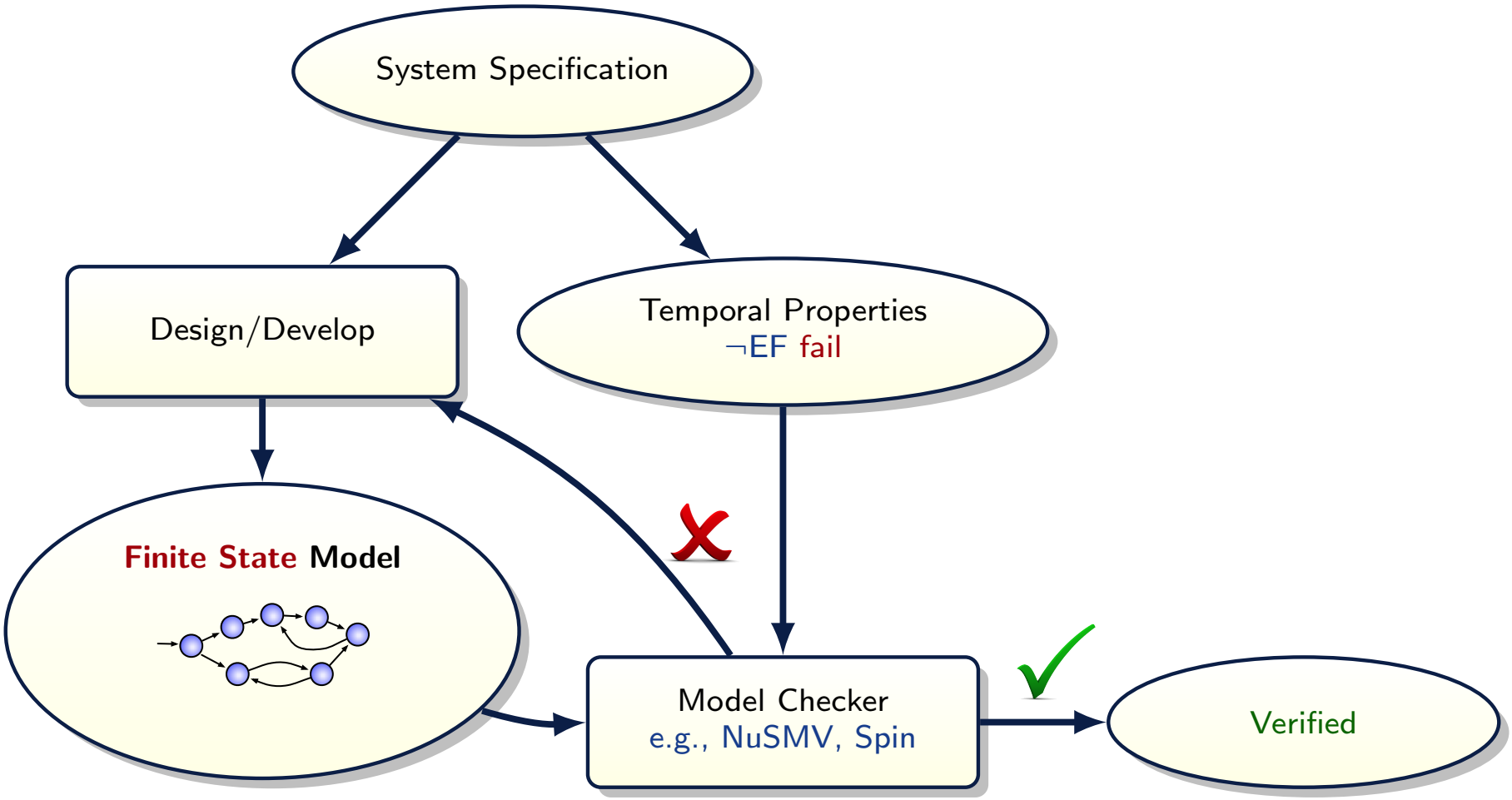
Why verify?

"Testing can only show the presence of errors, not their absence." [Edgar Dijkstra]



unibz.it

Model Checking Cycle

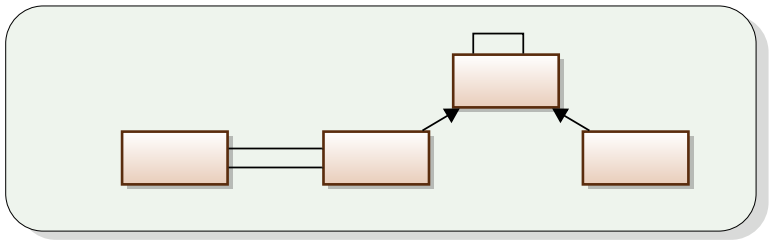


The finite state requirement is severe and restrictive
Especially for settings that capture data and dynamics simultaneously (e.g., Artifact-Centric Business Process Systems).

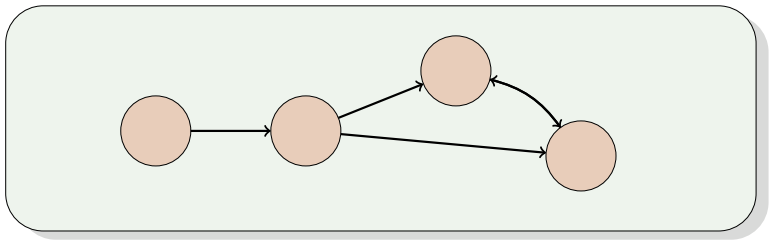
Traditional Process Modeling

A "**divide et impera**" approach, to attack the complexity of the domain of interest:

- Structural modeling: conceptual models, domain ontologies, DB schemas
  - UML, ORM, ER, ...



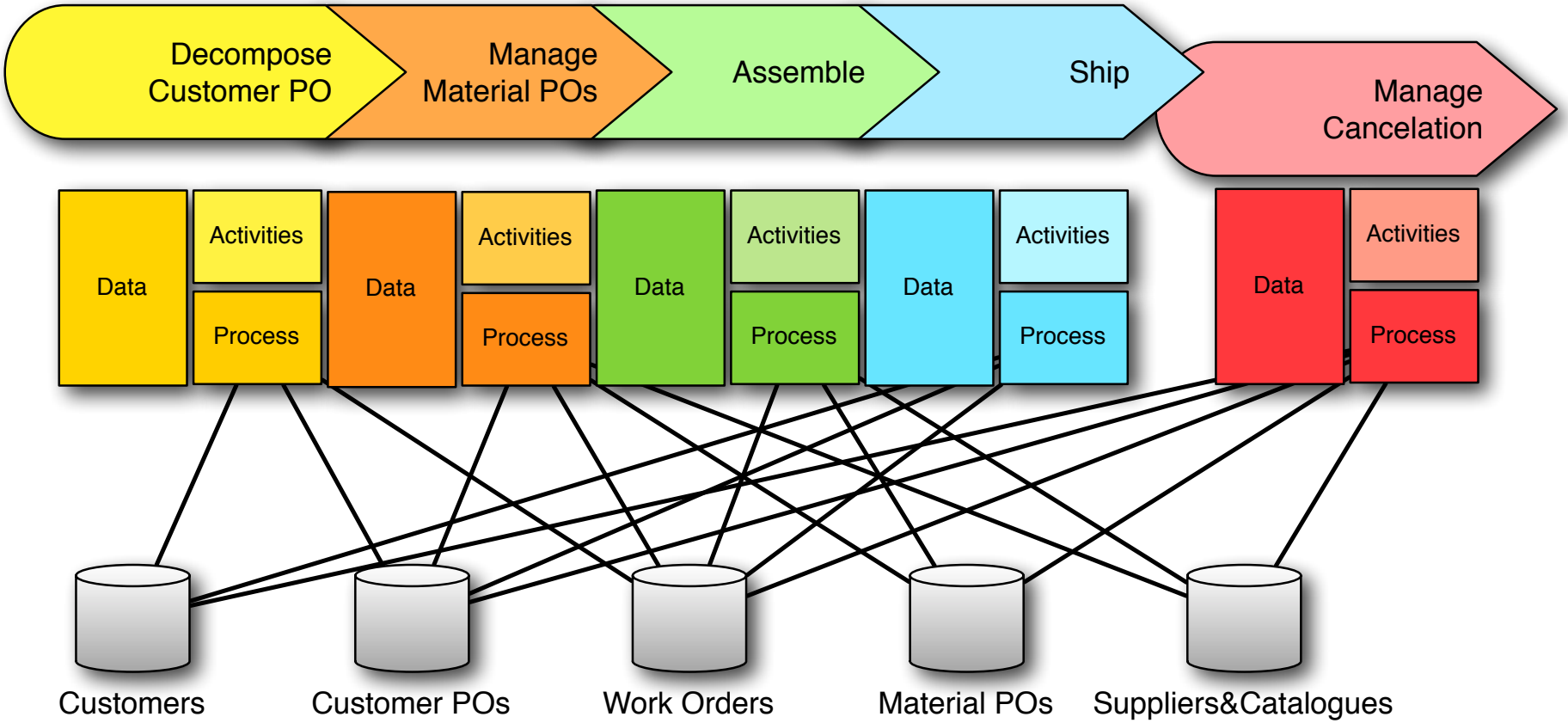
- Behavioral modeling: activities, services, business processes
  - BPMN, UML, BPEL, SOA-related technologies, ...



Drawback: lack of a coherent holistic view:

- The two models are loosely connected.
- The full combined behavior is never captured.

Spaghetti Layer



Outline

1 Motivations

2 Artifact-Centric Approach

3 Data-Centric Dynamic Systems

4 Verification of Data-Aware Processes

5 Incomplete Information in the Data Layer

6 References

Business Artifacts to the Rescue

- In early 2000, the **artifact-centric approach** emerged as a foundational proposal for merging data and processes together.
  - The emphasis is on data, which are modeled taking into account that they will be manipulated by processes.
  - Processes are modeled by considering how they manipulate data.
- Initial proposals by IBM [Nigam and Caswell, 2003], followed by [Bhattacharya *et al.*, 2007; Deutsch *et al.*, 2007], ...
- See also EU project ACSI (for **Artifact-Centric Service Interoperation**), 2010–2013.

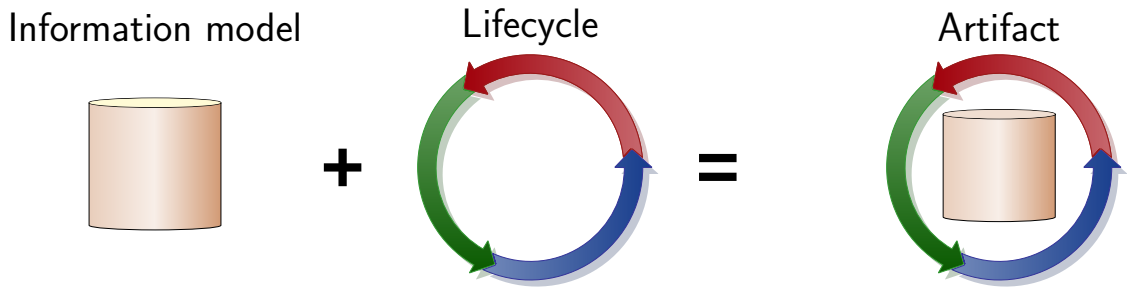
What is an Artifact?

Definition

A key, business-relevant conceptual dynamic entity that is used in guiding the operation of a business.

Consists of:

- **Information model** - relevant data maintained by the artifact
- **Lifecycle model** - (implicit) description of the evolutions of the information model that are allowed through the execution of a process.

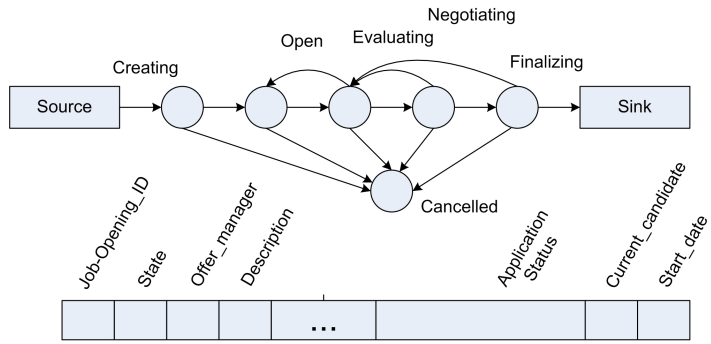


Goal: unified, end-to-end view of relevant entities and their possible evolutions.

Concrete Models for Artifacts

Key questions:

- How and where to store data maintained by their information models?
- How to specify the lifecycle of such artifacts?
- At which level of abstraction?



Some concrete information models:

- **Relational database** (with nested records).
- **Knowledge base**, e.g., expressed in a Description Logic.

Some concrete lifecycle models:

- **Finite-state machines**. State = phase; events trigger transitions.
  - Implemented in the *Siena* prototype by IBM.
- **Guard-Stage-Milestone** lifecycles, based on declarative (event-condition-action)-like rules.
  - Implemented in the *Barcelona* prototype by IBM.
- **Proclets** (interacting Petri nets).
  - Emphasise many-to-many relationships between artifacts.

Reasoning about Artifacts as Dynamic Entities

We want to provide a formal foundation for artifact-centric systems, and provide corresponding reasoning facilities for their trustworthy design.

In particular, we want to decide whether dynamic/temporal properties of interest hold over the life of such systems.

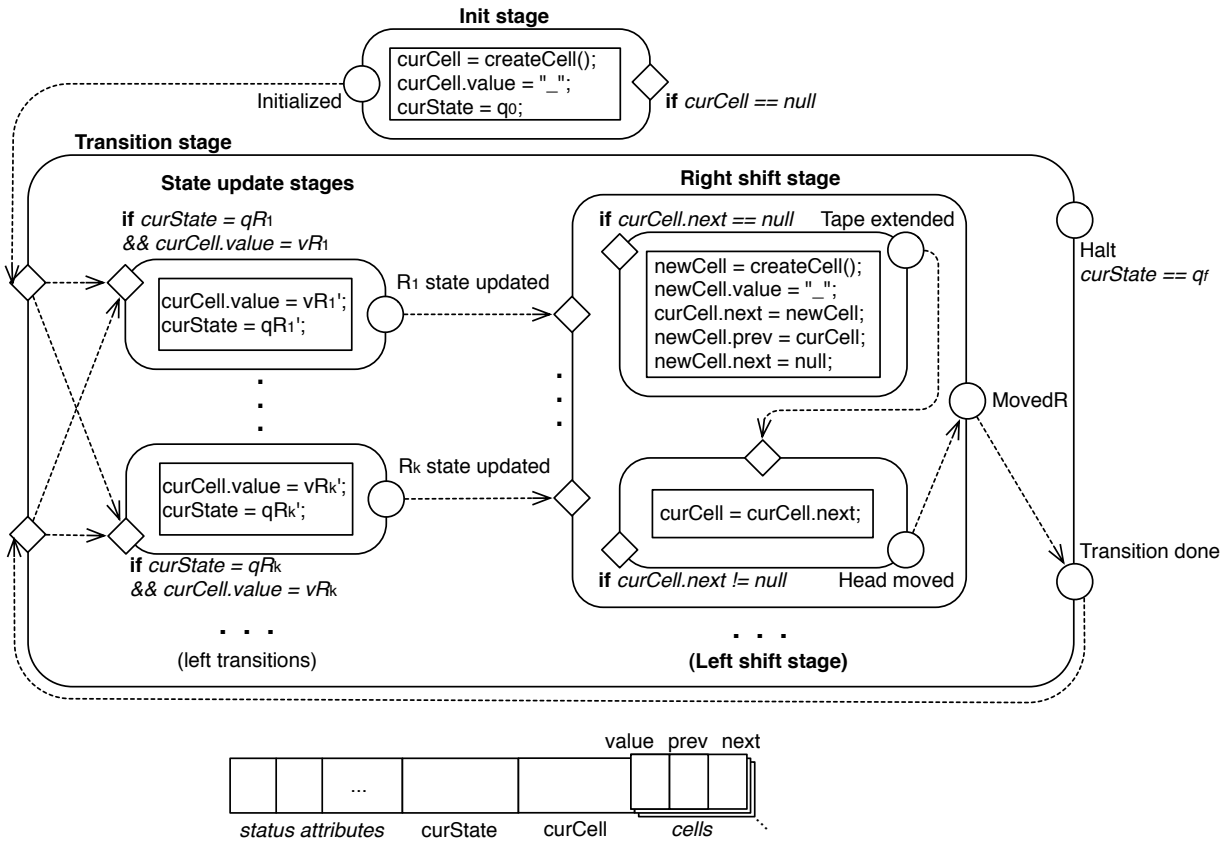
- **Verification** of temporal formulae.
- **Dominance/simulation/bisimulation/containment** properties.
- Automated **composition** of artifacts-based systems.
- Automated **process synthesis** from dynamic/temporal specifications.

Currently (2010's) the scientific community is quite good at each of these, but only in a **finite setting**!
However, artifacts pose two challenging problems:

- the presence of data makes them **infinite-state** systems;
- properties need to accommodate temporal operators *and* queries over the artifact information models $\rightarrow$ **first-oder temporal formulae**.

Verification of Artifacts is Tough

What is a non-artifact example of a finite-state control process manipulating possibly unbounded data? **A Turing Machine**



Verification of the propositional CTL / LTL reachability property "eventually milestone *Halt* achieved" is **undecidable**.

Artifact Formal Foundations

Is there hope to find interesting decidable cases?

- This requires to identify "classes of systems" that enjoy verifiability.
- First step: devise a minimal, clean mathematical framework as the basis of investigation.
- Many approaches in the literature:
  - University California San Diego, University California Santa Barbara, IBM Watson, Imperial College, Sapienza Università di Roma, Free University of Bozen-Bolzano.
  - Starting from previous work, we have defined a rich but "pristine" formal framework: **Data-Centric Dynamic Systems**.

Note: approaches based on many-dimensional modal logics (one dimension for data, one dimension for process) are not suitable.

- Undecidability holds for weak domain models, when rigid relations are allowed.
- No hope to isolate an interesting class.

Outline

1 Motivations

2 Artifact-Centric Approach

3 Data-Centric Dynamic Systems

4 Verification of Data-Aware Processes

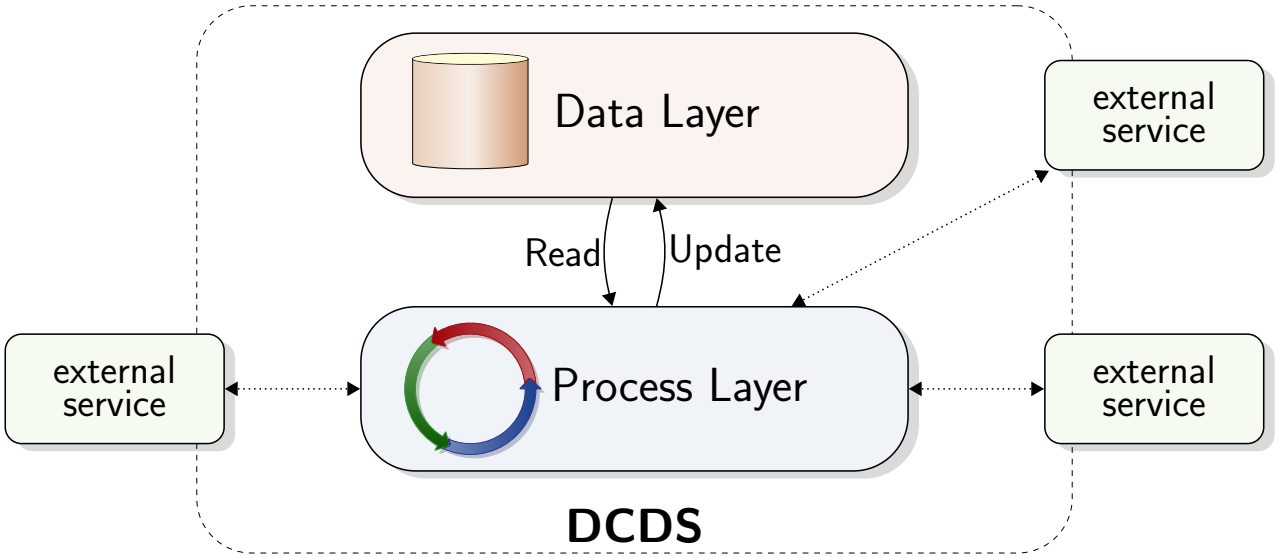
5 Incomplete Information in the Data Layer

6 References

Data-Centric Dynamic Systems (DCDSs)

An abstract, pristine framework to formally describe processes that manipulate data.

- Subsumes virtually all existing approaches for modeling data-centric processes, such as the artifact-centric paradigm.



- **Data layer**: models and maintains the data of interest.
- **Process layer**: manages the system dynamics and how the execution of actions impact on the data layer.
  - Calls **external services** to interact with the environment.

Components of a Relational DCDS

Data Layer

- **Relational schema** with **constraints** (domain-dependent FO formulas).
- **Initial DB** (instance) conforming to the schema+constraints.
- The initial DB is evolved by the process layer into new DBs conforming to the schema+constraint.

Process Layer

- Declarative description of the process with **condition-action** rules.
  - Condition: (domain-independent) FO query.
  - Each rule queries the current DB and determines the *executability* of the corresponding action and its parameters.
- **Actions** with parameters, specified in terms of effects that:
  1. query the current DB (with UCQs + domain-independent FO filters);
  2. transfer the answers into facts that become part of the new DB.
- **External services** are called in the action effects to introduce fresh data in the new DB - taken from a *countably infinite domain*.

Deterministic vs. non-deterministic services

We distinguish between two different semantics for service-execution:

Deterministic services semantics

Along the same run, when the same service is invoked again with the same arguments, it returns the same result as in the previous call.

Are used to model an environment whose behavior is completely determined by the parameters.

Example: temperature, given the location and the date and time

Non-deterministic services semantics

Along the same run, when the same service is invoked again with the same arguments, it may return a different value than in the previous call.

Are used to model:

- an environment whose behavior is determined by parameters that are outside the control of the system;
- input of external users, whose choices depend on external factors.

Example: current temperature, given the location

An Example: Hotels and Price Conversion

Data Layer: Info about hotels and room prices

$$\mathsf{Cur} = \langle \mathit{UserCurrency} \rangle \quad \mathsf{CH} = \langle \underline{\mathit{Hotel}}, \mathit{Currency} \rangle \quad \mathsf{PEntry} = \langle \underline{\mathit{Hotel}}, \mathit{Price}, \underline{\mathit{Date}} \rangle$$

Process Layer/1

User selection of a currency.

- Process: $\mathit{true} \longmapsto \mathsf{ChooseCur}()$
- Service call for currency selection: $\textsc{uInputCurr}()$
  - Models *user input* with **non-deterministic** behavior.
- $\mathsf{ChooseCur}() : \left\{ \begin{array}{rcl} \mathit{true} & \leadsto & \mathsf{Cur}(\textsc{uInputCurr}()) \\ \mathsf{CH}(h,c) & \leadsto & \mathsf{CH}(h,c) \\ \mathsf{PEntry}(h,p,d) & \leadsto & \mathsf{PEntry}(h,p,d) \end{array} \right\}$

An Example: Hotels and Price Conversion

Data Layer: Info about hotels and room prices

$$\mathsf{Cur} = \langle \mathit{UserCurrency} \rangle \quad \mathsf{CH} = \langle \underline{\mathit{Hotel}}, \mathit{Currency} \rangle \quad \mathsf{PEntry} = \langle \underline{\mathit{Hotel}}, \mathit{Price}, \underline{\mathit{Date}} \rangle$$

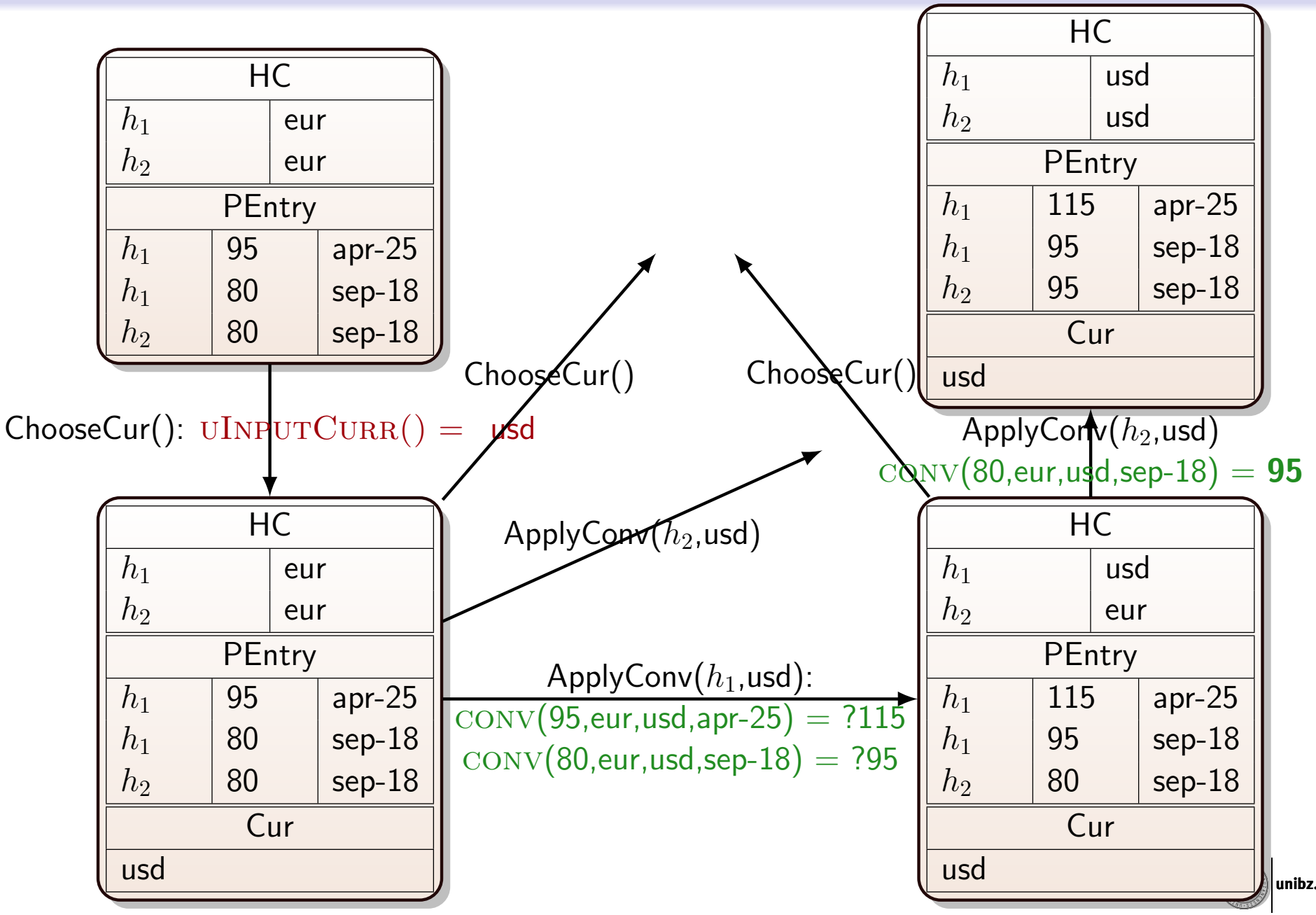
Process Layer/2

Price conversion for a hotel.

- Process: $\mathsf{Cur}(c) \wedge \mathsf{CurHotel}(h, c_h) \wedge c_h \neq c \longmapsto \mathsf{ApplyConv}(h, c)$
- Service call for currency selection: $\textsc{conv}(\mathit{price}, \mathit{from}, \mathit{to}, \mathit{date})$
  - Models historical conversion with deterministic behavior.
- $\mathsf{ApplyConv}(h, c)$:

$$\left\{ \begin{array}{rl} \mathsf{PEntry}(h, p, d) \wedge \mathsf{CH}(h, c_{old}) \wedge \mathsf{Cur}(c) & \rightsquigarrow \mathsf{PEntry}(h, \textsc{conv}(p, c_{old}, c, d), d) \\ \mathsf{PEntry}(h', p, d) \wedge h' \neq h & \rightsquigarrow \mathsf{PEntry}(h', p, d) \\ \mathsf{CH}(h, c_{old}) & \rightsquigarrow \mathsf{CH}(h, c) \\ \mathsf{CH}(h', c') \wedge h' \neq h & \rightsquigarrow \mathsf{CH}(h', c') \\ \mathsf{Cur}(c) & \rightsquigarrow \mathsf{Cur}(c) \end{array} \right\}$$

Run of the System



Execution Semantics

Transition system accounting for all possible runs for the DCDS:

- States: each linked to a DB – instance of the data layer;
- Transitions: *legal* application of action+params+service call evals.
  - Action+params: executable according to the process rules.
  - Deterministic services behave consistently with the previous results.

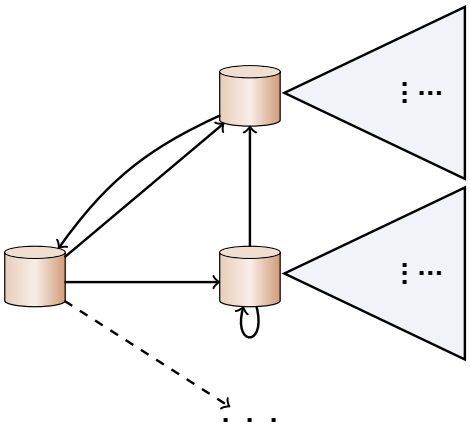
We obtain

- from the initial DB,
- by applying transitions in all possible ways,

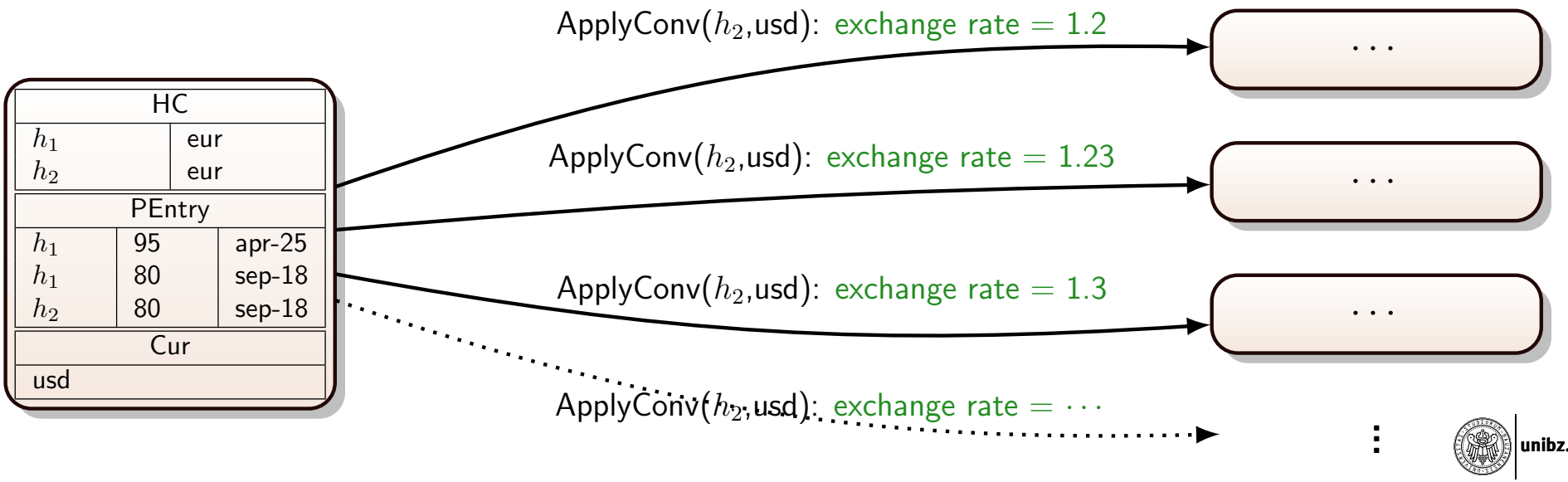
a possibly **infinite-state** (relational) transition system.

Sources of Unboundedness/Infinity

In general: service calls cause . . .



- Infinite branching (due to all possible results of service calls).
- Infinite runs (usage of values obtained from unboundedly many service calls).
- Unbounded DBs (accumulation of such values).



Outline

1 Motivations

2 Artifact-Centric Approach

3 Data-Centric Dynamic Systems

4 Verification of Data-Aware Processes

5 Incomplete Information in the Data Layer

6 References

Verification of DCDSs

Verification

Given a DCDS $\mathcal{S}$ (with transition system $\Upsilon_{\mathcal{S}}$), and a temporal/dynamic property Φ, check whether

$$\Upsilon_{\mathcal{S}} \models \Phi$$

Requirements for temporal/dynamic properties:

- to capture data $\leadsto$ first-order queries;
- to capture dynamics $\leadsto$ temporal modalities;
- to capture evolution of data $\leadsto$ quantification across states.

Our goal

Investigate "robust" conditions on decidability of verification:

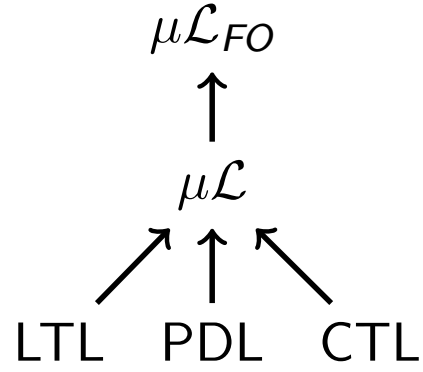
- for sophisticated branching- and linear-time temporal properties;
- exploiting conventional, finite-state model checking via construction of a **faithful** (sound and complete) **finite-state abstraction**.

Problem Design Space

We employ variants of **first-order μ-calculus** ($\mu\mathcal{L}_{FO}$):

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\Phi \mid \langle - \rangle \Phi \mid Z \mid \mu Z.\Phi$$

- Employs fixpoint constructs to express sophisticated properties defined via induction or co-induction.
- Subsumes virtually all logics used in verification, such as LTL, CTL, CTL*.



Problem 1

Unrestricted first-order quantification: no hope of reducing verification to finite-state model checking.

See: $\exists x_1, \ldots, x_n. \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i \in \{1,\ldots,n\}} \langle - \rangle Q(x_i)$

$\rightsquigarrow$ We need to consider fragments of $\mu\mathcal{L}_{FO}$ with controlled quantification.

Problem 2

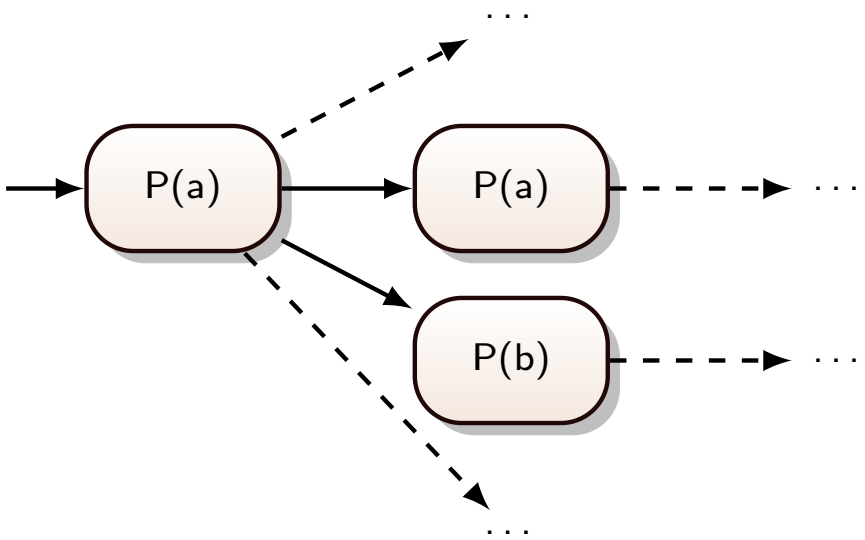
Verification is undecidable for simple propositional CTL $\cap$ LTL properties.

$\rightsquigarrow$ We need to pose restrictions on DCDSs.

Towards decidability

We need to tame the two sources of infinity in DCDSs:

- infinite branching
- infinite runs.



To prove decidability of model checking for a given restriction and verification formalism:

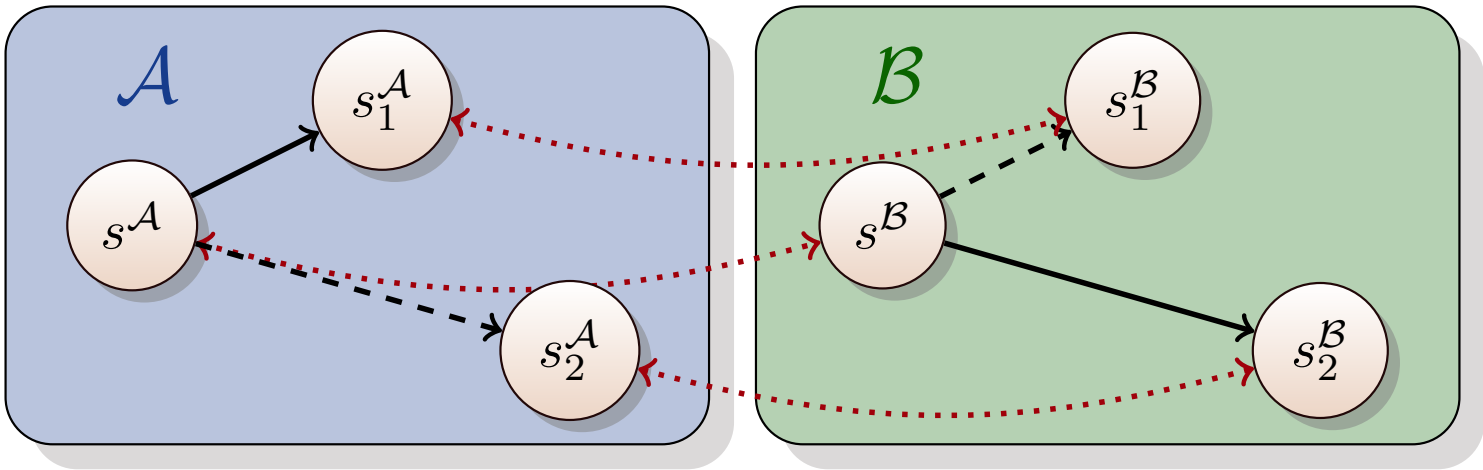
- We use bisimulation as a tool.
- We show that restricted DCDSs have a finite-state bisimilar transition system.

Bisimulation between transition systems

States $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ of transition systems $\mathcal{A}$ and $\mathcal{B}$ are bisimilar if:

1. $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are isomorphic;
2. If there exists a state $s_1^{\mathcal{A}}$ of $\mathcal{A}$ such that $s^{\mathcal{A}} \Rightarrow_{\mathcal{A}} s_1^{\mathcal{A}}$, then there exists a state $s_1^{\mathcal{B}}$ of $\mathcal{B}$ such that $s^{\mathcal{B}} \Rightarrow_{\mathcal{B}} s_1^{\mathcal{B}}$, and $s_1^{\mathcal{A}}$ and $s_1^{\mathcal{B}}$ are bisimilar;
3. The other direction!

$\mathcal{A}$ and $\mathcal{B}$ are bisimilar, if their initial states are bisimilar.



$\mu\mathcal{L}$ invariance property of bisimulation:

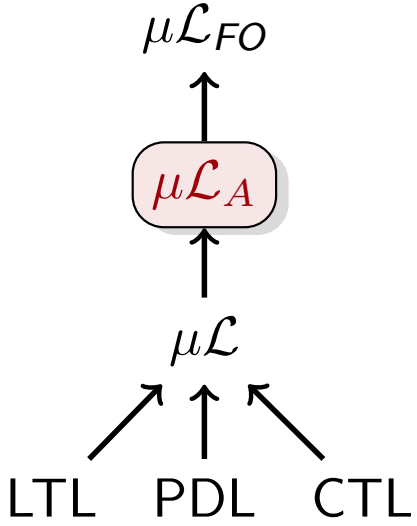
Bisimilar transition systems satisfy the same set of $\mu\mathcal{L}$ properties.

History-Preserving μ-calculus ($\mu\mathcal{L}_A$)

Active-domain quantification: restricted to those individuals *present in the current database*.

$$\exists x.\Phi \qquad \rightsquigarrow \qquad \exists x.\textsc{live}(x) \wedge \Phi$$

where $\textsc{live}(x)$ states that x is present in the current active domain.



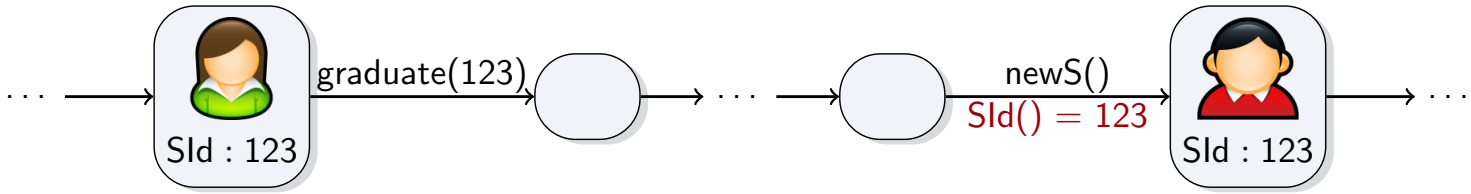
Example

$$\nu X.(\forall x.\textsc{live}(x) \wedge \mathsf{Stud}(x) \rightarrow \mu Y.(\exists y.\textsc{live}(y) \wedge \mathsf{Grad}(x,y) \vee \langle - \rangle Y) \wedge [-]X)$$

Along every path, it is always true, for each student x, that there exists an evolution eventually leading to a graduation of the student (with some final mark y).

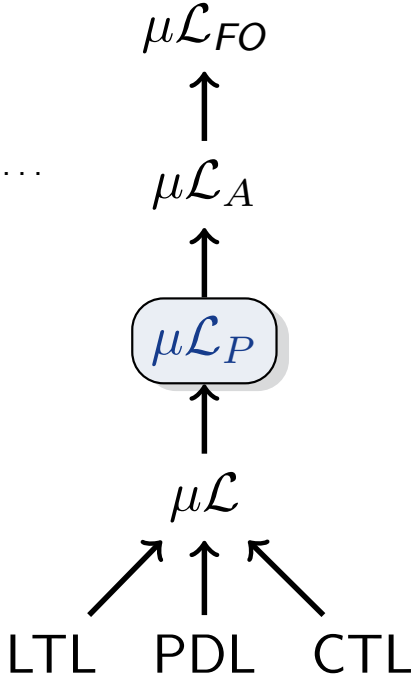
Persistence-Preserving μ-calculus ($\mu\mathcal{L}_P$)

In some cases, objects maintain their identity only if they **persist** in the active domain (cf. business artifacts and their IDs).



$\mu\mathcal{L}_P$ restricts $\mu\mathcal{L}_A$ to **quantification over persisting objects only**, i.e., objects that *continue* to be LIVE.

$$
\begin{array}{rcl}
\exists x.\Phi & \rightsquigarrow & \exists x.\textsc{live}(x) \wedge \Phi \\
\langle - \rangle \Phi(\vec{x}) & \rightsquigarrow & \textsc{live}(\vec{x}) \wedge \langle - \rangle \Phi(\vec{x}) \\
[-]\Phi(\vec{x}) & \rightsquigarrow & \textsc{live}(\vec{x}) \wedge [-]\Phi(\vec{x})
\end{array}
$$



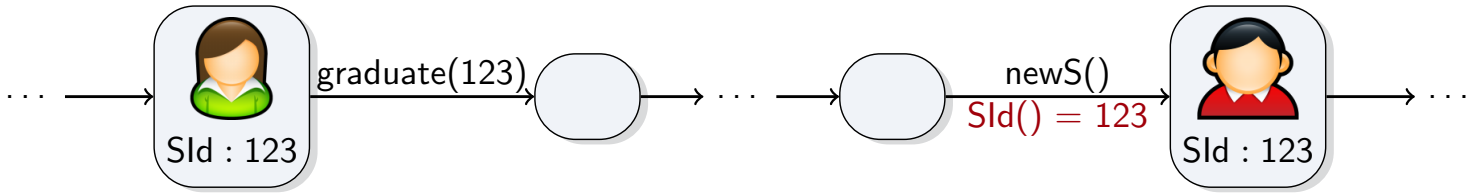
Example ("strong" persistence)

$$\nu X.(\forall x.\textsc{live}(x) \wedge \mathit{Stud}(x) \rightarrow \mu Y.(\exists y.\textsc{live}(y) \wedge \mathit{Grad}(x,y) \vee (\textsc{live}(x) \wedge \langle - \rangle Y)) \wedge [-]X)$$

Along every path, it is always true, for each student x, that there exists an evolution in which x **persists in the database until** she eventually graduates.

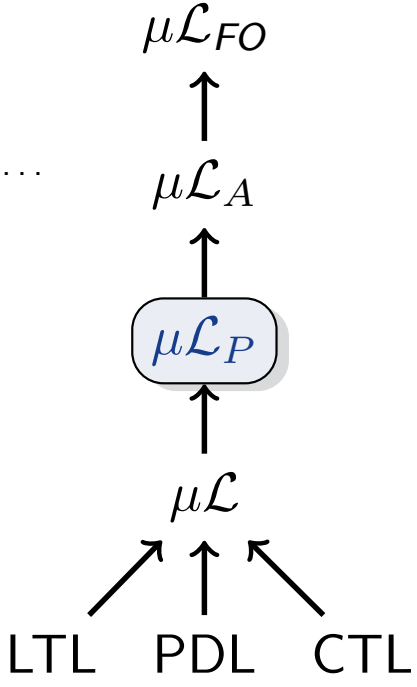
Persistence-Preserving μ-calculus ($\mu\mathcal{L}_P$)

In some cases, objects maintain their identity only if they **persist** in the active domain (cf. business artifacts and their IDs).



$\mu\mathcal{L}_P$ restricts $\mu\mathcal{L}_A$ to **quantification over persisting objects only**, i.e., objects that *continue* to be LIVE.

$$
\begin{array}{rcl}
\exists x.\Phi & \rightsquigarrow & \exists x.\textsc{live}(x) \wedge \Phi \\
\langle - \rangle \Phi(\vec{x}) & \rightsquigarrow & \textsc{live}(\vec{x}) \wedge \langle - \rangle \Phi(\vec{x}) \\
[-]\Phi(\vec{x}) & \rightsquigarrow & \textsc{live}(\vec{x}) \wedge [-]\Phi(\vec{x})
\end{array}
$$



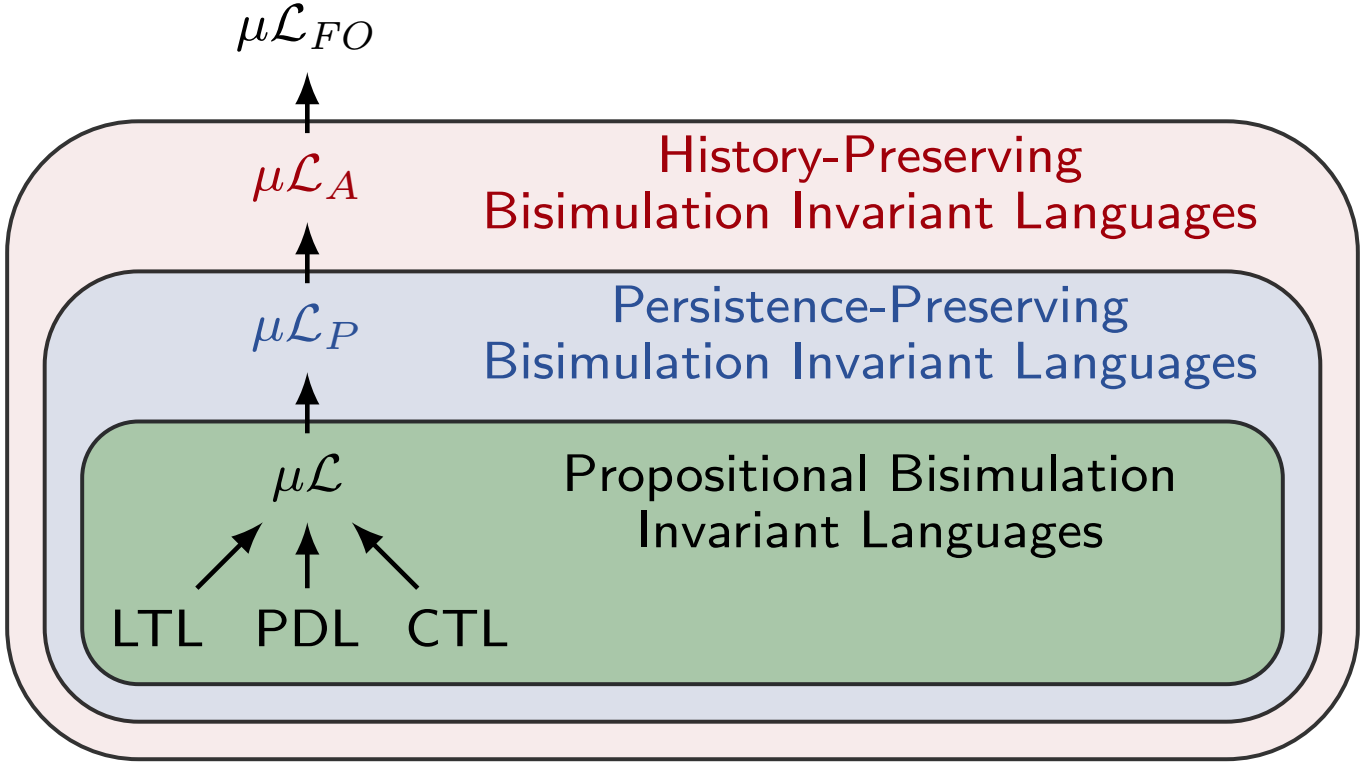
Example ("weak" persistence)

$$\nu X.(\forall x.\textsc{live}(x) \wedge \mathit{Stud}(x) \rightarrow \mu Y.(\exists y.\textsc{live}(y) \wedge \mathit{Grad}(x,y) \vee (\textsc{live}(x) \rightarrow \langle - \rangle Y)) \wedge [-]X)$$

Along every path, it is always true, for each student x, that there exists an evolution in which **either x does not persist, or** she eventually graduates.

Bisimulations

We introduce two novel notions of **bisimulation** to account for $\mu\mathcal{L}_A/\mu\mathcal{L}_P$.



These bisimulation relations capture:

- **dynamics** $\rightsquigarrow$ standard notion of bisimulation;
- **data** $\rightsquigarrow$ DB isomorphism;
- **evolution of data** $\rightsquigarrow$ compatibility of the bijections witnessing the isomorphisms along a run.

Bisimulations

History-preserving bisimulation requires each isomorphism to be witnessed by a bijection that extends the bijection used in the previous step.

Theorem

If Υ_1 and Υ_2 are history-preserving bisimilar, then for every $\mu\mathcal{L}_A$ closed formula Φ, we have:

$$\Upsilon_1 \models \Phi \quad \text{if and only if} \quad \Upsilon_2 \models \Phi$$

Persistence-preserving bisimulation requires each isomorphism to be witnessed by a bijection that extends the bijection used in the previous step, restricted only to the persisting objects.

Theorem

If Υ_1 and Υ_2 are persistence-preserving bisimilar, then for every $\mu\mathcal{L}_P$ closed formula Φ, we have:

$$\Upsilon_1 \models \Phi \quad \text{if and only if} \quad \Upsilon_2 \models \Phi$$

unibz.it

Conditions for DCDSs

We have devised two conditions over the transition system $\Upsilon_{\mathcal{S}}$ of a DCDS $\mathcal{S}$.

Run boundedness

Each **run** of $\Upsilon_{\mathcal{S}}$ accumulate only a **bounded number of objects**.

- No bound on the overall number of objects: $\Upsilon_{\mathcal{S}}$ is still infinite-state, due to infinite branching induced by service calls.
- Unboundedly many deterministic service calls can still be issued with a bounded number of inputs.
- Only boundedly many nondeterministic service calls can be issued.

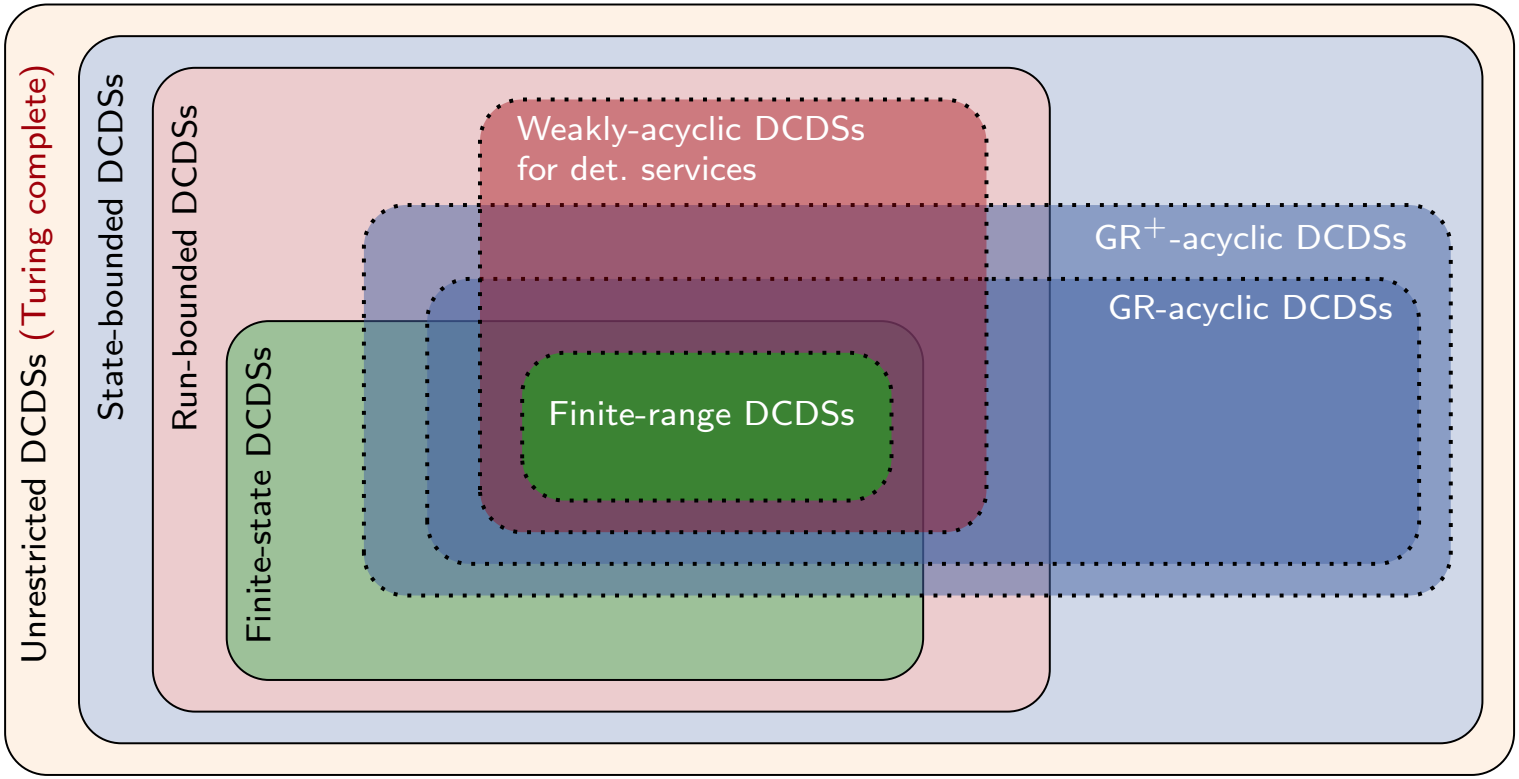
State boundedness

Each **state** of $\Upsilon_{\mathcal{S}}$ contain only a **bounded number of objects**.

- Relaxation of run-boundedness: unboundedly many objects can be encountered along a run, provided that they are not accumulated in the same state.
- $\Upsilon_{\mathcal{S}}$ can contain infinite branches and infinite runs.

unibz.it

Summary of Results on Verification for DCDSs



	Unrestricted	State-bounded	Run-bounded	Finite-state
$\mu\mathcal{L}_{FO}$	**U**	**U**	**N**	**D**
$\mu\mathcal{L}_{A}$	**U**	**U**	**D**	**D**
$\mu\mathcal{L}_{P}$	**U**	**D**	**D**	**D**
$\mu\mathcal{L}$	**U**	**D**	**D**	**D**

D: decidable; **U**: undecidable; **N**: no finite abstraction.

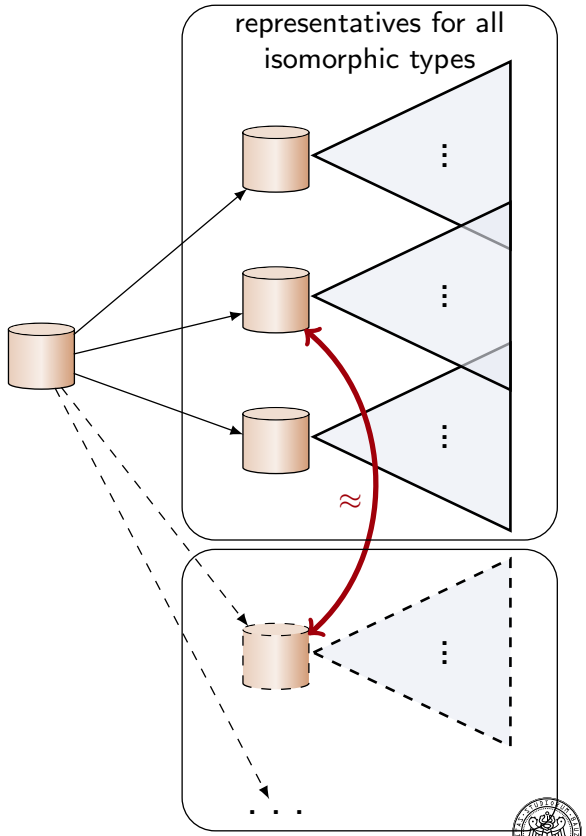
Run-Bounded Systems: Decidability for $\mu\mathcal{L}_A$

Theorem

Verification of $\mu\mathcal{L}_A$ over run-bounded DCDSs is decidable and can be reduced to model checking of propositional $\mu\mathcal{L}$ over a finite TS.

Crux: construct a **faithful abstraction** $\Theta_{\mathcal{S}}$ for $\Upsilon_{\mathcal{S}}$, collapsing infinite branching.

- We use **isomorphic types** instead of actual service call results.



State-bounded Systems: Undecidability for $\mu\mathcal{L}_A$

Theorem

Verification of $\mu\mathcal{L}_A$ over state-bounded DCDSs is undecidable.

Intuition: $\mu\mathcal{L}_A$ can use quantification to store and compare the unboundedly many values encountered along the runs.

Crux: reduction from satisfiability of LTL with freeze quantifiers.

- $\mu\mathcal{L}_A$ can express LTL with freeze quantifier by making registers explicit.
- There is a state-bounded DCDS that simulates all the possible traces with register assignments (i.e., data words).
- Satisfiability via model checking.

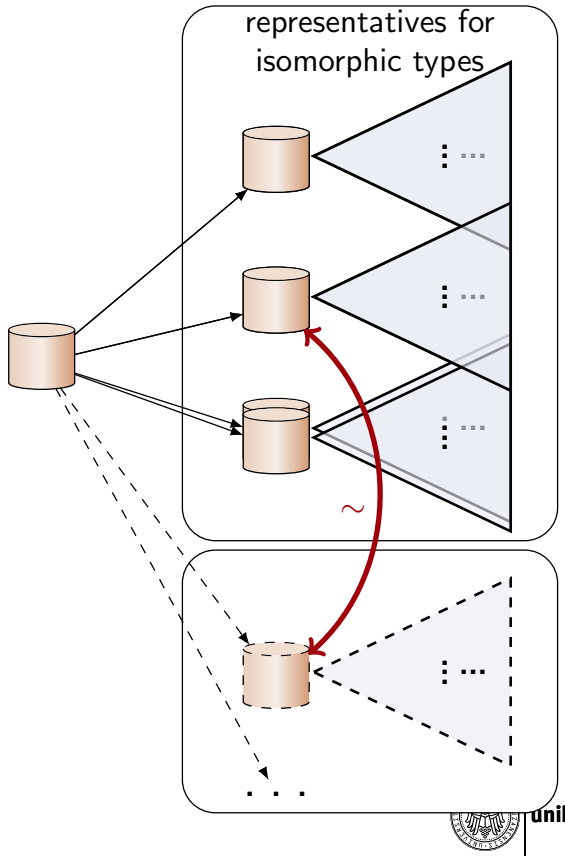
State-bounded Systems: Decidability for $\mu\mathcal{L}_P$

Theorem

Verification of $\mu\mathcal{L}_P$ over state-bounded DCDSs is decidable and can be reduced to model checking of propositional μ-calculus over a finite transition system.

Crux: construct a **faithful abstraction** $\Theta_{\mathcal{S}}$ for $\Upsilon_{\mathcal{S}}$, collapsing infinite branching and compacting infinite runs.

1. **Prune** infinite branching (isomorphic types).
2. Finite abstraction along the runs:
   - **Recycle** old, non-persisting objects instead of inventing new ones.



Sufficient Syntactic Conditions

State- and run-boundedness are semantic conditions, which are undecidable to check.

Hence, we introduce two sufficient **syntactic conditions**:

- Weak acyclicity (cf. data exchange), to check whether a DCDS with **deterministic services** is run-bounded.
- Generate-recall acyclicity, to check whether a DCDS is state-bounded.

The two acyclicity conditions are incomparable.

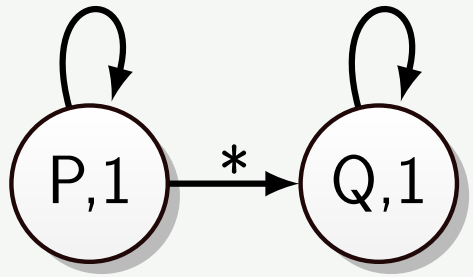
Both conditions are checked against a **dependency graph** that abstracts the data-flow of the DCDS process layer.

Sufficient Syntactic Conditions - Example 1

Example

Consider a DCDS $\mathcal{S}$ with process $\{\text{true} \longmapsto \alpha()\}$,

$$\text{action } \alpha() : \left\{ \begin{array}{l} P(x) \rightsquigarrow P(x) \\ P(x) \rightsquigarrow Q(f(x)) \\ Q(x) \rightsquigarrow Q(x) \end{array} \right\}$$



Consider **nondeterministic** service calls.

$\mathcal{S}$ is **not** state-bounded.

The problem comes from the interplay between:

- a generate cycle that continuously feeds a path issuing service calls;
- a recall cycle that accumulates the obtained results.
- (+ the fact that both cycles are active at the same time)

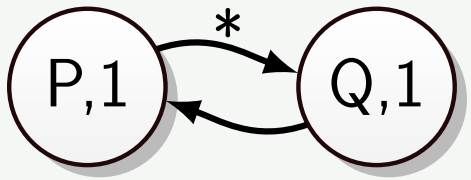
GR-acyclicliity detects exactly these undesired situations.

Sufficient Syntactic Conditions - Example 2

Example

Consider a DCDS $\mathcal{S}$ with process $\{\text{true} \longmapsto \alpha(), \text{true} \longmapsto \beta()\}$,

actions $\alpha() : \{P(x) \rightsquigarrow Q(f(x))\}$
$\beta() : \{Q(x) \rightsquigarrow P(x)\}$



Consider **deterministic** service calls.

$\mathcal{S}$ is **not** run-bounded.

The problem comes from:

- repeated calls to the same service...
- every time using fresh values that are directly (or indirectly) obtained by manipulating previous results produced by the same service.

Weak acyclicity detects these undesired situations.

Outline

1 Motivations

2 Artifact-Centric Approach

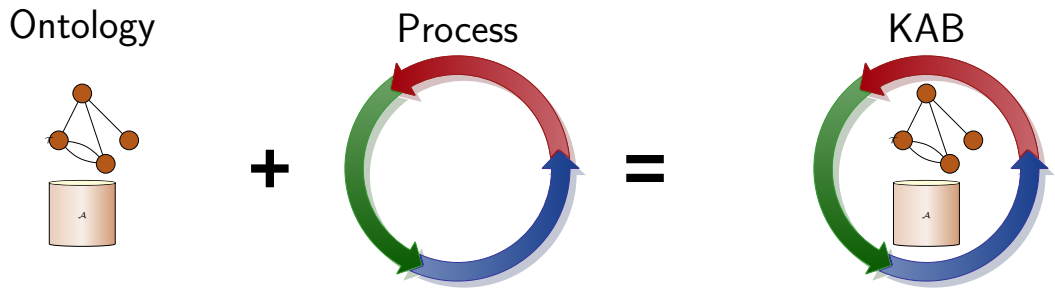
3 Data-Centric Dynamic Systems

4 Verification of Data-Aware Processes

5 Incomplete Information in the Data Layer

6 References

Knowledge and Action Bases (KAB)



Using a knowledge base as information model, we can:

- Better capture the semantics of the domain of interest at the conceptual level.
- Take into account incomplete information.

Data Layer: Description logic KB

- Data schema: TBox expressed in a light-weight Description Logic
- Data instance: DL ABox

	$\mu\mathcal{L}_{FO}$		$\mu\mathcal{L}_{A}$		$\mu\mathcal{L}_{P}$		$\mu\mathcal{L}$
unrestricted	**U**	$\leftarrow$	**U**	$\leftarrow$	**U**	$\leftarrow$	**U**
weak-acyclicity	**?**		**D**	$\rightarrow$	**D**	$\rightarrow$	**D**

D: decidable

U: undecidable

unibz.it

Ongoing and Future Work

- Study relaxation of syntactic restrictions for state-boundedness.
- Develop further the KAB setting:
  - Deal with inconsistency w.r.t. the ontology.
  - Consider an Ontology-Based Data Access setting, where a data layer and a conceptual layer co-exist and are mapped to each other.
- Connection to other infinite-state formalisms.
  - Petri nets;
  - LTL with freeze quantifier;
  - Well-structured transition systems.
- Investigate how to deal with the exponential explosion w.r.t. the data.
- Investigate the fragments with lower complexities.
- Implementation of the approach, using state-of-the-art finite-state model checkers.
- Consider other reasoning services, e.g., composition, adversarial synthesis.

Thank you for your attention!

References I

[Bagheri Hariri *et al.*, 2011] Babak Bagheri Hariri, Diego C., Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli.

Foundations of relational artifacts verification.

In *Proc. of the 9th Int. Conference on Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 379–395. Springer, 2011.

[Bagheri Hariri *et al.*, 2012a] Babak Bagheri Hariri, Diego C., Giuseppe De Giacomo, Riccardo De Masellis, Marco Montali, and Paolo Felli.

Verification of description logic Knowledge and Action Bases.

In *Proc. of the 20th Eur. Conf. on Artificial Intelligence (ECAI 2012)*, pages 103–108, 2012.

[Bagheri Hariri *et al.*, 2012b] Babak Bagheri Hariri, Diego C., Giuseppe De Giacomo, Alin Deutsch, and Marco Montali.

Verification of relational data-centric dynamic systems with external services.

CoRR Technical Report arXiv:1203.0024, arXiv.org e-Print archive, 2012.

Available at `http://arxiv.org/abs/1203.0024`.

[Bagheri Hariri *et al.*, 2013] Babak Bagheri Hariri, Diego C., Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli.

Description logic Knowledge and Action Bases.

J. of Artificial Intelligence Research, 46:651–686, 2013.

References II

[Bhattacharya *et al.*, 2007] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su.

Towards formal analysis of artifact-centric business process models.

In *Proc. of the 5th Int. Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–234. Springer, 2007.

[Cangialosi *et al.*, 2010] Piero Cangialosi, Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati.

Conjunctive artifact-centric services.

In *Proc. of the 8th Int. Joint Conf. on Service Oriented Computing (ICSOC 2010)*, volume 6470 of *Lecture Notes in Computer Science*, pages 318–333. Springer, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati.

Verification of conjunctive artifact-centric services.

Int. J. of Cooperative Information Systems, 21(2):111–139, 2012.

[Deutsch *et al.*, 2007] Alin Deutsch, Liying Sui, and Victor Vianu.

Specification and verification of data-driven web applications.

J. of Computer and System Sciences, 73(3):442–474, 2007.

[Nigam and Caswell, 2003] A. Nigam and N. S. Caswell.

Business artifacts: An approach to operational specification.

IBM Systems J., 42(3):428–445, 2003.

unibz.it