unibz

# Verification of Parameterized Data-Aware Dynamic Systems

Marco Montali

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

Credits to
Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, Andrey Rivkin

PV 2014

# Data and Processes

unibz

The information assets of an organization are constituted by:

- data, and
- processes, that determine how data changes and evolves over time.

# Data and Processes

unibz

The information assets of an organization are constituted by:

- data, and
- processes, that determine how data changes and evolves over time.

## Conceptual Modeling

Both aspects are modelled conceptually, but:

- Using different modeling tools
- By different teams with different competences
- Connection between the two is NOT modelled conceptually

# Data and Processes

unibz

The information assets of an organization are constituted by:

- data, and
- processes, that determine how data changes and evolves over time.

## Conceptual Modeling

Both aspects are modelled conceptually, but:

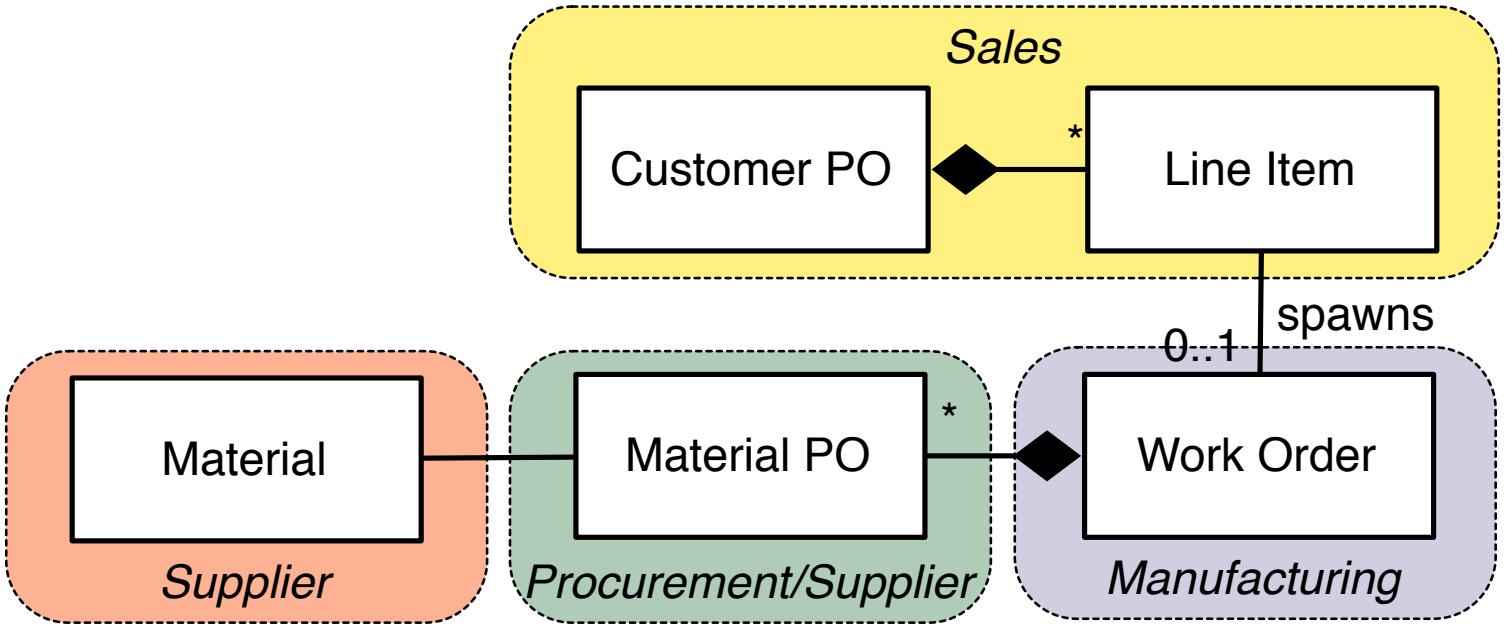
- Using different modeling tools
- By different teams with different competences
- Connection between the two is NOT modelled conceptually

## Consequence

Full reasoning support, e.g., for verification taking into account both process and data, is not possible!

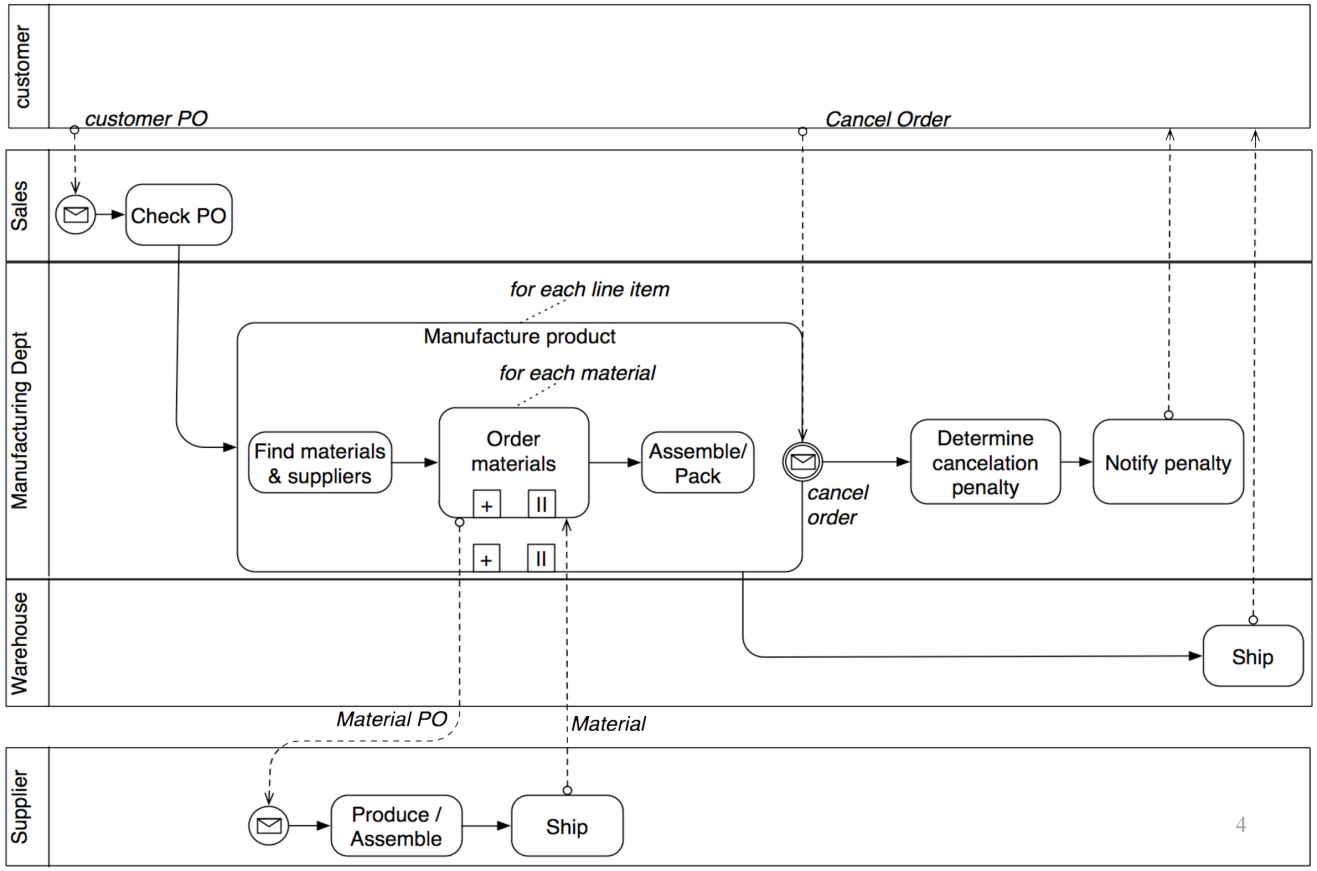
# A Classical Data Model

unibz




**How are data of this schema evolved over time?**

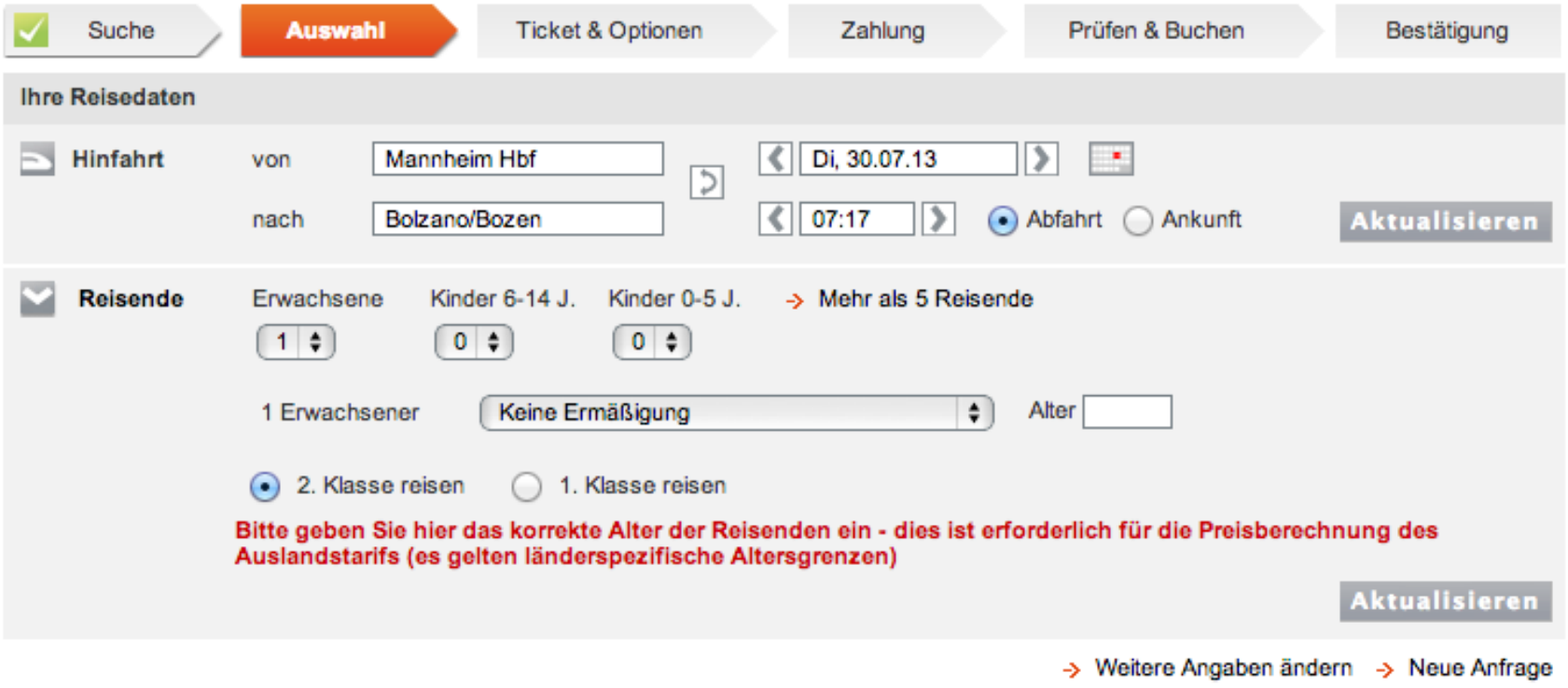
# A Classical Process Model




**Where are the data?**

# Reality: A Deployed Process

unibz




- Current data $\rightsquigarrow$ fields, text, highlights.
  - Include status attributes.
- Actions $\rightsquigarrow$ buttons, links.
- Input params $\rightsquigarrow$ writable fields.

# Our goals

unibz

## Formalization of Data-Aware Dynamic Systems

Lay the foundations for processes dealing with a full-fledged data layer:

- **relational database with constraints** (complete information);
- conceptual model/ontology (incomplete information);
- both (cf. ontology-based data access).

## Reasoning support along the entire process lifecycle

Develop techniques for:

- **(design-time) verification** and synthesis;
- (run-time) operational support (monitoring);
- (a-posteriori) analysis and mining.

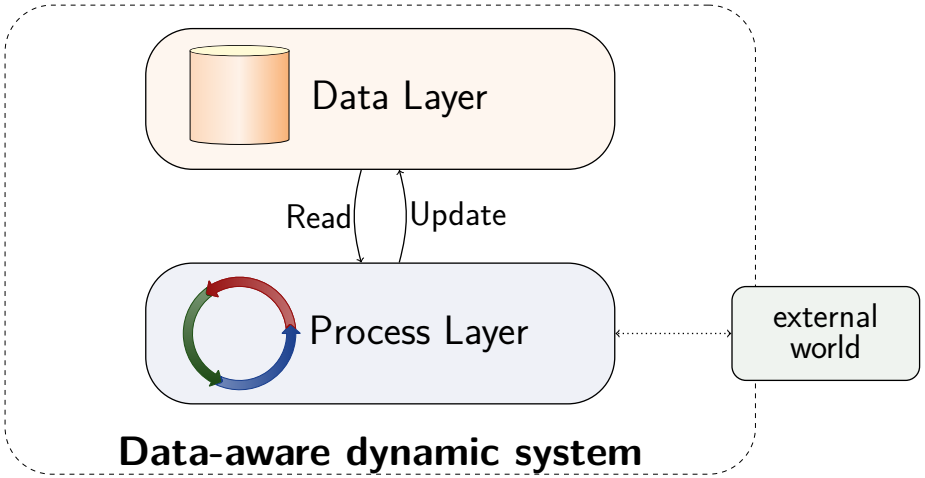
## Grounding of the approach

Focus on concrete languages/settings like: business artifacts + adaptive case management (GSM, CMMN, ...), healthcare processes, dynamic web apps, ...

# Data-Aware Dynamic System

unibz

A dynamic system that manipulates data over time.




- Data layer: maintains data of interest.
  - Relational database.
  - (Description logic) ontology.
- Process layer: evolves the extensional part of the data layer.
  - Control-flow component: determines when actions can be executed.
  - Actions: atomic units of work that update the data.
    - ★ Interact with the external world to inject *fresh data* into the system.

# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x, y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(p) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq p & \rightsquigarrow & R(x,y) \\ R(p,y) & \rightsquigarrow & R(p, f(y)) \\ R(p,y) & \rightsquigarrow & Q(p) \end{array} \right\}$

# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x,y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(p) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq p & \rightsquigarrow & R(x,y) \\ R(p,y) & \rightsquigarrow & R(p,f(y)) \\ R(p,y) & \rightsquigarrow & Q(p) \end{array} \right\}$

R(a,b), R(a,c),
R(c,d),
Q(d),
S(b)

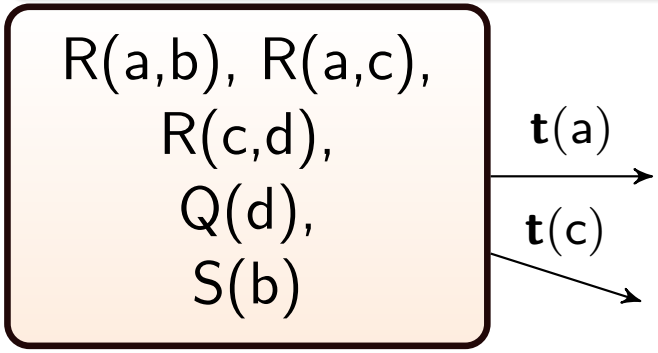
# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x,y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(p) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq p & \rightsquigarrow & R(x,y) \\ R(p,y) & \rightsquigarrow & R(p, f(y)) \\ R(p,y) & \rightsquigarrow & Q(p) \end{array} \right\}$



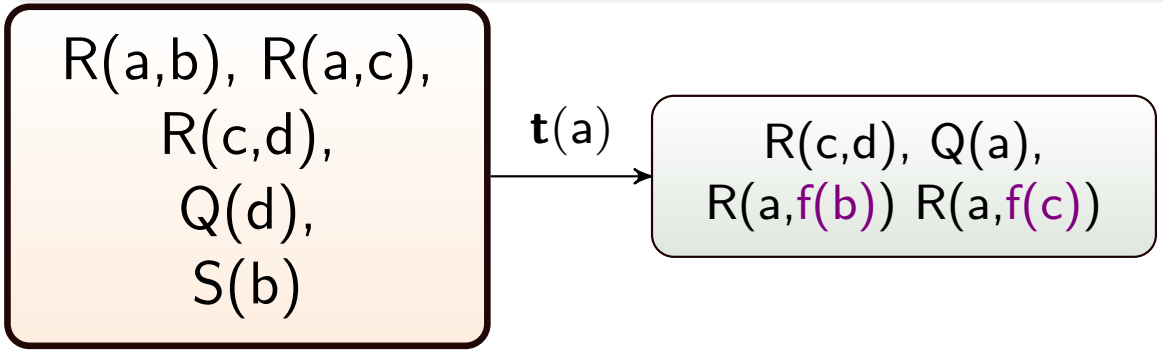
# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x,y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \rightsquigarrow & R(x,y) \\ R(\mathsf{a},y) & \rightsquigarrow & R(\mathsf{a},f(y)) \\ R(\mathsf{a},y) & \rightsquigarrow & Q(\mathsf{a}) \end{array} \right\}$



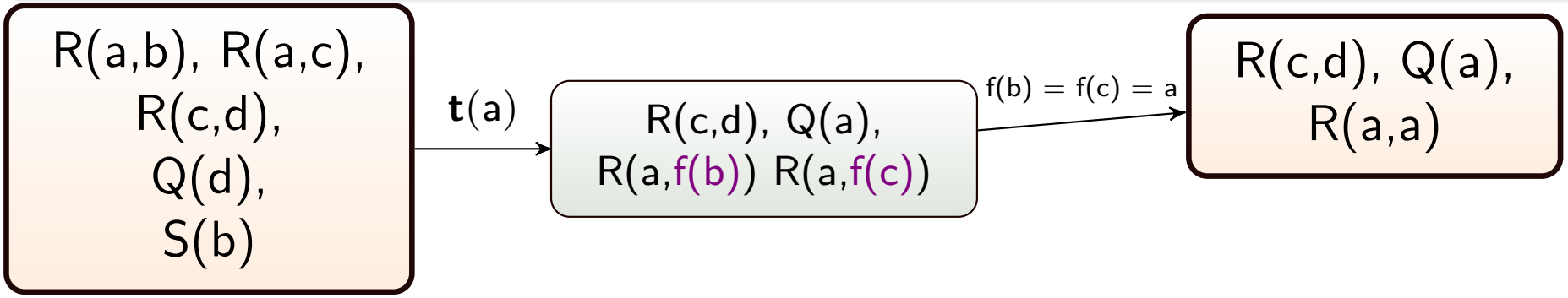
# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x,y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \rightsquigarrow & R(x,y) \\ R(\mathsf{a},y) & \rightsquigarrow & R(\mathsf{a},f(y)) \\ R(\mathsf{a},y) & \rightsquigarrow & Q(\mathsf{a}) \end{array} \right\}$



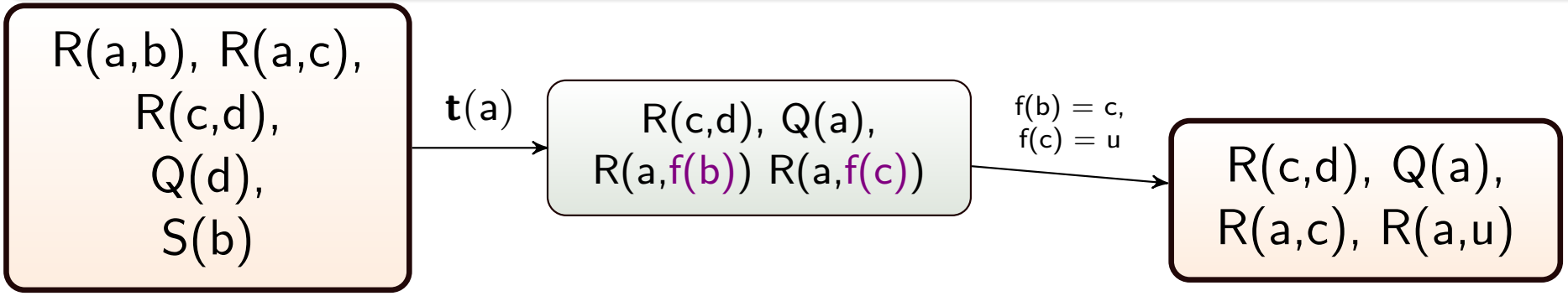
# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x,y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \rightsquigarrow & R(x,y) \\ R(\mathsf{a},y) & \rightsquigarrow & R(\mathsf{a},f(y)) \\ R(\mathsf{a},y) & \rightsquigarrow & Q(\mathsf{a}) \end{array} \right\}$



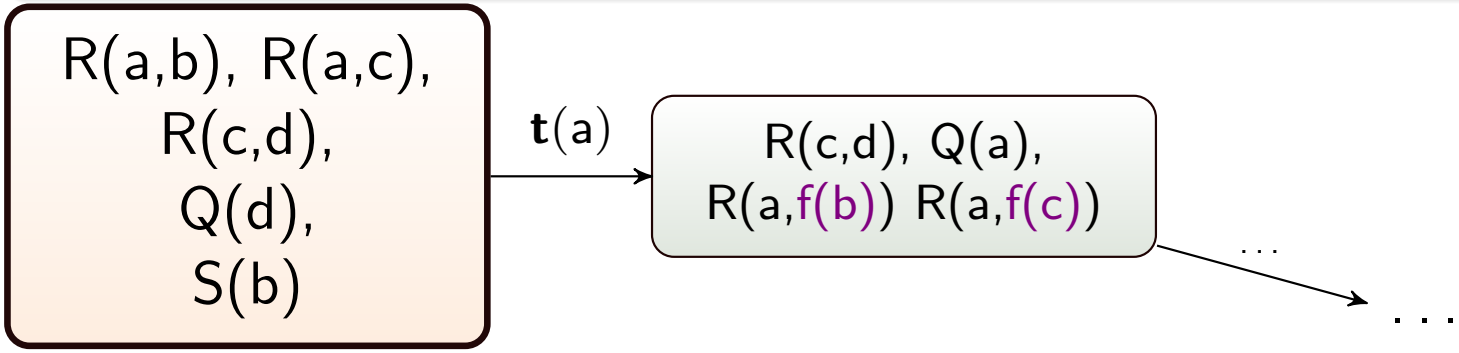
# Data-Centric Dynamic Systems (DCDSs) [PODS13]

unibz

- Data layer: relational database with FO constraints.
- Process (control-flow): condition-action rules.
- Actions: specified by (parallel) effects that query the current state and determine the next state.
  - Service calls can be invoked to incorporate new data.

## Example

- Data layer: schema $\{R/2, Q/1, S/1\}$, no constraint.
- Process: $\exists y.R(x,y) \mapsto \mathbf{t}(x)$.
- Action: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \rightsquigarrow & R(x,y) \\ R(\mathsf{a},y) & \rightsquigarrow & R(\mathsf{a},f(y)) \\ R(\mathsf{a},y) & \rightsquigarrow & Q(\mathsf{a}) \end{array} \right\}$

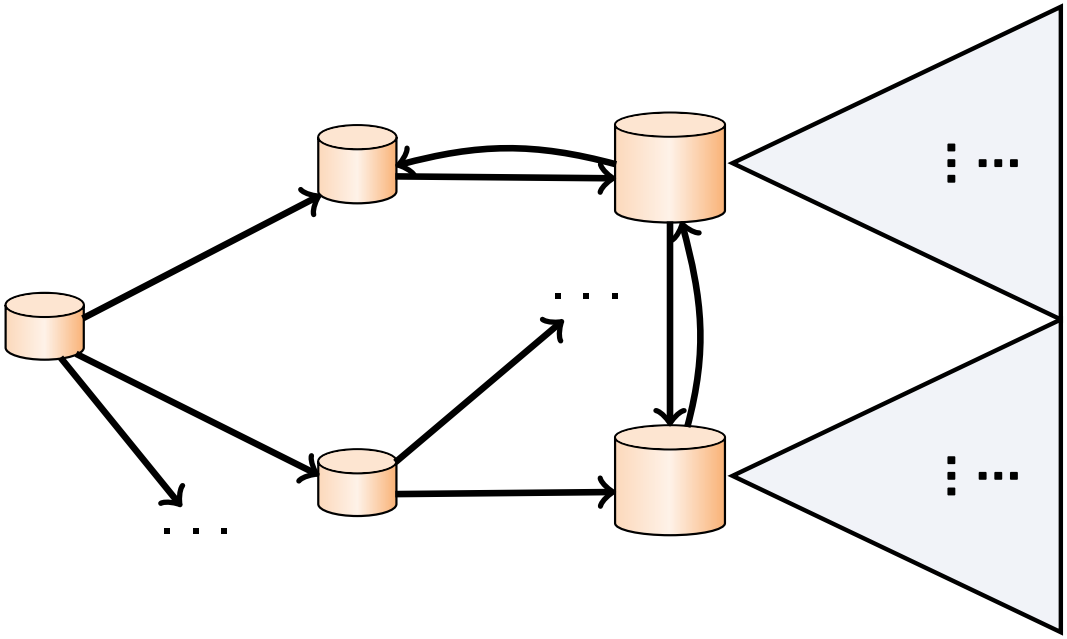


# Verification

unibz

Execution semantics: a relational, infinite-state transition system.

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



## Verification Problem

Check whether the dynamic system guarantees a desired property, expressed in some variant of a first-order temporal logic.

# Verification Logics

unibz

Requirements for temporal/dynamic properties:

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

**$\mu\mathcal{L}_A$**

First-order $\mu$-calculus with active domain quantification.

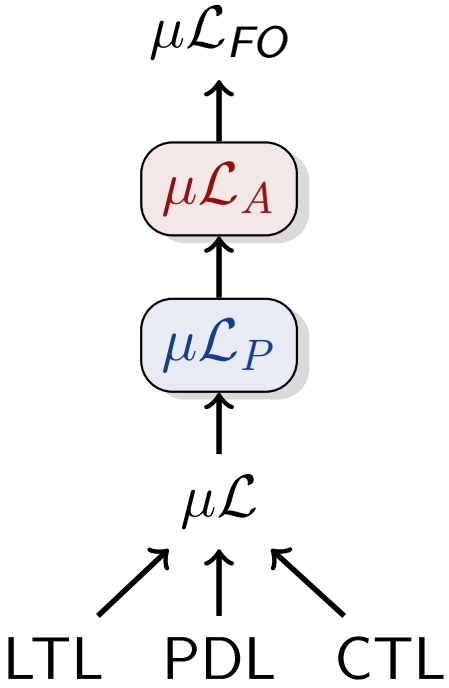
Example: In every state of the system, each order stored in that state is eventually shipped.

**$\mu\mathcal{L}_P$**

Syntactically limits first-order quantification across states: it applies only to those objects that persist.

- Internal primary keys, student ids, pure names, ...

Example: In every state of the system, each order stored in that state persists in the system until it is shipped.



# Undecidability

unibz

Verification of propositional reachability properties is **undecidable** even for DCDSs with unary relations only.

## Counter increment `k1:c++; goto k2`

$$PC(\texttt{k1}) \mapsto \textbf{inc-C}(\texttt{k2}) \text{ where } \textbf{inc-C}(k_n) : \left\{ \begin{array}{rcl} C(x) & \rightsquigarrow & C(x) \\ C(x) & \rightsquigarrow & C_{old}(x) \\ \text{true} & \rightsquigarrow & C_{new}(f()), C(f()) \\ \text{true} & \rightsquigarrow & PC(k_{next}) \end{array} \right\}$$

with fresh name database constraint: $\exists x. C_{new}(x) \wedge C_{old}(x) \rightarrow \text{false}$

## Counter conditional decrement `k1:if(c=0) goto k2; else{c--; goto k3;}`

$$PC(\texttt{k1}) \wedge C(e) \mapsto \textbf{dec-C}(e, \texttt{k2}) \text{ where } \textbf{dec-C}(e, k_n) : \left\{ \begin{array}{rcl} C(x) \wedge x \neq e & \rightsquigarrow & C(x) \\ \text{true} & \rightsquigarrow & PC(k_n) \end{array} \right\}$$

$$PC(\texttt{k1}) \wedge \neg\exists x. C(x) \mapsto \textbf{jmp}(\texttt{k3}) \text{ where } \textbf{jmp}(k_n) : \left\{ \text{true} \rightsquigarrow PC(k_n) \right\}$$

# Conditions for DCDSs

unibz

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

## Run boundedness

Each run of $\Upsilon_{\mathcal{S}}$ accumulates 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.

# Conditions for DCDSs

unibz

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

## Run boundedness

Each run of $\Upsilon_{\mathcal{S}}$ accumulates 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

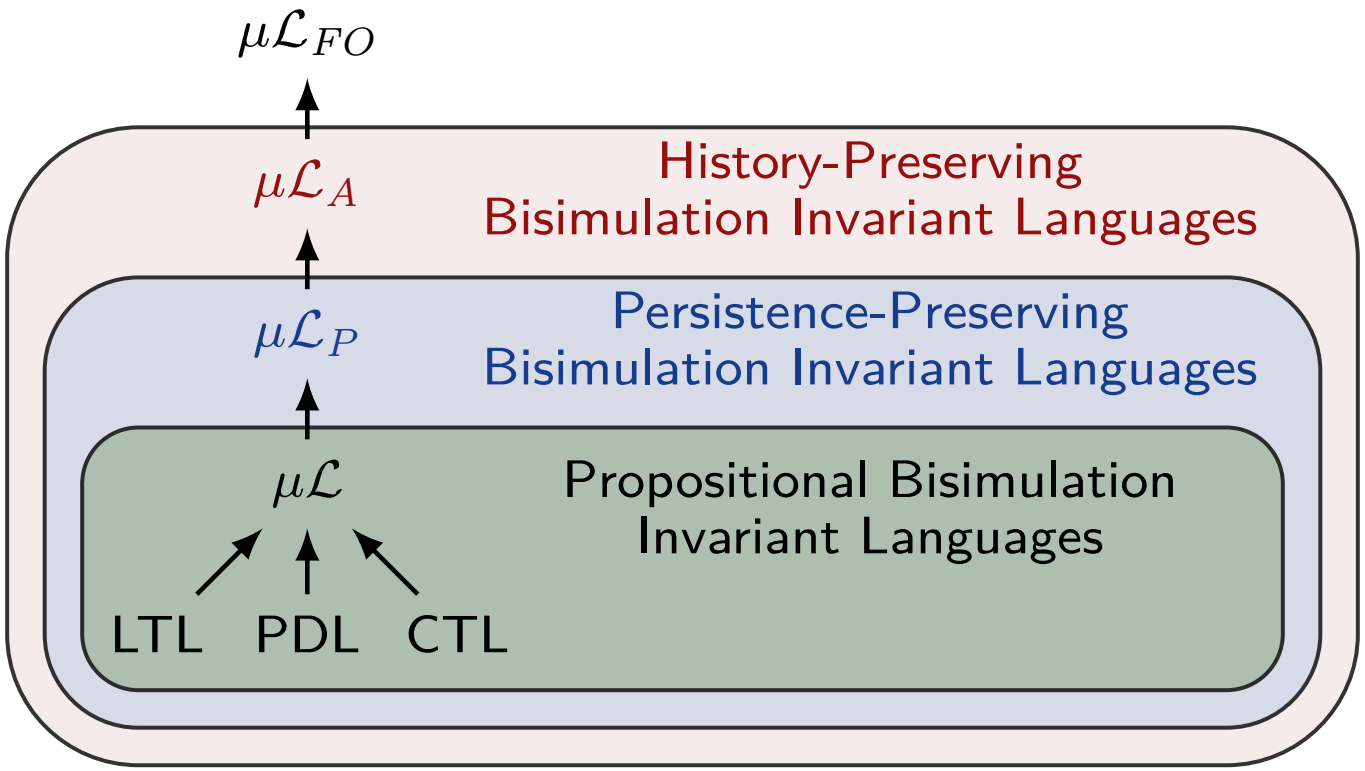
Each state of $\Upsilon_{\mathcal{S}}$ contains only a bounded number of objects.

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

# Bisimulations

Corresponding notions of bisimulation are defined. They 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.
  - For $\mu\mathcal{L}_P$: forgetting about old, non-persisting objects.



# Summary of Results [PODS13]

unibz

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

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

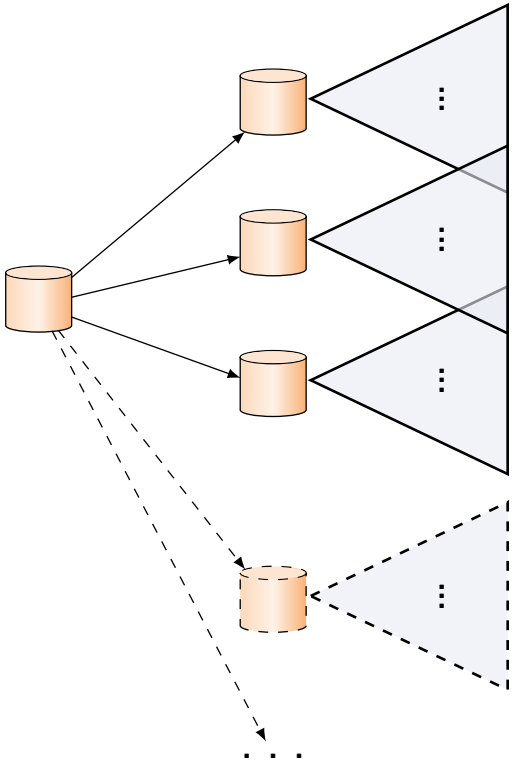
# Run-Bounded Systems: Decidability for $\mu\mathcal{L}_A$

unibz

**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.



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

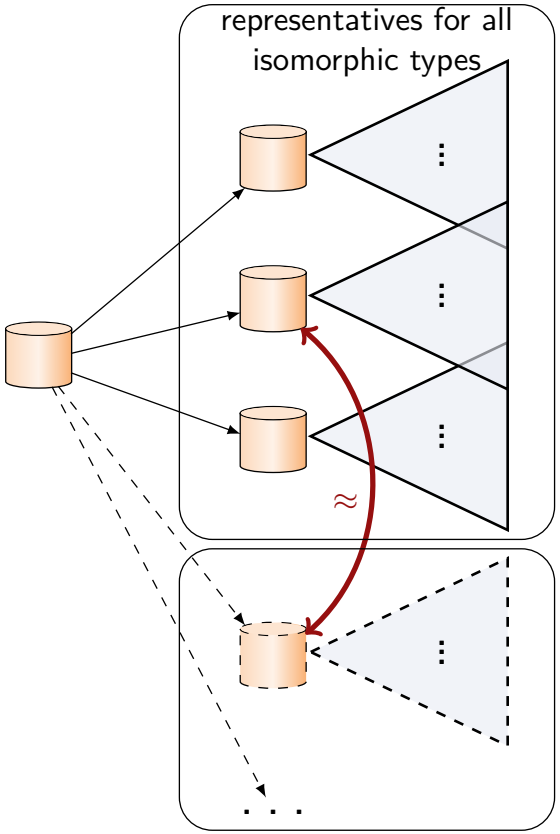
unibz

**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 $\mu\mathcal{L}_A$

unibz

**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 $\mu\mathcal{L}_P$

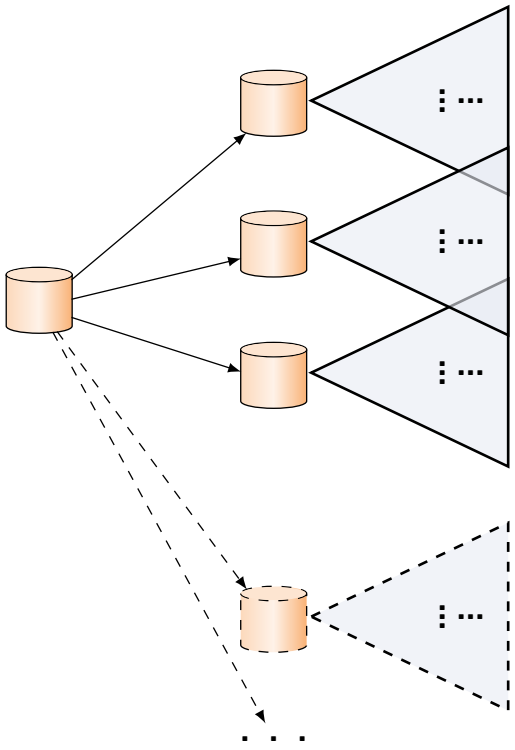
unibz

**Theorem**

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

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).



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

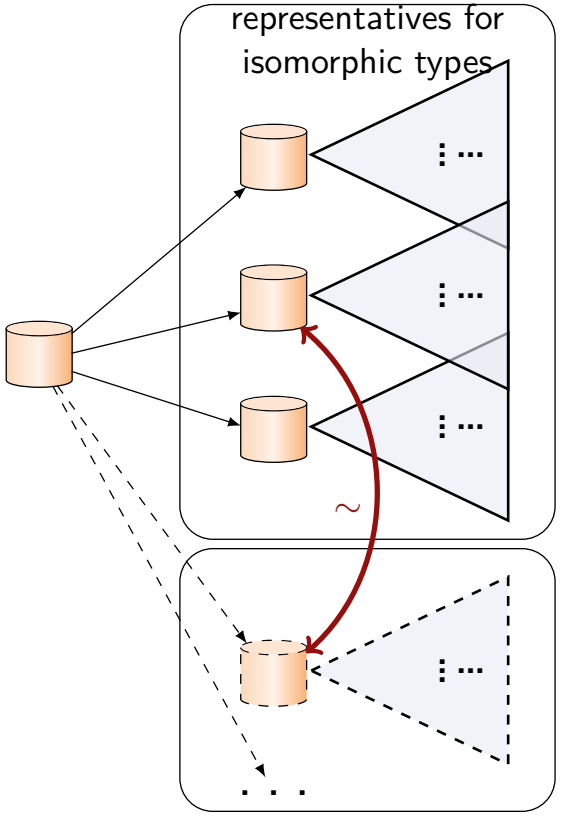
unibz

**Theorem**

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

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).



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

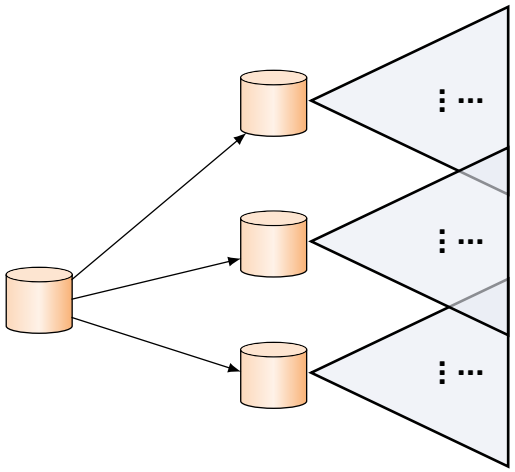
unibz

**Theorem**

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

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).



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

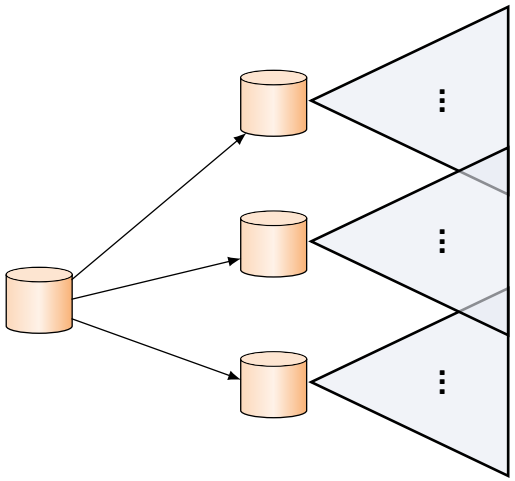
unibz

**Theorem**

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

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 when possible.
   - At some point, no fresh value needs to be introduced anymore.

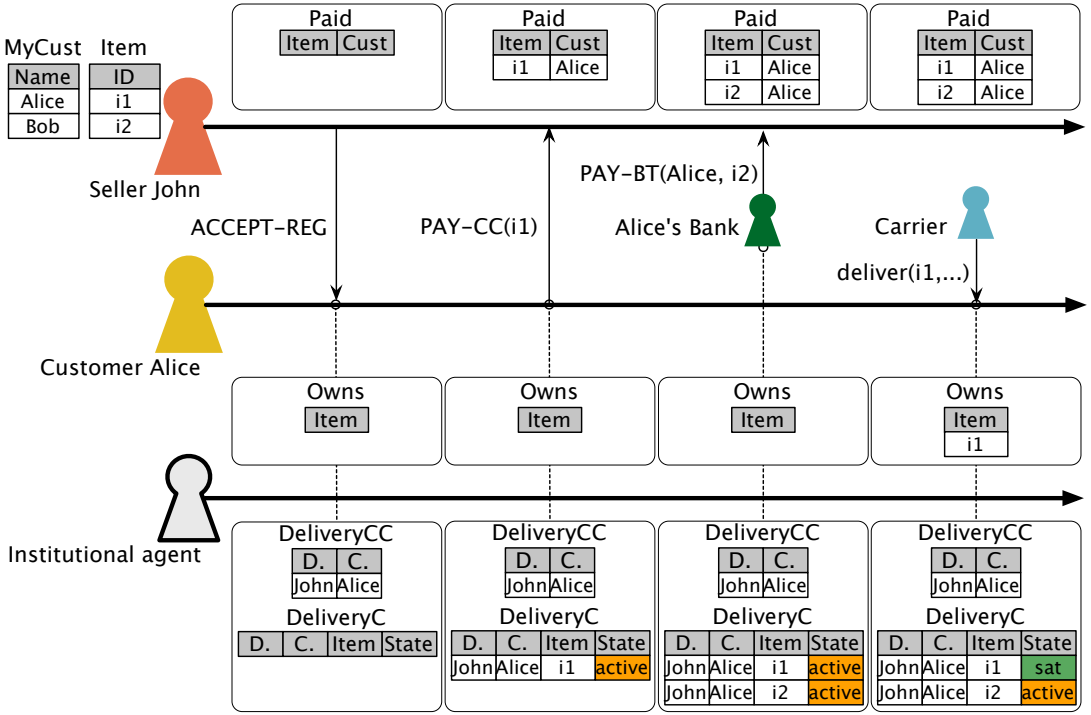


No need to know the actual bound!

# Parameterization with Names

In [AAMAS14]: extension towards data-aware multiagent systems.

- Agents use names to exchange data.
- An institutional agent manages name creation and deletion.




State-boundedness implies that unboundedly many agents/names can be seen in a run, provided that they do not coexist in the same state.

# On State-Boundedness

unibz

State- (and run-) boundedness are semantic properties, undecidable to check.

## Big question 1

Do there exist interesting classes for which state-boundedness is decidable?

# On State-Boundedness

unibz

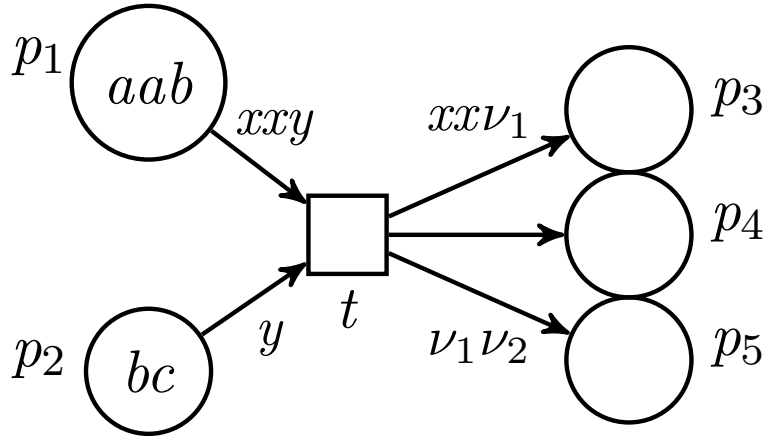
State- (and run-) boundedness are semantic properties, undecidable to check.

## Big question 1

Do there exist interesting classes for which state-boundedness is decidable?

**+: Petri nets with name management**

By Rosa-Velardo et al.




[WSFM14]:
Translation to DCDSs and $\mu\mathcal{L}_P$

# On State-Boundedness

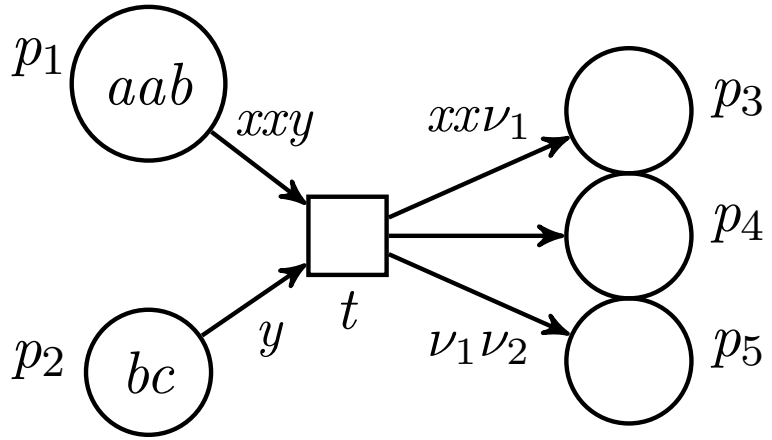
State- (and run-) boundedness are semantic properties, undecidable to check.

## Big question 1

Do there exist interesting classes for which state-boundedness is decidable?

**+: Petri nets with name management**

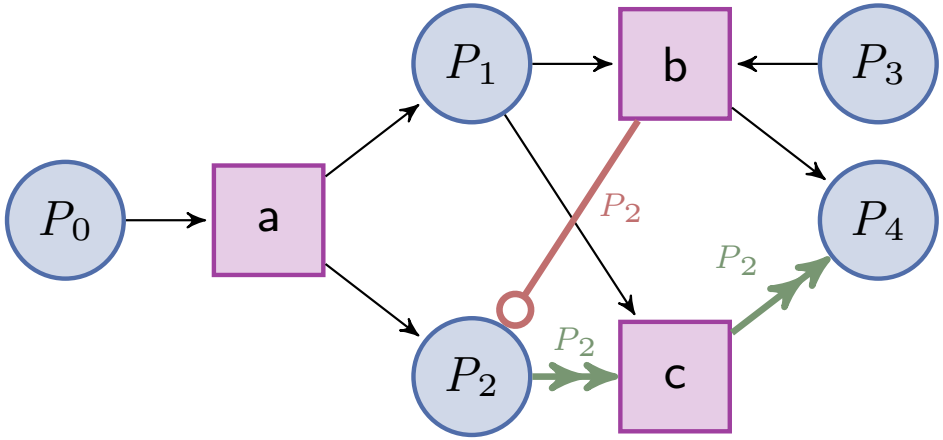
By Rosa-Velardo et al.




[WSFM14]:
Translation to DCDSs and $\mu\mathcal{L}_P$

**–: Reset-Transfer Nets**

$\subset$ Reset Post G-nets by Dufour et al.




[KR14]:
"Lossy" correspondence with DCDSs.

# Attacking State-Boundedness

unibz

These variants of Petri nets correspond to restricted classes of DCDSs with unary relations, limited use of negation, no or limited joins, ...

## Big question 2

How to check/guarantee that a system is state-bounded?

# Attacking State-Boundedness

unibz

These variants of Petri nets correspond to restricted classes of DCDSs with unary relations, limited use of negation, no or limited joins, ...

## Big question 2

How to check/guarantee that a system is state-bounded?

## Strategy 1

Sufficient, syntactic conditions.

Taking inspiration from conditions on chase termination for TGDs:

- Extract a data-flow graph from the DCDS action specification.
- Check how data flow through this graph.
- See [PODS13] and [KR14].

# Attacking State-Boundedness

unibz

These variants of Petri nets correspond to restricted classes of DCDSs with unary relations, limited use of negation, no or limited joins, ...

## Big question 2

How to check/guarantee that a system is state-bounded?

## Strategy 1

Sufficient, syntactic conditions.

Taking inspiration from conditions on chase termination for TGDs:

- Extract a data-flow graph from the DCDS action specification.
- Check how data flow through this graph.
- See [PODS13] and [KR14].

## Strategy 2

State-boundedness by design.

Design methodologies so as to guarantee state-boundedness. E.g., in [ICSOC13]:

- Processes are bound to evolving business objects (artifacts).
- Each business object manipulate boundedly many data.
- (New) business objects pick their names from a fixed pool of ids.

# Towards Controlled Unboundedness

unibz

## Observation: Parameterization in data-aware dynamic systems

- In classical process management ...
  - Central notion of case representing a process instance.
- In artifact-centric process management:
  - Central notion of business object gluing data and behaviour together.
- New cases/objects often created by external stakeholders!

# Towards Controlled Unboundedness

unibz

## Observation: Parameterization in data-aware dynamic systems

- In classical process management . . .
  - Central notion of case representing a process instance.
- In artifact-centric process management:
  - Central notion of business object gluing data and behaviour together.
- New cases/objects often created by external stakeholders!

## Hence. . .

No bound on the number of simultaneously active cases/objects.

# Towards Controlled Unboundedness

unibz

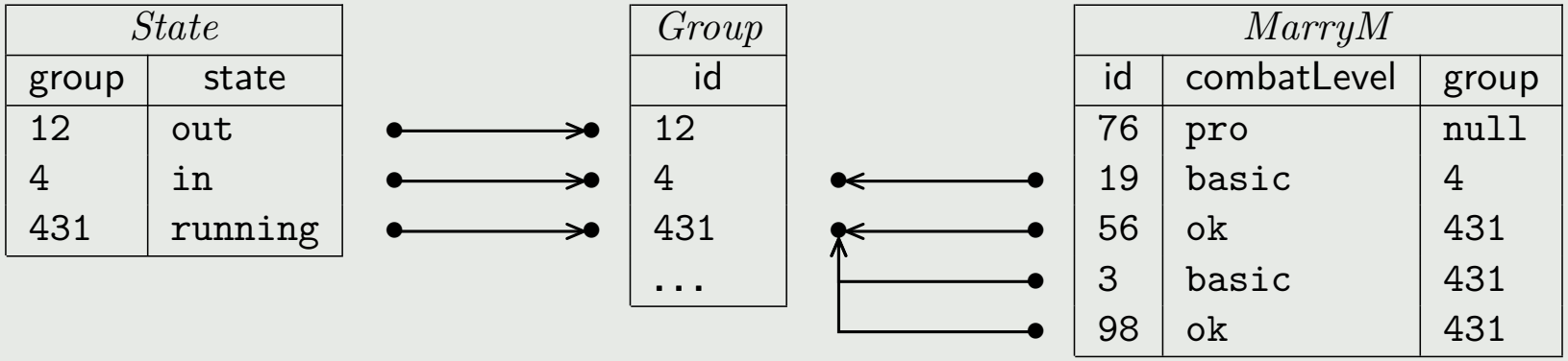
## Observation: Parameterization in data-aware dynamic systems

- In classical process management . . .
  - Central notion of case representing a process instance.
- In artifact-centric process management:
  - Central notion of business object gluing data and behaviour together.
- New cases/objects often created by external stakeholders!

## Hence. . .

No bound on the number of simultaneously active cases/objects.

## But. . .



*State*

| group | state |
|---|---|
| 12 | out |
| 4 | in |
| 431 | running |

*Group*

| id |
|---|
| 12 |
| 4 |
| 431 |
| ... |

*MarryM*

| id | combatLevel | group |
|---|---|---|
| 76 | pro | null |
| 19 | basic | 4 |
| 56 | ok | 431 |
| 3 | basic | 431 |
| 98 | ok | 431 |

# Towards Controlled Unboundedness

unibz

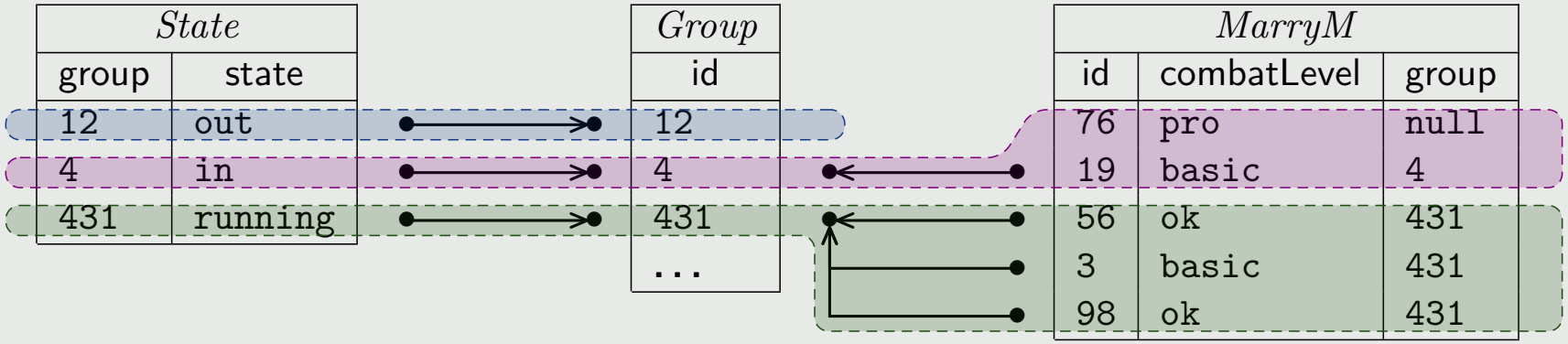
## Observation: Parameterization in data-aware dynamic systems

- In classical process management . . .
  - Central notion of case representing a process instance.
- In artifact-centric process management:
  - Central notion of business object gluing data and behaviour together.
- New cases/objects often created by external stakeholders!

## Hence. . .

No bound on the number of simultaneously active cases/objects.

## But. . .



# Towards Controlled Unboundedness

unibz

In [CIKM14,Subm14] we formally capture the two notions of "data isolation" and "relative boundedness":

- Modelling guidelines for the data component and how the process component accesses to it.

## Restrictions

1. Queries must be navigational: no arbitrary access to relations.
2. 1-to-many relations require a number restriction on the "many" side.
3. Each case cannot create a chain of tuples of unbounded lenght.
4. Cases can share tuples only in a controlled way.
   - So as to avoid the construction of chains of cases.

## Key Decidability Result

Verification of navigational $\mu\mathcal{L}_P$ properties on the unbounded system can be reduced to verifying a bounded number of cases/business objects!

- A sort of data-aware cutoff technique.

# Towards Concrete Domains

unibz

Ongoing work with Diego Calvanese and Giorgio Delzanno.

## What about concrete domains with specific relations?

E.g., ordered domains with $>$.

- Think about classical ticket-based coordination algorithms.

## How does this interact with state-boundedness?

Some key insights:

- No hope to include the **successor** relation.
  - 2 data slots are sufficient to encode two counters.
- OK to include dense linear orders.
  - Thanks to state-boundedness:

| Rigid $>$ relation over the entire domain | $\longrightarrow$ | Non-rigid $GreaterThan$ relation over active domain elements. |
|---|---|---|

# Conclusion

unibz

- Our work is grounded in real-world data-aware processes.
- State-boundedness provides the basis for robust conditions for decidability.
- Key idea: only finitely many data are important per sè, the others act as placeholders (cf. pure values).
- Complexity-wise: we are studying how the exponentiality in the data inherent in data-aware systems can be tamed, through a suitable modularization of the read-write data slots into independent portions.
- Parameterization currently works for boundedly many components simultaneously coexisting in the system, or by restricting their interaction through the data component.

# Relevant References 1/2

unibz


[PODS13] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, M. Montali. **Verification of relational data-centric dynamic systems with external services**. 32nd Symp. on Principles of Database Systems (PODS). ACM Press, 2013.

[PODS13b] D. Calvanese, G. De Giacomo, M. Montali. **Foundations of data-aware process analysis: A database theory perspective**. 32nd Symposium on Principles of Database Systems (PODS). ACM Press, 2013.

[ICSOC13] D. Solomakhin, M. Montali, S. Tessaris, R. De Masellis. **Verification of artifact-centric systems: Decidability and modeling issues**. 11th Int. Conf. on Service Oriented Computing (ICSOC), v. 8274 of LNCS. Springer, 2013.

[KR14] B. Bagheri Hariri, D. Calvanese, A. Deutsch, M. Montali. **State-boundedness for decidability of verification in data-aware dynamic systems**. 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR). AAAI Press, 2014.

# Relevant References 2/2

unibz


[AAMAS14] M. Montali, D. Calvanese, G. De Giacomo. **Verification of data-aware commitment-based multiagent systems**. 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS). IFAAMAS, 2014.

[WSFM14] M. Montali, A. Rivkin. **Formal Verification of Petri Nets with Names**. 11th Int. WS on Web Services and Formal Methods (WS-FM). 2014, *to appear*.

[CIKM14] D. Calvanese, M. Montali, M. Estanol, E. Teniente. **Verifiable UML Artifact-Centric Business Process Models**. 23rd Int. Conf. on Information and Knowledge Management (CIKM). 2014, *to appear*.

[Subm14] M. Montali. **Soundness of data-aware, case-centric processes**. 2014, *submitted*.


Other works consider a description-logic knowledge base or an ontology-based data access system as the data component.

- See [RR12, ECAI12, JAIR13, IJCAI13, RR13, ICSOC13b].