Data-aware Processes: Modeling, Mining, and Verification

Part 3: Verification

Diego Calvanese

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

3rd International Winter School on Big Data (BigDat 2017)
13–17/2/2017 – Bari, Italy

Outline

1. Formal verification
2. Verification for Data-Centric Dynamic Systems
3. Conclusions

unibz

Outline

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 $

Ariane 5 (1996)

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

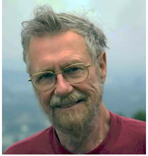
Toyota Prius (2010)

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]



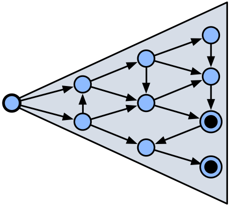
unibz

Verification via model checking

Verification via model checking

Process control-flow

Finite-state transition system



$\models \Phi$

Propositional temporal formula

(Un)desired property

[2007 Turing Award: Clarke, Emerson, Sifakis]

Model checking technology requires the transition system to be finite.

Business process analysis

In BPM, **process model analysis** is considered the second most influential topic in the last decade (after process modeling languages) [Aalst 2012].

However:

- Data has been abstracted away.
- Emphasis has been on the control-flow dimension:
  $\leadsto$ sophisticated techniques for absence of deadlocks, boundedness, soundness, or domain-dependent properties expressed in LTL or CTL.

Basic assumption: control-flow is captured by a (possibly infinite-state) propositional labeled transition system,

- labels represent the process tasks/activities
- concurrency is represented by interleaving
- transition system usually not represented explicitly, but is implicitly "folded" into a Petri net

Verification of Petri nets

Verification of Petri nets:

- Undecidable in general [Esparza 1997, 1998].
- Decidable for safe/bounded nets (transition graph is finite-state).

No satisfactory solution for specification and analysis of data-aware processes:

- Colored Petri nets not suited to represent a DB:
  - Data are variables associated to tokens.
  - Data are manipulated by procedural attachments to the transition in the net $\rightsquigarrow$ Cannot be analyzed!
- BPMN (OMG standard) and BPEL (OASIS standard) suffer from similar problems:
  - They leave connection between data and process unspecified (e.g., do not capture atomic task behaviour).
  - Hence, require to attach a program to every BPMN atomic task or BPEL service.

Impact of data on verification

The presence of data complicates verification significantly:

- **States** must be **modeled relationally** rather than propositionally.
- The resulting transition system is typically **infinite state**.
- Query languages for analysis need to combine two dimensions:
  - a temporal dimension to query the process execution flow, and
  - a first-order dimension to query the data present in the relational structures.

  $\leadsto$ We need **first-order variants of temporal logics**.

Model checking data-aware processes becomes immediately undecidable!

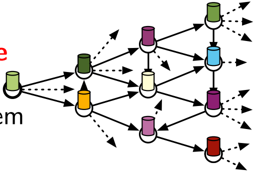
unibz

Formal verification of data-aware processes

Process + Data

↓

Infinite-state relational transition system



$\models \Phi$

First-order temporal formula

?

↑

(Un)desired property

Standard model checking technology fails!

Why first-order temporal logics

- To inspect data: **FO queries**
- To capture system dynamics: **temporal modalities**
- To track the evolution of objects: FO **quantification across** states

Example:

It is always the case that every order is eventually either cancelled or paid.

$$\mathbf{G}(\forall x.Order(x) \rightarrow \mathbf{F}(State(x, \texttt{cancelled}) \vee State(x, \texttt{paid})))$$

unibz

Finding the right balance

How can we mediate between:

- the form of data-aware processes, and
- the expressiveness of the temporal property language

such that

1. we are able to capture notable, real-world scenarios, but
2. verification stays decidable, and possibly efficient.

unibz

Dimensions of the verification problem space

We can consider variations of the verification problem that differ along various dimensions:

1. Static information model
2. Dynamic component
3. Interaction between static and dynamic component
4. Interaction with environment
5. Verification task / language

The richness of the problem space has brought about a great variety of approaches and results, and it is **difficult to** compare them and **get a comprehensive picture**.

unibz

Dim. 1: Static information model

- Propositional symbols $\rightsquigarrow$ Finite state system
- Fixed number of values from an unbounded domain
- Full-fledged database:
  - relational database
  - tree-structured data, XML
  - graph-structured data

Moreover:

- Presence or absence of constraints, and how they are considered
- Data under incomplete information
  - ontology (with intensional part usually assumed to be fixed)
  - full-fledged ontology-based data access system

Dim. 2: Dynamic component

- Implicit representation of time vs. implicit progression mechanism vs. explicit process
- When an explicit process is present:
  - how is the process dynamics represented?
  - procedural vs. declarative approaches (e.g., finite state machines vs. rule-based)
- Deterministic vs. non-deterministic behaviour
- Linear time vs. branching time model
- Finite vs. infinite traces

Dim. 3: Interaction between structure and dynamics

- Data is only accessed, but not modified
- No new values are inserted
- Full-fledged combination of the temporal and structural dimensions
- Restrictions play an important role:
    - restricted forms of querying the data
    - restricted quantification across time

Dim. 4: Interaction with environment

- Bounded vs. unbounded input
- Synchronous vs. asynchronous communication
  - message passing, possibly with queues
  - one-way or two-way service calls
- Which components are assumed fixed, and which may vary over time:
  - fixed database vs. varying database vs. varying portion of data
- Multiple devices/agents interacting with each other

Dim. 5: Verification task / language

Type of verification:

- Verification of specific temporal properties, e.g., reachability, absence of deadlock, boundedness, (weak) soundness, ...
- Verification of arbitrary formulas specified in some temporal logic
- Checking of properties with queries across the temporal dimension (in the style of temporal DBs)
- Different forms of verification / analysis:
  - dominance, simulation, containment, equivalence
  - synthesis from a given specification
  - composition of available components

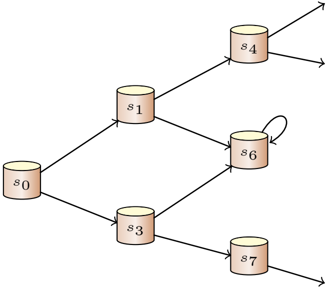
Outline

1. Formal verification
2. Verification for Data-Centric Dynamic Systems
3. Conclusions

Semantics of DCDSs via transition systems

Semantics of a DCDS $\mathcal{S}$ is given in terms of a **transition system** $\Upsilon_{\mathcal{S}}$:

- each state of $\Upsilon_{\mathcal{S}}$ has an associated DB over a common schema;
- the initial state is associated to the initial DB of the DCDS.



Note: $\Upsilon_{\mathcal{S}}$ is in general **infinite state**:

- infinite branching, due to the results of service calls,
- infinite runs, since infinitely many DBs may occur along a run;
- the DBs associated to the states are of unbounded size.

Verification for DCDSs

We are interested in the **verification** of temporal properties over $\Upsilon_{\mathcal{S}}$.

Idea to overcome infiniteness:

1. Devise a **finite-state** transition system $\Theta_{\mathcal{S}}$ that is a **faithful abstraction** of $\Upsilon_{\mathcal{S}}$ **independent of the formula** to verify.
2. Reduce the verification problem $\Upsilon_{\mathcal{S}} \models \Phi$ to the verification of $\Theta_{\mathcal{S}} \models \Phi$.

Problem: Verification of DCDSs is undecidable even for propositional reachability properties.
$\leadsto$ We need to pose restrictions on DCDSs.

We could draw inspiration from chase termination for tuple-generating dependencies in data exchange, and specifically from weak-acyclicity.

Restrictions on DCDSs

Run-bounded DCDS

Runs cannot accumulate more than a fixed number of different values.

- Transition system is still infinite-state due to infinite branching.
- This is a semantic condition, whose checking is undecidable.
  $\leadsto$ Sufficient syntactic condition: Weak-acyclicity.
- Run-boundedness is very restrictive for DCDSs with nondeterministic services.

State-bounded DCDS

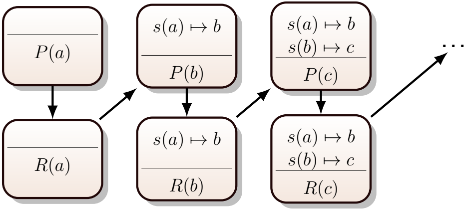
States cannot contain more than a fixed number of different values.

- Relaxation of run-boundedness.
- Infinite runs are possible.
- This is a semantic condition, whose checking is undecidable.
  $\leadsto$ Sufficient syntactic condition: e.g., GR-acyclicity.

Weak-acyclicity [Fagin et al. 2005]

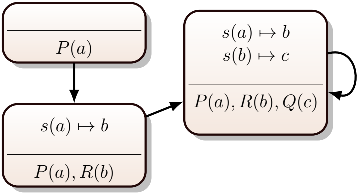
$\mathcal{I}_0 = \{P(a)\}$

$$\alpha : \begin{cases} P(x) \rightsquigarrow R(x), \\ R(x) \rightsquigarrow P(s(x)) \end{cases}$$



$\mathcal{I}_0 = \{P(a)\}$

$$\alpha : \begin{cases} P(x) \rightsquigarrow P(x), \\ P(x) \rightsquigarrow R(s(x)) \\ R(x) \rightsquigarrow Q(s(x)) \end{cases}$$



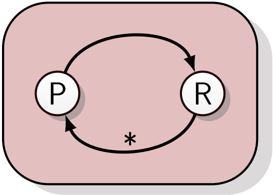
(We consider s to be a deterministic service.)

unibz

Weak-acyclicity [Fagin et al. 2005]

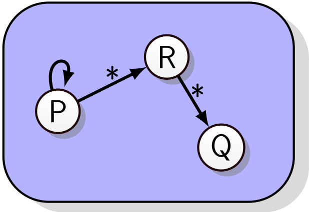
$$\mathcal{I}_0 = \{P(a)\}$$

$$\alpha : \begin{cases} P(x) \rightsquigarrow R(x), \\ R(x) \rightsquigarrow P(s(x)) \end{cases}$$



$$\mathcal{I}_0 = \{P(a)\}$$

$$\alpha : \begin{cases} P(x) \rightsquigarrow P(x), \\ P(x) \rightsquigarrow R(s(x)) \\ R(x) \rightsquigarrow Q(s(x)) \end{cases}$$



(We consider s to be a deterministic service.)

unibz

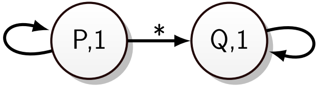
GR-acyclicity [Bagheri Hariri et al. 2013]

Example

Consider a DCDS with process $\{\text{true} \mapsto \alpha()\}$, a non-deterministic service s, and

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

We approximate the DCDS data-flow through a **dependency graph**.



The system is **not** state-bounded, due to:

- 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 simultaneously active).

GR-acycliclity detects exactly these undesired situations.

Verification formalisms for DCDSs

Boundedness is not sufficient for decidability.
We introduce two extensions of the modal μ-calculus $\mu\mathcal{L}$ / LTL with **restricted** forms of first order quantification.

History-Preserving quantification: $\mu\mathcal{L}_A$ / LTL-FO$_A$

FO quantification ranges over current active domain only.

Examples:

LTL-FO$_A$: $\forall x.\textsc{live}(x) \wedge \mathsf{Customer}(x) \rightarrow \mathbf{F}\ \mathsf{Gold}(x)$

$\mu\mathcal{L}_A$: $\forall x.\textsc{live}(x) \wedge \mathsf{Customer}(x) \rightarrow \mu Z.\mathsf{Gold}(x) \vee [-]Z$

Persistence-Preserving quantification: $\mu\mathcal{L}_P$ / LTL-FO$_P$

FO quantification ranges over persisting individuals only.

Examples:

LTL-FO$_P$: $\forall x.\textsc{live}(x) \wedge \mathsf{Gold}(x) \rightarrow \mathbf{G}\ \mathsf{Gold}(x)$

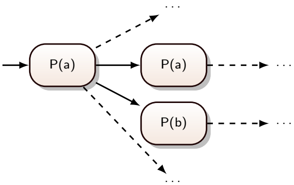
$\mu\mathcal{L}_P$: $\forall x.\textsc{live}(x) \wedge \mathsf{Gold}(x) \rightarrow \nu Z.\mathsf{Gold}(x) \wedge \textsc{live}(x) \wedge [-]Z$

$\mu\mathcal{L}_{FO}$/LTL-FO
↑
$\mu\mathcal{L}_A$/LTL-FO$_A$
↑
$\mu\mathcal{L}_P$/LTL-FO$_P$
↑
$\mu\mathcal{L}$
↑
LTL

Towards decidability

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

- infinite branching, due to external input;
- infinite runs, i.e., runs visiting infinitely many DBs.



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

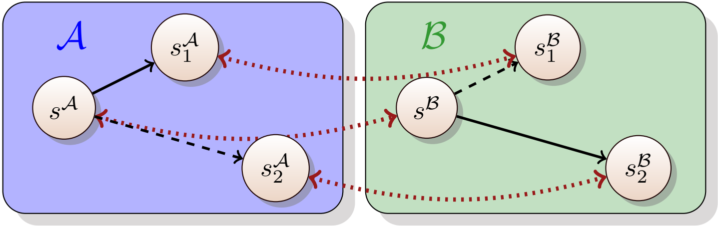
- We use bisimulation as a tool.
- We show that restricted DCDSs have a finite-state bisimilar transition system.

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!

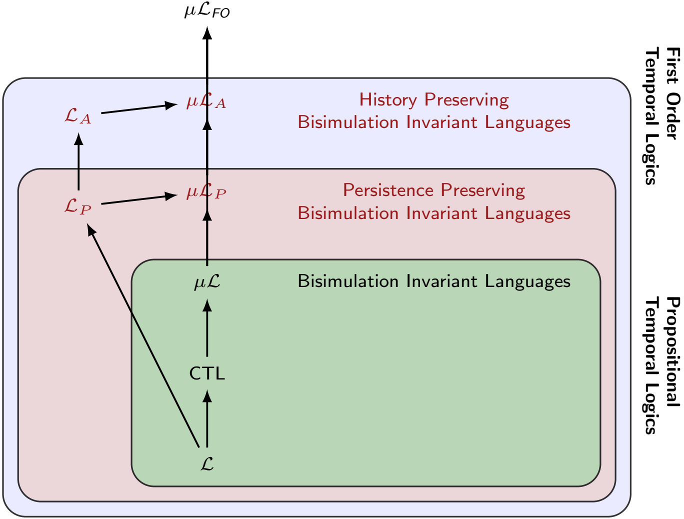
$\mathcal{A}$ and $\mathcal{B}$ are bisimilar, if their initial states are bisimilar.



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

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

Adapting the notion of bisimulation



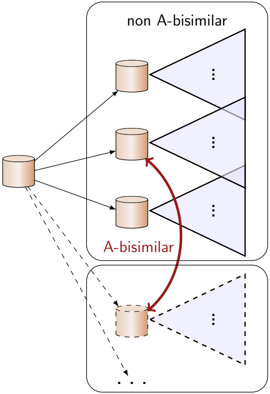
Decidability of $\mu\mathcal{L}$ extensions for run-bounded systems

Theorem

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

Idea: use **isomorphic types** instead of actual values.

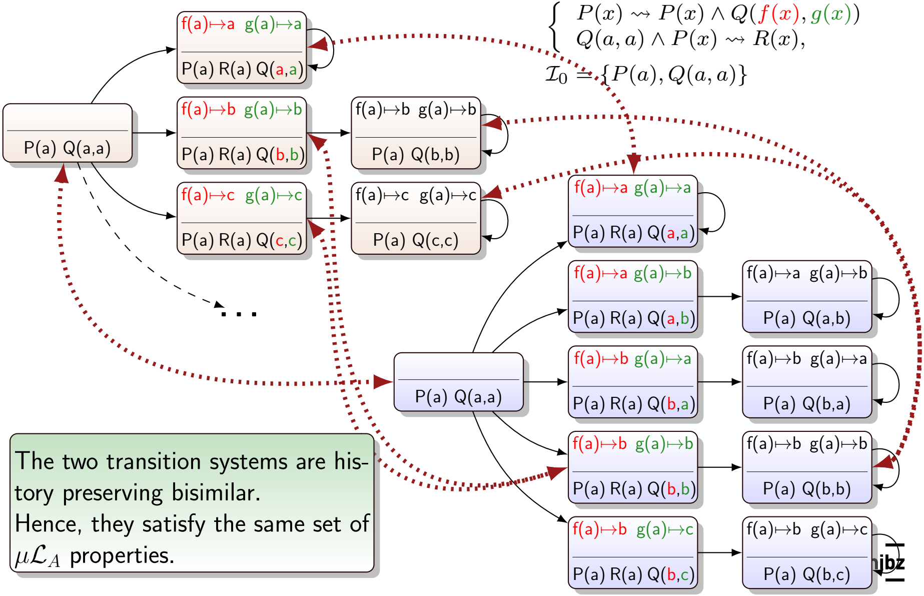
Remember: runs are bounded!



History preserving bisimulation

$$\begin{cases} P(x) \rightsquigarrow P(x) \land Q(f(x), g(x)) \\ Q(a,a) \land P(x) \rightsquigarrow R(x), \end{cases}$$

$$\mathcal{I}_0 = \{P(a), Q(a,a)\}$$



The two transition systems are history preserving bisimilar.
Hence, they satisfy the same set of $\mu\mathcal{L}_A$ properties.

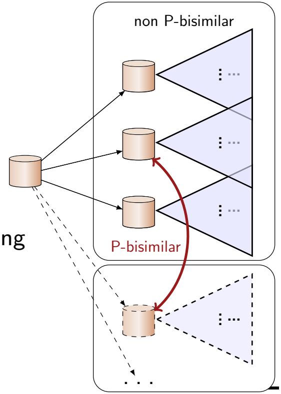
Decidability of $\mu\mathcal{L}$ extensions for state-bounded systems

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.

Steps:

1. **Prune** infinite branching (isomorphic types).
2. Finite abstraction along the runs:
   - $\mu\mathcal{L}_P$ looses track of previous values that do not exist anymore.
   - New values can be replaced with old, non-persisting ones.
   - This eventually leads to **recycle** the old values without generating new ones.



unibz

What about LTL-FO?

For verification of LTL-FO over DCDSs, analogous decidability results hold:

Theorem

Verification of LTL-FO$_A$ over run-bounded DCDSs, and
LTL-FO$_P$ over state-bounded DCDSs
are decidable and can be reduced to model checking of propositional LTL over a finite transition system.

Moreover:

Theorem

Verification of LTL-FO$_A$ over state-bounded DCDSs is undecidable.

Intuition: LTL-FO$_A$ can arbitrarily quantify over the infinitely many values encountered during a single run, and start comparing them.

Proof is based on a reduction from satisfiability of LTL with freeze quantifiers over infinite data words.

And verification of $\mu\mathcal{L}_A$ over state-bounded DCDSs?

Well-known

Propositional LTL can be expressed in $\mu\mathcal{L}$, i.e., the propositional μ-calculus.

Folklore "theorem" (see, e.g., [Okamoto 2010])

This correspondence carries over to the FO-variants, i.e., LTL-FO can be expressed in $\mu\mathcal{L}_{FO}$.

Note: This, together with the undecidability of LTL-FO_A verification over state-bounded DCDSs, would imply that also:

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

unibz

Verification of $\mu\mathcal{L}_{FO}$ over state-bounded DCDSs

Instead, the following positive result holds:

Theorem

Verification of $\mu\mathcal{L}_{FO}$ (and hence $\mu\mathcal{L}_{A}$) over state-bounded DCDSs is decidable.

Relies on the fact that DCDSs generate transition systems that are **generic**:

- Intuitively, if a state s has a successor state s' with fresh values $\vec{v}$, then it has also all successor states that are obtained from s' by varying in all possible ways the fresh values $\vec{v}$.
- This is a consequence of the fact that the progression mechanism is defined by means of a logical specification.

Lemma

- For generic TSs (with infinite domain), persistence-preserving bisimilarity and bisimilarity (and hence history-preserving bisimilarity) coincide.
- For TSs of state-bounded DCDSs, we can devise finite state abstractions that are faithful for $\mu\mathcal{L}_{FO}$ formulas (although such abstractions may depend on the formula).

Genericity

We consider isomorphisms $\sim_h$ between interpretations, where h is a bijection between the interpretation domains that preserves relations and constants.

Generic transition system

A TS Υ with domain Δ is **generic** if for all states s_1, s_2 and every bijection $h : \Delta \mapsto \Delta$, if $\mathcal{I}(s_1) \sim_h \mathcal{I}(s_2)$ and there exists s'_1 s.t. $s_1 \to s'_1$, then there exists s'_2 s.t. $s_2 \to s'_2$ and $\mathcal{I}(s'_1) \sim_h \mathcal{I}(s'_2)$.

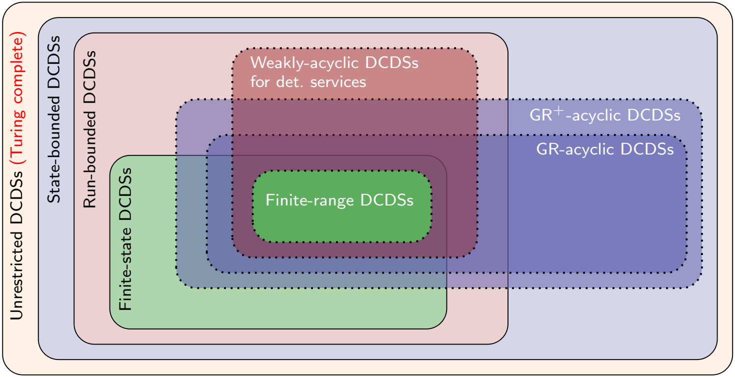
Note: s_1 and s_2 can be the same state, hence the existence of a successor state induces the existence of all successor states isomorphic to it.

DCDS enjoy genericity since:

- The progression mechanism is defined by means of a logical specification.
- In particular, the semantics of service calls induces the existence of a successor state for each combination of values returned by the service calls.

It follows that **successor states are "indistinguishable"** from each other, modulo isomorphisms on the results of service calls.

Results on decidability of verification for DCDSs



	Unrestricted	State-bounded	Run-bounded	Finite-state
LTL-FO / $\mu\mathcal{L}_{FO}$	**U**	**U** / **N**	? / **N**	**D**
LTL-FO$_A$ / $\mu\mathcal{L}_A$	**U**	**U** / **N**	**D**	**D**
LTL-FO$_P$ / $\mu\mathcal{L}_P$	**U**	**D**	**D**	**D**
LTL / $\mu\mathcal{L}$	**U**	**D**	**D**	**D**

D: decidable **U**: undecidable **N**: decidable, but no finite abstraction

Outline

unibz

Conclusions

- There is a huge amount of work carried out in database theory that is relevant to data-aware process analysis, using a plethora of techniques.
- The problem space has several dimensions that partly interact.
  $\leadsto$ Thorough systematization of the area is still missing.
- Many of the works are based on specific restrictions and assumptions that make them difficult to compare.
- Moreover, the positive results appear rather fragile.
- Analysis techniques are typically exponential in those data that "change"
  $\leadsto$ Circumscribing what can be changed is a key point.
- The assumptions would need validation also from the practical and business perspective.
  $\leadsto$ Requires making frameworks more robust.
- Some of the techniques are borrowed from different fields, although underlying assumptions and objectives might be different.
  $\leadsto$ Basic assumptions need to be reassessed.

Acknowledgements

Thanks to the many people who contributed interesting ideas, suggestions, discussions, and collaborated to the presented results.

Giuseppe De Giacomo
Marco Montali

Babak Bagheri Hariri
Riccardo De Masellis
Alin Deutsch
Paolo Felli
Rick Hull
Maurizio Lenzerini
Alessio Lomuscio
Fabio Patrizi
Ario Santoso
Moshe Vardi

unibz

Thank you for your attention!

References I

[1] Wil M. P. van der Aalst. "A Decade of Business Process Management Conferences: Personal Reflections on a Developing Discipline". In: *Proc. of the 10th Int. Conf. on Business Process Management (BPM)*. Vol. 7481. Lecture Notes in Computer Science. Springer, 2012, pp. 1–16.

[2] Javier Esparza. "Decidability of Model Checking for Infinite-State Concurrent Systems". In: *Acta Informatica* 34.2 (1997), pp. 85–107.

[3] Javier Esparza. "Decidability and Complexity of Petri Net Problems – An Introduction". In: *Lectures on Petri Nets I*. Lecture Notes in Computer Science. Springer, 1998, pp. 374–428.

[4] Ronald Fagin et al. "Data Exchange: Semantics and Query Answering". In: *Theoretical Computer Science* 336.1 (2005), pp. 89–124.

[5] Babak Bagheri Hariri et al. "Verification of Relational Data-Centric Dynamic Systems with External Services". In: *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*. 2013.

References II

[6] Keishi Okamoto. "Comparing Expressiveness of First-Order Modal μ-calculus and First-Order CTL*". In: *RIMS Kokyuroku* 1708 (2010), pp. 1–14.