Integrated Modeling and Verification of Processes and Data

Exploiting DCDSs: models, methods, concrete systems

Diego Calvanese, Marco Montali

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

15th International Conference on Business Process Management
Barcelona, Spain – 12 September 2017

The story so far, with main references

- The need of **combining (business) processes and data**.
  [Calvanese, De Giacomo, and Montali 2013]
- A pristine formalism for data-aware business processes: **DCDS**.
  [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013; Montali and Calvanese 2016]
- Suitable **verification logics** for data-aware processes.
  [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013; Calvanese, De Giacomo, Montali, and Patrizi 2017]
- Corresponding **characterization theorems**.
  [Calvanese, De Giacomo, Montali, and Patrizi 2017]
- A **decidability map**, with an unexpected dichotomy between $\mu\mathcal{L}_A$ and LTL-FO$_A$.
  [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013; Calvanese, De Giacomo, Montali, and Patrizi 2017]

Note: Incorrect results in [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013; Okamoto 2010] fixed in [Calvanese, De Giacomo, Montali, and Patrizi 2017].

How to check/ensure state boundedness?

Theorem

Checking whether a DCDS is **state-/run-bounded** is:

- **Decidable** for a **given** bound.
- **Undecidable** for an **unknown** bound.

Three possible strategies:

- Single out **classes of DCDSs** for which checking state-/run-boundedness is decidable.
- Identify **sufficient syntactic conditions** that are decidable to check, and that guarantee state-/run-boundedness
  - cf. syntactic conditions for chase termination in data exchange.
- Devise **modeling methodologies** that guarantee state boundedness.

How to check/ensure state boundedness?

Theorem

Checking whether a DCDS is **state-/run-bounded** is:

- **Decidable** for a **given** bound.
- **Undecidable** for an **unknown** bound.

Three possible strategies:

- Single out **classes of DCDSs** for which checking state-/run-boundedness is decidable.
- Identify **sufficient syntactic conditions** that are decidable to check, and that guarantee state-/run-boundedness
  - cf. syntactic conditions for chase termination in data exchange.
- Devise **modeling methodologies** that guarantee state boundedness.

DCDSs with decidable state-boundedness

Fact

DCDSs using only unary relations correspond to variants of Petri nets.

- The specific variant depends on the features used in the DCDS.

Note: State-boundedness relate to boundedness in Petri nets.

Petri nets with name management

Decidable boundedness.
[Rosa-Velardo and Frutos-Escrig 2011]

Reset-Transfer Nets

Undecidable boundedness.
[Dufourd, Jancar, and Schnoebelen 1999]

[Montali and Rivkin 2016]

Translation to DCDSs and $\mu\mathcal{L}_P$ verification.

[Bagheri Hariri, Calvanese, Deutsch, et al. 2014]

"Lossy" correspondence with DCDSs.

DCDSs with decidable state-boundedness

Fact

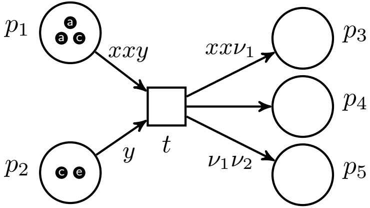
DCDSs using only unary relations correspond to variants of Petri nets.

- The specific variant depends on the features used in the DCDS.

Note: State-boundedness relate to boundedness in Petri nets.

Petri nets with **name management**

Decidable boundedness.
[Rosa-Velardo and Frutos-Escrig 2011]



Reset-Transfer Nets

Undecidable boundedness.
[Dufourd, Jancar, and Schnoebelen 1999]

[Montali and Rivkin 2016]

Translation to DCDSs and $\mu\mathcal{L}_P$ verification.

[Bagheri Hariri, Calvanese, Deutsch, et al. 2014]

"Lossy" correspondence with DCDSs.

DCDSs with decidable state-boundedness

Fact

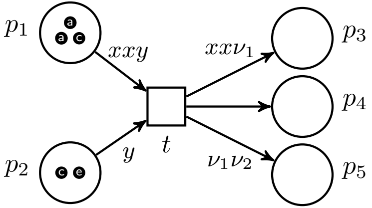
DCDSs using only unary relations correspond to variants of Petri nets.

- The specific variant depends on the features used in the DCDS.

Note: State-boundedness relate to boundedness in Petri nets.

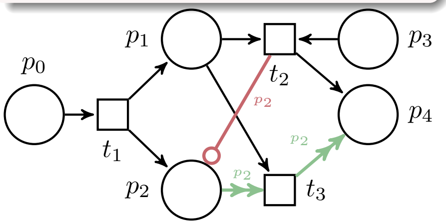
Petri nets with **name management**

Decidable boundedness.
[Rosa-Velardo and Frutos-Escrig 2011]



Reset-Transfer Nets

Undecidable boundedness.
[Dufourd, Jancar, and Schnoebelen 1999]



[Montali and Rivkin 2016]

Translation to DCDSs and $\mu\mathcal{L}_P$ verification.

[Bagheri Hariri, Calvanese, Deutsch, et al. 2014]

"Lossy" correspondence with DCDSs.

Attacking state-boundedness

The class of DCDSs with decidable state-boundedness very restrictive

These variants of Petri nets corresponds to DCDSs with only unary relations, limited use of negation, no or limited joins, . . .

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

Sufficient, **syntactic conditions**:

- Extract a data flow graph from the DCDS.
- Check sources of unboundedness through this graph.

See [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013] and [Bagheri Hariri, Calvanese, Deutsch, et al. 2014].

State-boundedness **by design**:

Design methods for state-bounded DCDSs. In [Solomakhin et al. 2013]:

- 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.

More sophisticated techniques in [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014].

Attacking state-boundedness

The class of DCDSs with decidable state-boundedness very restrictive

These variants of Petri nets corresponds to DCDSs with only unary relations, limited use of negation, no or limited joins, ...

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

Sufficient, **syntactic conditions**:

- Extract a data flow graph from the DCDS.
- Check sources of unboundedness through this graph.

See [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013] and [Bagheri Hariri, Calvanese, Deutsch, et al. 2014].

State-boundedness **by design**:

Design methods for state-bounded DCDSs. In [Solomakhin et al. 2013]:

- 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.

More sophisticated techniques in [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014].

Attacking state-boundedness

The class of DCDSs with decidable state-boundedness very restrictive

These variants of Petri nets corresponds to DCDSs with only unary relations, limited use of negation, no or limited joins, ...

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

Sufficient, syntactic conditions:

- Extract a data flow graph from the DCDS.
- Check sources of unboundedness through this graph.

See [Bagheri Hariri, Calvanese, De Giacomo, et al. 2013] and [Bagheri Hariri, Calvanese, Deutsch, et al. 2014].

State-boundedness by design:

Design methods for state-bounded DCDSs. In [Solomakhin et al. 2013]:

- 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.

More sophisticated techniques in [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014].

State-boundedness in concrete process modeling languages

Classical BPM languages/suites

- Central notion of **case** representing a process instance.
- Each case carries its own case data, in isolation to the other cases (e.g., order details, customer address, ...).
- Cases interact by accessing a central, persistent data storage.

Artifact-centric approaches:

- Central notion of **business object** gluing data and behaviour together.
- All data relevant to a business object are attached to it.
- Processes may query multiple business objects at once, to determine the possible next steps.

External and internal stakeholders...

- New cases/business objects are created upon **events** issued by external stakeholders (e.g., new order request).
- But then they are bound to internal resources, responsible for progressing the corresponding process instances.

State-boundedness in concrete process modeling languages

Classical BPM languages/suites

- Central notion of **case** representing a process instance.
- Each case carries its own case data, in isolation to the other cases (e.g., order details, customer address, . . .).
- Cases interact by accessing a central, persistent data storage.

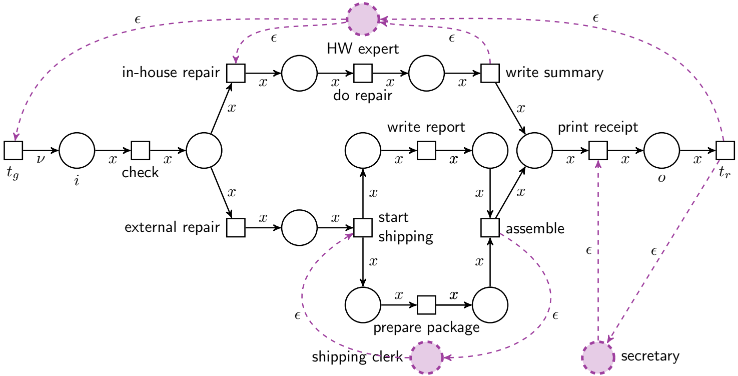
Artifact-centric approaches:

- Central notion of **business object** gluing data and behaviour together.
- All data relevant to a business object are attached to it.
- Processes may query multiple business objects at once, to determine the possible next steps.

External and internal stakeholders. . .

- New cases/business objects are created upon **events** issued by external stakeholders (e.g., new order request).
- But then they are bound to internal resources, responsible for progressing the corresponding process instances.

RIAW-nets [Montali and Rivkin 2016]



RIAW-nets = ν-PNs + workflow nets

- Emitter transition generating a **new process id** when fired.
- Control-flow name matching to **selectively spawn/synch** tokens using their id.
- Resource places to **bound the number of simultaneously coexisting active process instances**! (but **unboundedly many over time**).

Decidability of model checking via translation to state-bounded DCDSs.

Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

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

Modeling guidelines to guarantee data isolation and relative boundedness:

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 (no construction of chains).

Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

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`

Modeling guidelines to guarantee data isolation and relative boundedness:

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 (no construction of chains).

Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

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

Modeling guidelines to guarantee data isolation and relative boundedness:

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 (no construction of chains).

Data isolation and case unboundedness

What if the number of simultaneously active cases cannot be bounded?

In [Montali and Calvanese 2016; Calvanese, Montali, et al. 2014], we show that **decidability** of model checking can be retained, if the system obeys to:

- **relative boundedness** (each case manipulates boundedly many data);
- **data isolation** (cases interact very weakly).

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

Modeling guidelines to guarantee data isolation and relative boundedness:

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 (no construction of chains).

Beyond State-Boundedness

Question

Are there classes of DCDSs that are **unbounded**, but still **amenable to verification**?

Key result in [Abdulla et al. 2016].

Recency-bounded data-aware processes

Unbounded DB, but only the latest inserted/accessed values can bound to parameters.

Verification via under-approximation

Decidability by focusing only on runs that are k-recency-bounded for an explicitly given key.

Open problem

Investigate the relationships between all such results and those where the initial DB is **not fixed**, and verification is studied **for every possible** initial DB.

Beyond State-Boundedness

Question

Are there classes of DCDSs that are **unbounded**, but still **amenable to verification**?

Key result in [Abdulla et al. 2016].

Recency-bounded data-aware processes

Unbounded DB, but only the latest inserted/accessed values can bound to parameters.

Verification via under-approximation

Decidability by focusing only on runs that are k-recency-bounded for an explicitly given key.

Open problem

Investigate the relationships between all such results and those where the initial DB is **not fixed**, and verification is studied **for every possible** initial DB.

Beyond State-Boundedness

Question

Are there classes of DCDSs that are **unbounded**, but still **amenable to verification**?

Key result in [Abdulla et al. 2016].

Recency-bounded data-aware processes

Unbounded DB, but only the latest inserted/accessed values can bound to parameters.

Verification via under-approximation

Decidability by focusing only on runs that are k-recency-bounded for an explicitly given key.

Open problem

Investigate the relationships between all such results and those where the initial DB is **not fixed**, and verification is studied **for every possible** initial DB.

Incorporation of datatypes

Databases have **datatypes**

Numeric domains, domain-specific predicates, arithmetic.

- Many coordination algorithms and auctions require dense orders.
- Processes with costs and payment policies require integers and arithmetic.

Dense orders combine well with state-boundedness

Data-aware, state-bounded distributed systems with reals [Calvanese, Delzanno, and Montali 2015]:

- **OK** to include **dense linear orders**: minor extension to the standard DCDS abstraction technique. Intuition...

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

- **No hope** to include the **successor** relation (or integers): 2 data slots are sufficient to encode two counters.

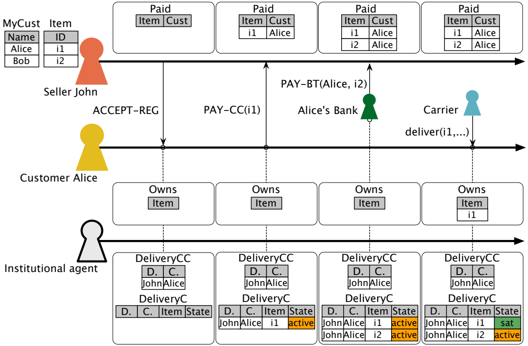
Discrete orders and arithmetic combine well with run-boundedness

Ongoing work...

Relational multiagent systems and commitments

Relational MAS [Montali, Calvanese, and De Giacomo 2014]

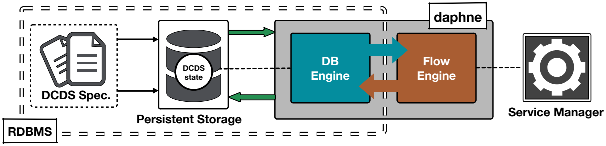
- Agents have names and hold/manipulate local, state-bounded DBs.
- Agents exchange data using their names for addressing.
- An institutional agent manages agent creation and deletion.
  - Due to state-boundedness: unboundedly many agents can dynamically enter into the system, but at each moment only boundedly many are active.



Relational commitments

In the same work: first proposal for modeling and verifying interaction protocols based on relational commitments, i.e., commitments with data payload and multiple instances.

daphne: implementing DCDSs with relational technology



Native **modeling and execution** of DCDSs using relational DBMSs:

- **SQL-like syntax** for DCDSs with datatypes.
- Automated translation into **relational DBMSs**, as (temporal) tables, constraints, and stored procedures.
- **Java APIs** to support enactment and integration with concrete services.

Native **explicit model checking** of DCDSs using relational DBMSs:

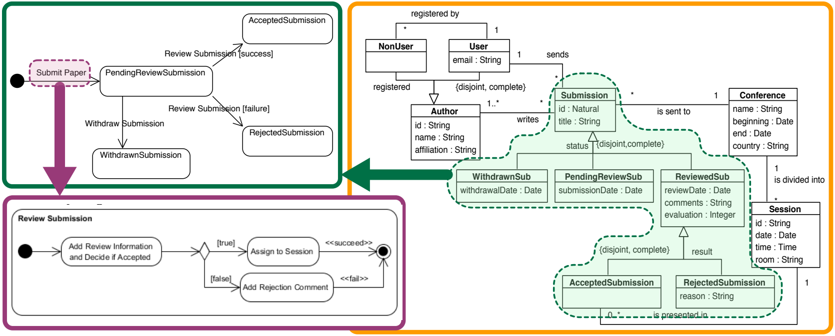
- **Same model for execution and verification!**
- **Special tables** for storing the RTS induced by a DCDSs.
- **Factoring of tables** into temporal and atemporal parts.
- **Computation** of isomorphic type and value recycling in services.
- **Java APIs** for RTS construction and search.

Can we cook with all ingredients?

"REAL" PROCESS

- Explicit control-flow
- Local, case data
- Global, persistent data
- Queries/updates on the persistent data
- External inputs
- Internal generation of fresh IDs

BAUML: artifact-centric processes with UML

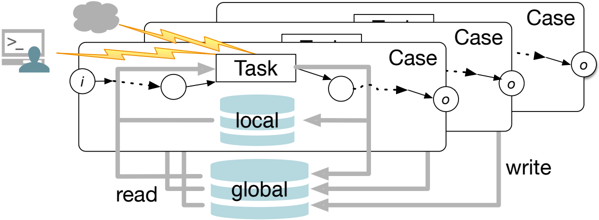


BAUML approach

- **Business objects**, states, associations and attributes: UML class diagrams.
- Business object **lifecycle**: UML statechart diagram.
- **Complex event** triggering a lifecycle transition: UML activity diagram.
  - **Tasks** modeled as OCL operation contracts.

In [Calvanese, Montali, et al. 2014]: **methodology to guarantee decidability of model checking** (see before). Estanol PhD thesis: BAUML to DCDS!

raw-sys: marrying workflow nets and databases



raw-sys model [De Masellis et al. 2017]:

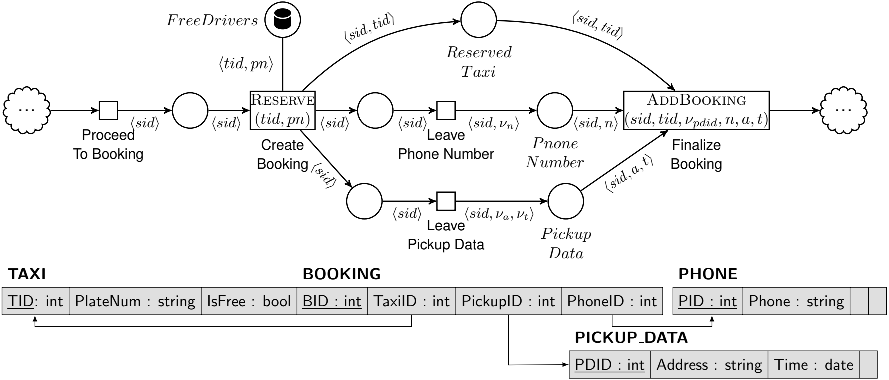
Data-aware processes using **well-known formalisms**:

- **Data**: global and local relational databases.
- **Process control-flow**: workflow nets, enriched with:
  - Guards (queries over the DBs).
  - STRIPS-like actions with external inputs from an infinite domain, invoked upon firing net transitions.

raw-sys verification [De Masellis et al. 2017]:

- Map of **(un)decidability**, exploiting **translation to DCDSs**.
- **Encoding into planning** systems to handle reachability problems.

db-nets: marrying colored Petri nets and databases

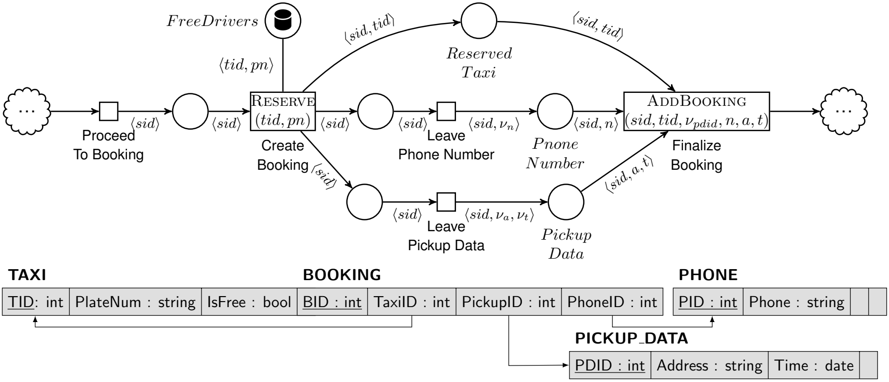


db-net **model** [Montali and Rivkin 2017], three layers:

1. **Persistence**: relational database with constraints.
2. **Data logic**: queries and actions over the persistence layer.
3. **Control**: colored Petri net with ν-variables, enriched with **view places** and **transition-action bindings** to inspect/update the persistence layer.

Note: Natural formalization of contemporary process modeling suites!

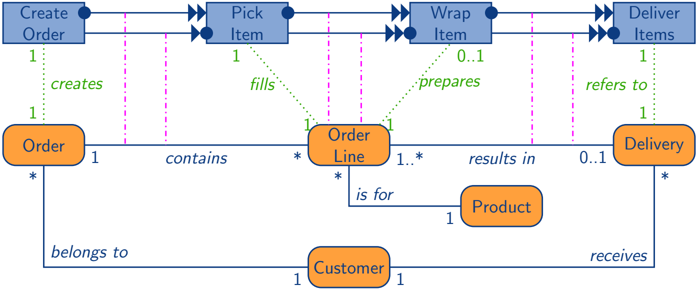
db-nets: marrying colored Petri nets and databases



db-nets **execution, simulation, verification** [Montali and Rivkin 2017]:

- Foundational results thanks to **translation to DCDSs**.
- Ongoing implementation effort inside `www.cpntools.org`.

OCBC: declarative data+process integrated model



OCBC **model** [Artale et al. 2017], three components:

1. **Data model**: UML class diagram.
2. **Tasks**: units of work, referencing classes in the data model. Each task instance comes with objects belonging to such classes.
3. **Behavioral constraints**: declarative patterns equipped with coreference relations pointing to the data model. They constrain when tasks can be executed, and which data objects they should carry.

Naturally captures many-to-many processes with no single notion of case!

Acknowledgements

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

Giuseppe De Giacomo
Fabio Patrizi

Babak Bagheri Hariri
Riccardo De Masellis
Alin Deutsch
Marlon Dumas
Paolo Felli
Rick Hull
Maurizio Lenzerini
Alessio Lomuscio
Andy Rivkin
Ario Santoso

Thank you for your attention!

References I

[1] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. "Foundations of Data-Aware Process Analysis: A Database Theory Perspective". In: *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*. ACM Press, 2013, pp. 1–12.

[2] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, 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)*. Extended version available at `http://arxiv.org/abs/1203.0024`. 2013, pp. 163–174.

[3] Marco Montali and Diego Calvanese. "Soundness of Data-Aware, Case-Centric Processes". In: *Int. J. on Software Tools for Technology Transfer* (2016). DOI: 10.1007/s10009-016-0417-2.

[4] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. "First-Order mu-Calculus over Generic Transition Systems and Applications to the Situation Calculus". In: *Information and Computation* (2017). To appear.

References II

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

[6] Fernando Rosa-Velardo and David de Frutos-Escrig. "Decidability and Complexity of Petri Nets with Unordered Data". In: *Theoretical Computer Science* 412.34 (2011), pp. 4439–4451.

[7] Marco Montali and Andrey Rivkin. "Model Checking Petri Nets with Names Using Data-Centric Dynamic Systems". In: *Formal Aspects of Computing* (2016), pp. 1–27.

[8] Catherine Dufourd, Petr Jancar, and Ph. Schnoebelen. "Boundedness of Reset P/T Nets". In: *Proc. of the 26th Int. Coll. on Automata, Languages and Programming (ICALP)*. Vol. 1644. Lecture Notes in Computer Science. Springer, 1999, pp. 301–310.

[9] Babak Bagheri Hariri, Diego Calvanese, Alin Deutsch, et al. "State-Boundedness in Data-Aware Dynamic Systems". In: *Proc. of the 14th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 2014.

References III

[10] Dmitry Solomakhin et al. "Verification of Artifact-Centric Systems: Decidability and Modeling Issues". In: vol. 8274. Lecture Notes in Computer Science. Springer, 2013, pp. 252–266.

[11] Diego Calvanese, Marco Montali, et al. "Verifiable UML Artifact-Centric Business Process Models". In: *Proc. of the 23rd Int. Conf. on Information and Knowledge Management (CIKM)*. 2014, pp. 1289–1298. DOI: 10.1145/2661829.2662050.

[12] Parosh Aziz Abdulla et al. "Recency-Bounded Verification of Dynamic Database-Driven Systems". In: *Proc. of the 35th ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*. ACM Press, 2016.

[13] Diego Calvanese, Giorgio Delzanno, and Marco Montali. "Verification of Relational Multiagent Systems with Data Types". In: *Proc. of the 29th AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 2015, pp. 2031–2037.

References IV

[14] Marco Montali, Diego Calvanese, and Giuseppe De Giacomo. "Verification of Data-Aware Commitment-Based Multiagent System". In: *Proc. of the 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS, 2014, pp. 157–164.

[15] Riccardo De Masellis et al. "Add Data into Business Process Verification: Bridging the Gap between Theory and Practice". In: *Proc. of the 31st AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 2017, pp. 1091–1099. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14627`.

[16] Marco Montali and Andrey Rivkin. "DB-Nets: on The Marriage of Colored Petri Nets and Relational Databases". In: *LNCS Transactions on Petri Nets and Other Models of Concurrency* (2017). To appear.

[17] Alessandro Artale et al. "Object-Centric Behavioral Constraints: Integrating Data and Declarative Process Modelling". In: *Proc. of the 30th Int. Workshop on Description Logics (DL)*. Ed. by Alessandro Artale, Birte Glimm, and Roman Kontchakov. CEUR Workshop Proceedings, `http://ceur-ws.org/`, 2017.