Verification of Data-Aware Processes

Boundaries of Decidability: Positive Results

Diego Calvanese, Marco Montali

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

29th European Summer School in Logic, Language, and Information
(ESSLLI 2017)
Toulouse, France – 17–28 July 2017

Outline

Understanding and comparing DCDSs

Before moving into verification, we need to understand how to characterize the (branching) **behavior** induced by a DCDS.

- How to compare the behaviors induced by two DCDSs?
- How does behavioral equivalence relate with satisfaction of verification formulae?

In the propositional case, the main tool for answering such questions is that of **bisimulation**.

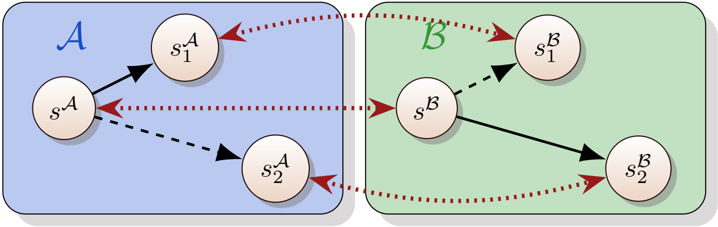
A crash course on bisimulation

Bisimulation between propositional transition systems

Consider two propositional transition systems $\mathcal{A} = \langle S^{\mathcal{A}}, s_0^{\mathcal{A}}, prop^{\mathcal{A}}, \Rightarrow_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle S^{\mathcal{B}}, s_0^{\mathcal{B}}, prop^{\mathcal{B}}, \Rightarrow_{\mathcal{B}} \rangle$. Two states $s^{\mathcal{A}} \in S^{\mathcal{A}}$ and $s^{\mathcal{B}} \in S^{\mathcal{B}}$ **bisimilar** if:

1. $s^{\mathcal{A}}$ and $s^{\mathcal{B}}$ are isomorphic (local condition).
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 (forth c.).
3. The other direction (back condition).

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



Two fundamental theorems

Consider two propositional transition systems $\mathcal{A}$ and $\mathcal{B}$.

Theorem

If $\mathcal{A}$ and $\mathcal{B}$ are bisimilar, then they satisfy exactly the same $\mu\mathcal{L}$ properties.

Intuitively, $\mu\mathcal{L}$ is **not able to distinguish** bisimilar transition systems.

Theorem

If $\mathcal{A}$ and $\mathcal{B}$ satisfy exactly the same $\mu\mathcal{L}$ properties, then they are bisimilar.

Intuitively, $\mu\mathcal{L}$ is the **maximal** logic that **captures bisimulation**.

Correspondence Theorems for DCDSs

Can we lift these fundamental correspondence theorems to the case of DCDSs?

In the general case, we are doomed, since relational transition systems are simply too rich.

We proceed as follows:

1. We single out key properties of the RTSs induced by DCDSs.
2. We introduce suitable notions of bisimulations for the FO temporal logics introduced before.
3. We reconstruct correspondence theorems.

Two key properties of DCDSs

We have already seen the two properties of DCDSs to exploit:

- Markovian, i.e., the next state only depends on the current state and the input.
- Based on generic queries, which do not distinguish structures that are identical modulo uniform renaming of (new) data objects.

DCDSs are **generic**, which implies that, modulo isomorphisms on the results of service calls, successor states are "indistinguishable" from each other.

Bisimulation between RTSs

Consider Υ_1, Υ_2 over **disjoint data domains** Δ_1, Δ_2, with states S_1, S_2.

A **bisimulation** between Υ_1 and Υ_2

- is a binary relation connecting pairs of states under a global **bijection**.
- In particular, $\approx \subseteq S_1 \times S_2$ is a bisimulation between Υ_1 and Υ_2 if there exists a bijection $h : \Delta_1 \mapsto \Delta_2$ such that $s_1 \approx s_2$ implies that:
  1. h induces an **isomorphism** between $db_1(s_1)$ and $db_2(s_2)$;
  2. for each s_1', if $s_1 \Rightarrow_1 s_1'$ then there is an s_2' with $s_2 \Rightarrow_2 s_2'$ s.t. $s_1' \approx s_2'$;
  3. the other direction.
- $\Upsilon_1 \approx \Upsilon_2$ if $s_{01} \approx s_{02}$.

The classical result on indistinguishability of bisimilar TSs by $\mu\mathcal{L}$ formulas extends to $\mu\mathcal{L}_{FO}$.

Theorem

If $\Upsilon_1 \approx \Upsilon_2$, then for every $\mu\mathcal{L}_{FO}$ closed formula Φ, we have that:

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

Outline

Weakening the bisimulations

The notion of **bisimulation** as just defined is suitable for $\mu\mathcal{L}_{FO}$ (and LTL-FO), but is **too strong** for our purposes.

Note:

- $\mu\mathcal{L}_{FO}$ allows for quantifying over the whole domain.
  $\rightsquigarrow$ Captured by the global bijection in the definition of bisimulation.
- In $\mu\mathcal{L}_{A}$, instead we can quantify only over the active domain of the current state, and the evolution of its elements over time.
  $\rightsquigarrow$ The bijection should consider the history so far plus the new objects.
- In $\mu\mathcal{L}_{P}$, we can quantify only over the objects that persist.
  $\rightsquigarrow$ The bijection should consider elements that persist in the state.

We suitably adjust the definition of bisimulation to reflect these restrictions.

History-preserving bisimulation

Consider Υ_1, Υ_2 over **disjoint data domains** Δ_1, Δ_2, with states S_1, S_2.
Let H be the set of all partial bijections between Δ_1 and Δ_2.

A history-preserving bisimulation between Υ_1 and Υ_2

- is a ternary relation $\approx^A \subseteq S_1 \times H \times S_2$, connecting pairs of states under a **bijection** that tracks the history.
- In particular, $\langle s_1, h, s_2 \rangle \in \approx_h^A$, denoted $s_1 \approx_h^A s_2$, implies that:
  1. $h \in H$ induces an **isomorphism** between $db_1(s_1)$ and $db_2(s_2)$;
  2. for each s_1', if $s_1 \Rightarrow_1 s_1'$ then there is an s_2' with $s_2 \Rightarrow_2 s_2'$ and a bijection h' that **extends** h, such that $s_1' \approx_{h'}^A s_2'$;
  3. the other direction.
- $\Upsilon_1 \approx^A \Upsilon_2$ if there exists a partial bijection h_0 such that $s_{01} \approx_{h_0}^A s_{02}$.

Theorem

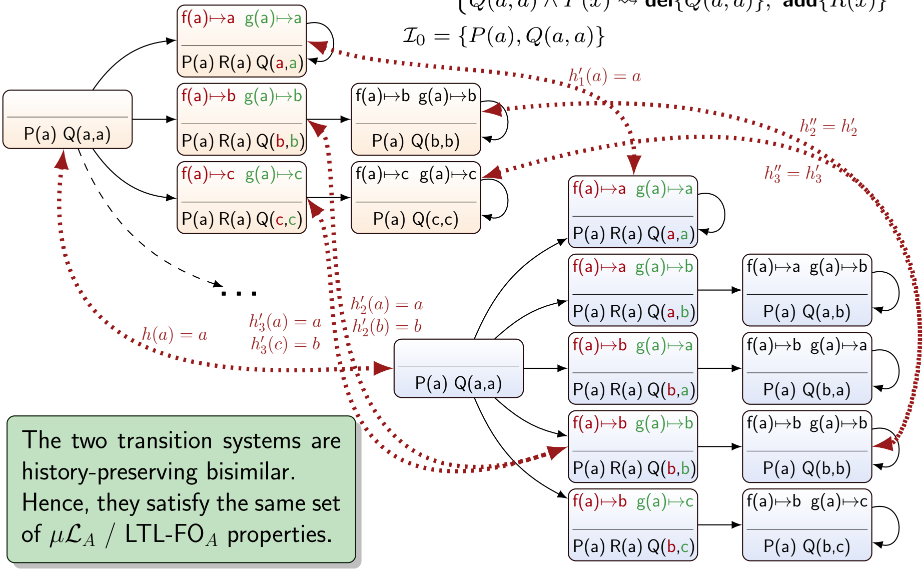
If $\Upsilon_1 \approx^A \Upsilon_2$, then for every $\mu\mathcal{L}_A$ closed formula Φ, we have that:

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

History-preserving bisimulation

$$a() : \begin{cases} P(x) \rightsquigarrow \textbf{del}\{R(x)\},\ \textbf{add}\{Q(f(x), g(x))\} \\ Q(a,a) \wedge P(x) \rightsquigarrow \textbf{del}\{Q(a,a)\},\ \textbf{add}\{R(x)\} \end{cases}$$

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



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

Persistence-preserving bisimulation

Consider Υ_1, Υ_2 over **disjoint data domains** Δ_1, Δ_2, with states S_1, S_2.
Let H be the set of all partial bijections between Δ_1 and Δ_2.

A **persistence-preserving bisimulation** between Υ_1 and Υ_2

- is a ternary relation $\approx^P \subseteq S_1 \times H \times S_2$, connecting pairs of states under a **bijection** that tracks the history of persisting objects.
- In particular, $\langle s_1, h, s_2 \rangle \in \approx^P_h$, denoted $s_1 \approx^P_h s_2$ implies that:
  1. $h \in H$ induces an **isomorphism** between $db_1(s_1)$ and $db_2(s_2)$;
  2. for each s'_1, if $s_1 \Rightarrow_1 s'_1$ then there exists an s'_2 with $s_2 \Rightarrow_2 s'_2$ and a bijection h' that **extends h restricted on $\text{ADOM}(db_1(s_1)) \cup \text{ADOM}(db_1(s'_1))$**, such that $s'_1 \approx^P_{h'} s'_2$;
  3. the other direction.
- $\Upsilon_1 \approx^P \Upsilon_2$ if there exists a partial bijection h_0 such that $s_{01} \approx^P_{h_0} s_{02}$.

Theorem

If $\Upsilon_1 \approx^P \Upsilon_2$, then for every $\mu\mathcal{L}_P$ closed formula Φ, we have that:

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

Genericity, Bisimulation Collapse

The different bisimulations are tightly related to the logic variants that we have introduced.

Consider two RTSs $\Upsilon_1 = \langle \Delta_1, \mathcal{R}, S_1, q_{10}, db_1, \Rightarrow_1 \rangle$ and
$\Upsilon_2 = \langle \Delta_2, \mathcal{R}, S_2, q_{20}, db_2, \Rightarrow_2 \rangle$
with $|\Delta_1| = |\Delta_2|$ infinite, a state s_1 of T_1, and a state s_2 of T_2.

Let $s_1 \equiv_{\mu\mathcal{L}_{FO}} s_2$ denote that states s_1 and s_2 satisfy the same $\mu\mathcal{L}_{FO}$ formulas, analogously for $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$.

Finite-active-domain transition system

Is a RTS such that the active domain of every state is finite (though not necessarily bounded by some given b).

Genericity, Bisimulation Collapse, and $\mu\mathcal{L}_{FO}$ Variants

The following always hold:

$s_1 \approx s_2$	implies	$s_1 \approx^A s_2$	implies	$s_1 \approx^P s_2$
$s_1 \approx^P s_2$	implies	$s_1 \equiv_{\mu\mathcal{L}_P} s_2$		
$s_1 \approx^A s_2$	implies	$s_1 \equiv_{\mu\mathcal{L}_A} s_2$		
$s_1 \approx s_2$	implies	$s_1 \equiv_{\mu\mathcal{L}_{FO}} s_2$		
$s_1 \equiv_{\mu\mathcal{L}_{FO}} s_2$	implies	$s_1 \equiv_{\mu\mathcal{L}_A} s_2$	implies	$s_1 \equiv_{\mu\mathcal{L}_P} s_2$

When T_1 and T_2 are generic:

$s_1 \approx^P s_2$ equivalent $s_1 \approx^A s_2$ equivalent $s_1 \approx s_2$

When T_1 and T_2 are generic and finite-active-domain:

$s_1 \equiv_{\mu\mathcal{L}_P} s_2$	equivalent	$s_1 \approx^P s_2$		
$s_1 \equiv_{\mu\mathcal{L}_A} s_2$	equivalent	$s_1 \approx^A s_2$		
$s_1 \equiv_{\mu\mathcal{L}_{FO}} s_2$	equivalent	$s_1 \approx s_2$		
$s_1 \equiv_{\mu\mathcal{L}_P} s_2$	equivalent	$s_1 \equiv_{\mu\mathcal{L}_A} s_2$	equivalent	$s_1 \equiv_{\mu\mathcal{L}_{FO}} s_2$

Outline

Summary of results so far

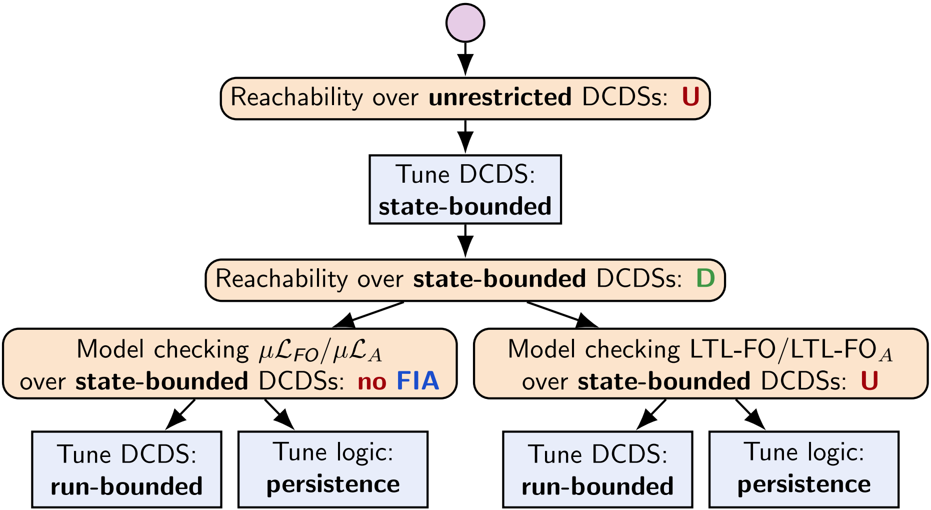
We have seen the following results:

- Without restrictions on the form of the DCDS, even the simplest properties (reachability) is undecidable.
  $\leadsto$ Towards decidability, we deal only with **state bounded DCDSs** and with logics with **active domain quantification** ($\mu\mathcal{L}_A$, LTL-FO$_A$).
- Even for state bounded DCDS, we have that:
  - Model checking LTL-FO$_A$ (and hence LTL-FO) is undecidable.
  - Model checking $\mu\mathcal{L}_A$ does not admit formula-independent abstractions.

To overcome these problems, we can follow different approaches:

- We consider a further restriction on DCDSs: run-boundedness (is only meaningful under deterministic services semantics).
- We consider a further restriction on the logics: $\mu\mathcal{L}_P$ and LTL-FO$_P$.
- We study formula-dependent abstractions.

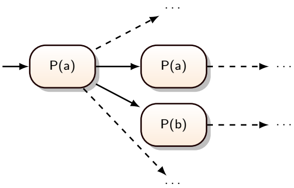
Current overall picture



Towards decidability

We need to tame the two sources of infinity in the RTS $\Upsilon_{\mathcal{X}}$ generated by a DCDS $\mathcal{X}$:

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



To prove decidability of model checking for restricted DCDSs and a specific verification logic $\mathcal{L}$:

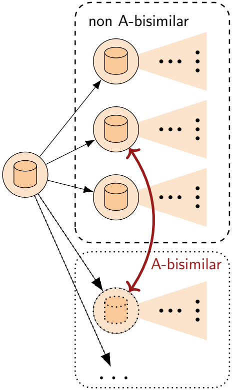
- We use as a tool bisimulations for the logic $\mathcal{L}$.
- We show that we can construct a finite-state RTS $\Theta_{\mathcal{X}}$ that provides a faithful abstraction of $\Upsilon_{\mathcal{X}}$ for formulas of $\mathcal{L}$.

  In other words, $\Theta_{\mathcal{X}}$ and $\Upsilon_{\mathcal{X}}$ are bisimilar, under the bisimulation for $\mathcal{L}$.

Outline

Dealing with infinite branching

- Infinite branching is caused by the infinite number of possible combinations of values returned by the service calls.
- Notice, however, that for each state along a run:
  - only a finite number of values have been encountered so far, and
  - only a finite number of service calls are issued when an action is executed.
- Hence, due to genericity, we need only to take into account:
  - whether a new value is equal to or differs from a value encountered so far;
  - whether new values obtained from different service calls are equal to or differ from each other.



⇝ *Note:* Instead of actual values, use **isomorphic types** based on equality commitments.

Equality commitments

Consider a set D consisting of:

- constants, and
- terms obtained by applying functions to constants (i.e., service calls).

Equality commitment (EqC) $\mathcal{H}$ on D

is a partition of D such that each element of the partition contains at most one constant (but arbitrarily many terms).

Note: each equality commitment $\mathcal{H}$ induces an equality relation $=_{\mathcal{H}}$ on the elements of D.

Given a state s of $\Upsilon_{\mathcal{X}}$ with DB $\mathcal{I}$, we consider now EqCs on

- $\text{ADOM}(\mathcal{I}) \cup \text{ADOM}(\mathcal{I}_0)$ as the set of constants, and
- $\text{CALLS}(\overline{\mathcal{I}})$ as the set of terms.

Note: there are only **finitely many such EqCs**.

Equality commitments and pruning

A service call evaluation θ respects an EqC $\mathcal{H}$

if for every two terms t_1, t_2, we have that $t_1\theta = t_2\theta$ if and only if $t_1 =_{\mathcal{H}} t_2$.

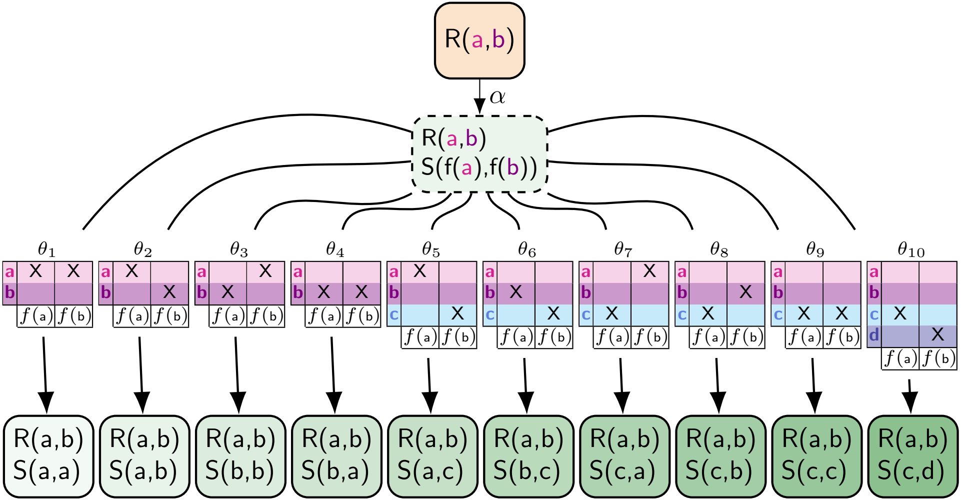
For an action α and parameter evaluation σ, consider now all successors of state s according to an EqC $\mathcal{H}$:

- For each θ that respects $\mathcal{H}$, state s has one successor $\text{DO}(\mathcal{I}, \alpha, \sigma, \theta)$.
- All such successors are isomorphic. Hence each EqC $\mathcal{H}$ determines an isomorphism type.
- We can now prune all isomorphic successors except one, which is kept as representative of the isomorphism type.

Equality commitments – Example

Consider action α with no params and using a nondeterministic service call f:

$$\alpha{\cdot}\mathrm{EFF} = \{R(x,y) \rightsquigarrow \textbf{add}\{S(f(x),f(y))\}\}$$



Constructing a finite branching abstraction

Theorem

Let $\Theta_\mathcal{X}$ be the RTS obtained from $\Upsilon_\mathcal{X}$ by pruning successor nodes according to equality commitments. Then:

- $\Theta_\mathcal{X}$ is **finite branching**.
- $\Theta_\mathcal{X}$ and $\Upsilon_\mathcal{X}$ are **persistence-preserving bisimilar**.

Note:

- In the construction of $\Theta_\mathcal{X}$, we have computed EqCs by considering as constants only the elements of the active domains of the current state and of the initial state s_0.
- Instead, if we determine EqCs by considering as constants all values along the history, then:
  - $\Theta_\mathcal{X}$ is still finite branching.
  - $\Theta_\mathcal{X}$ and $\Upsilon_\mathcal{X}$ are history-preserving bisimilar.

Outline

Dealing with infinite runs

We still need to address infiniteness of the RTS coming from possibly **infinite runs**, which may accumulate infinitely many new values along the run.

Two approaches to deal with this:

1. Restrict the DCDS, by ruling out a priori the accumulation of infinitely many values along a run.
   $\leadsto$ run-bounded DCDSs
2. Restrict the logics, making them "insensitive" to the infinitely many values.
   $\leadsto$ persistence-preserving variants of $\mu\mathcal{L}_{FO}$ and LTL-FO

Recall: the DCDSs we consider are state-bounded!

Run-boundedness

A DCDS $\mathcal{X}$ is **run-bounded**

if there exists a fixed number b such that the number of values used **in each (infinite) run** of $\mathcal{X}$, is **bounded by** b: given $\Upsilon_{\mathcal{X}} = \langle \Delta, \mathcal{R}, S, s_0, db, \Rightarrow \rangle$, for each sequence $s_0, s_1, s_2, \ldots$ such that $s_i \Rightarrow s_{i+1}$ for all $i \geq 0$, we have that

$$|\bigcup_{i \geq 0} \mathrm{ADOM}(db(s_i))| \leq b.$$

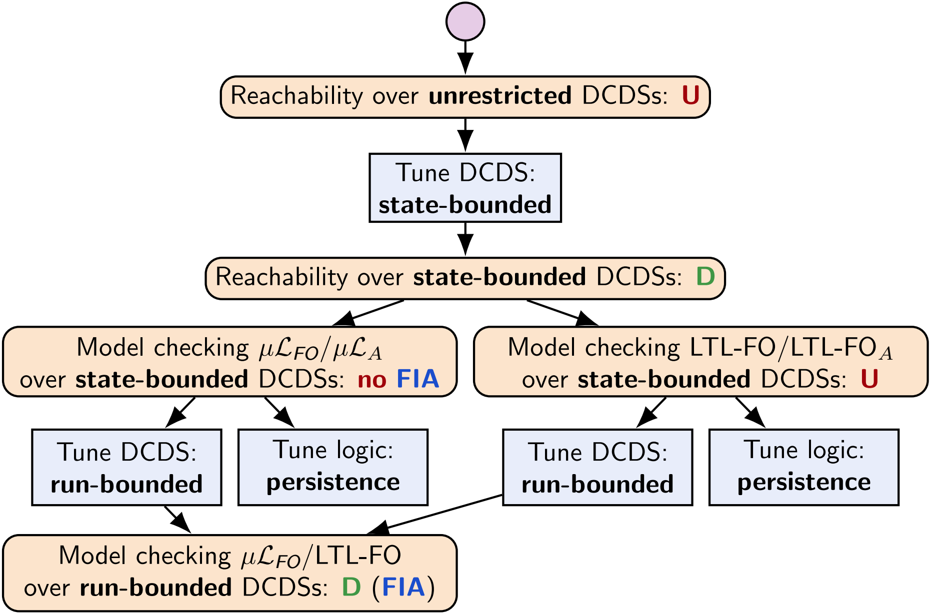
Note:

- In general, even when $\mathcal{X}$ is run-bounded, $\Upsilon_{\mathcal{X}}$ is still infinite-state due to infinite branching (but we have seen how to cope with this).
- Run-boundedness is a semantic condition.

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 TS.
- Verification of LTL-FO$_A$ over run-bounded DCDSs is decidable and can be reduced to model checking of propositional LTL over a finite TS.

Current overall picture



Avoiding run-boundedness

Run-boundedness is a **rather restrictive condition** for DCDSs

- With non-deterministic services: only a finite number of service calls ...
- With deterministic services: only a finite number of distinct service calls ...

... may be issued along a run.

Instead of requiring run-boundedness, we:

- restrict the form of quantification, and
- show how to construct a finite faithful abstraction in which we reuse values along runs.

Eventually recycling pruning

Intuition:

- We consider logics with persistence-preserving quantification, which cannot quantify over values, once they have left the active domain.
- When we need to return new values from service calls, we "recycle" those values that previously disappeared.
- We incorporate the recycling into the construction of the RTS for the DCDS, effectively pruning the set of generated states.
- If the DCDS is b-bounded, the recycling algorithm will introduce at most $2 \cdot b$ new values overall. Namely, for each state s:
  - at most b values that constitute $\textsc{adom}(db(s))$;
  - at most b new values that are introduced by the service calls, and that possibly replace some of the values in $\textsc{adom}(db(s))$.

Recycling algorithm

Algorithm RECYCLE
Input: DCDS $\mathcal{X} = \langle \mathcal{D}, \mathcal{P}, \mathcal{I}_0 \rangle$, with $\mathcal{D} = \langle \mathcal{R}, \mathcal{C} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \varrho \rangle$.

$S := \{\mathcal{I}_0\}; \quad \Rightarrow := \emptyset; \quad \mathit{UsedValues} := \textsc{adom}(\mathcal{I}_0);$

repeat

pick non visited triple of state $\mathcal{I} \in S$, action α, and legal parameters σ;

$\mathit{RecyclableValues} := \mathit{UsedValues} - (\textsc{adom}(\mathcal{I}_0) \cup \textsc{adom}(\mathcal{I}));$

pick set $\mathcal{V}$ of n service call results such that:

1. $|\mathcal{V}| = n = |\textsc{calls}(\textsc{add}(\mathcal{I}, \alpha, \sigma) \cup \textsc{del}(\mathcal{I}, \alpha, \sigma))|$, and
2. $\begin{cases} \mathcal{V} \subseteq \mathit{RecyclableValues}, & \text{if } |\mathit{RecyclableValues}| \geq n, \quad \text{\% recycled values} \\ \mathcal{V} \subset \Delta - \mathit{UsedValues}, & \text{otherwise;} \quad \text{\% fresh values} \end{cases}$

$F := \textsc{adom}(\mathcal{I}_0) \cup \textsc{adom}(\mathcal{I}) \cup \mathcal{V};$

for each $\theta \in \textsc{evals}_F(\mathcal{I}, \alpha, \sigma)$ such that $\mathcal{I}_{next} \models \mathcal{C}$,

where $\mathcal{I}_{next} := \textsc{do}(\mathcal{I}, \alpha, \sigma, \theta)$ **do**

$S := S \cup \{\mathcal{I}_{next}\};$

$\Rightarrow := \Rightarrow \cup \{\langle \mathcal{I}, \mathcal{I}_{next} \rangle\};$

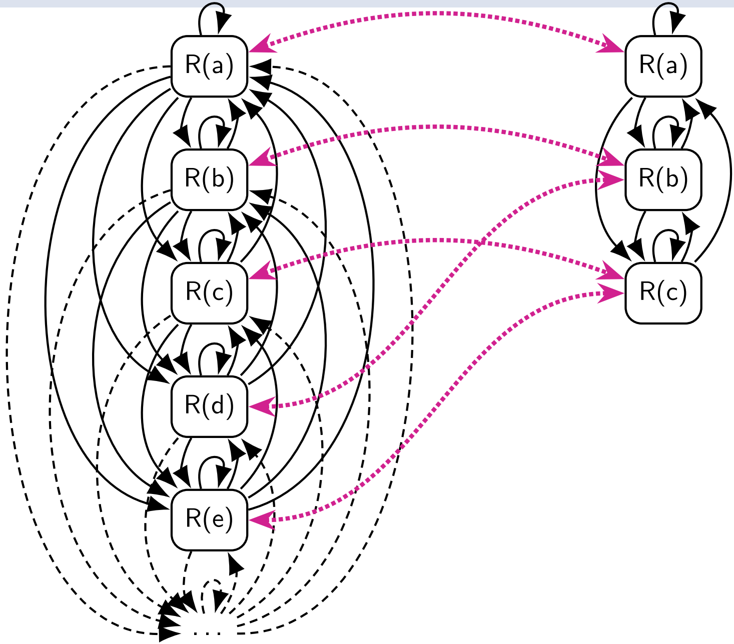
$\mathit{UsedValues} := \mathit{UsedValues} \cup \textsc{adom}(\mathcal{I}_{next});$

enddo

until S and $\Rightarrow$ no longer change;

return $\langle \Delta, \mathcal{R}, S, \mathcal{I}_0, db_{id}, \Rightarrow \rangle$, where db_{id} is the identity function.

Example



Outline

Decidability for persistence-preserving logics

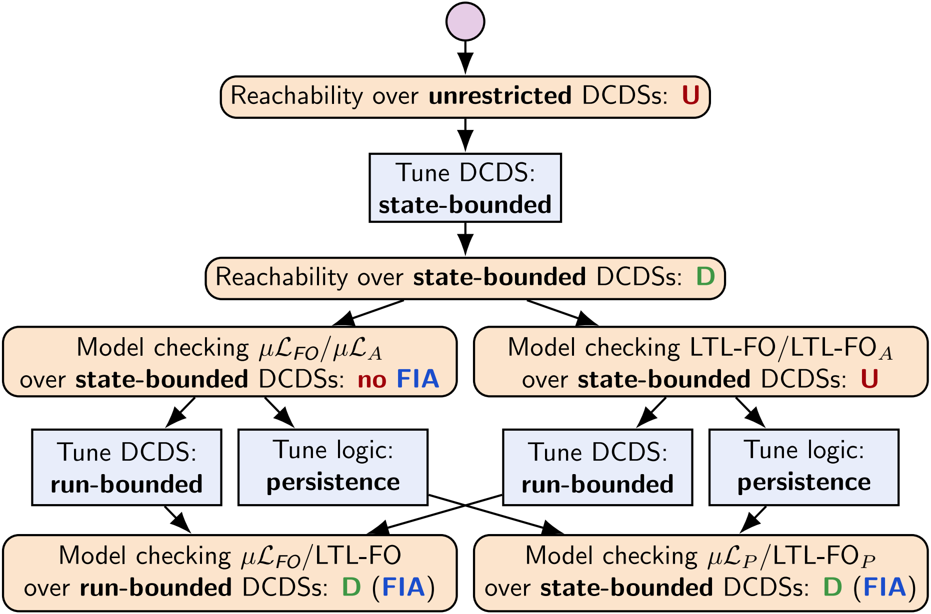
- Given as input a state-bounded DCDS $\mathcal{X}$, algorithm RECYCLE constructs a **finite** RTS $\Theta_{\mathcal{X}}$.
- Moreover, $\Theta_{\mathcal{X}}$ and $\Upsilon_{\mathcal{X}}$ are persistence-preserving bisimilar.
- *Note:* the algorithm does not require to know the bound b for the state.

From this, and the fact that $\mu\mathcal{L}_P$ / LTL-FO$_A$ are invariant under persistence-reserving bisimulations, we obtain decidability of verification.

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 TS.
- Verification of LTL-FO$_P$ over state-bounded DCDSs is decidable and can be reduced to model checking of propositional LTL over a finite TS.

Current overall picture



$\mu\mathcal{L}_A$ and $\mu\mathcal{L}_{FO}$ over state-bounded DCDSs

We have seen that $\mu\mathcal{L}_A$ (and hence $\mu\mathcal{L}_{FO}$) over state-bounded DCDSs does not admit formula-independent abstractions.

But is verification decidable?

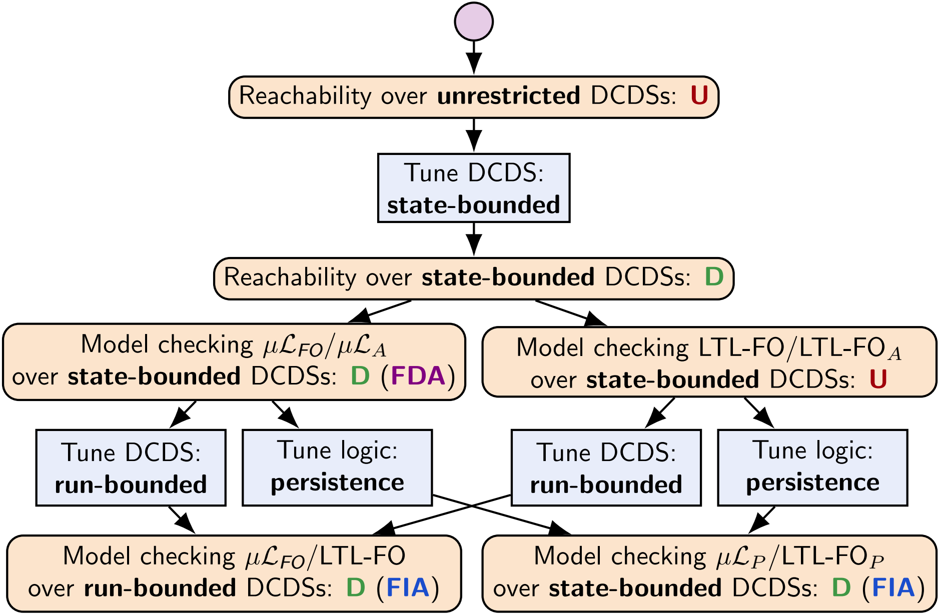
- $\mu\mathcal{L}_{FO}$ is not able to single out properties about a run.
- Combined with genericity of the RTS generated by a DCDS $\mathcal{X}$, this limits the ability to express first-order temporal properties over $\Upsilon_{\mathcal{X}}$.
- Hence, given a $\mu\mathcal{L}_{FO}$ formula Φ with n variables, we can introduce n data slots that keep track of their assignments.

Theorem

Given a state-bounded DCDS $\mathcal{X}$ and an integer n, we can construct a finite state abstraction $\Theta_{\mathcal{X}}$ of $\Upsilon_{\mathcal{X}}$ (that depends on n) such that, for every $\mu\mathcal{L}_{FO}$ formula Φ with n variables,

$$\Theta_{\mathcal{X}} \models \Phi \quad \text{if and only if} \quad \Upsilon_{\mathcal{X}} \models \Phi.$$

Final overall picture



Deciding state and run-boundedness

State-boundedness and run-boundedness are **semantic properties**.

Theorem

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

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

Proof of undecidability of checking boundedness

By encoding the halting problem of TMs. Given a TM M:

- We construct a DCDS $\mathcal{X}_M$ that encodes the computation of M.
- $\mathcal{X}_M$ also maintains an additional unary relation R, in which it inserts a fresh value for each transition that M performs.

We have that:

The TM M halts iff $\mathcal{X}_M$ is state-bounded iff $\mathcal{X}_M$ is run-bounded.

Deciding state and run-boundedness

State-boundedness and run-boundedness are **semantic properties**.

Theorem

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

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

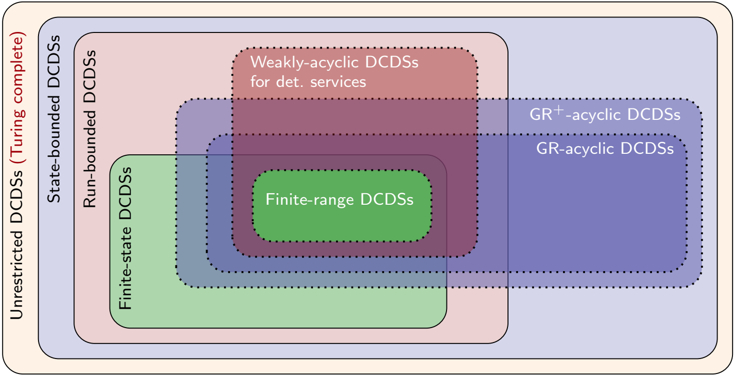
Proof of decidability of checking b-boundedness of a DCDS $\mathcal{X}$

We construct a new DCDS $\mathcal{X}'$ as follows:

- Define a Boolean query $Q_{>b}$ testing that the active domain contains more than b distinct values.
- Conjoin each condition in the condition-action rules with $\neg Q_{>b}$, thus blocking all actions when the size of the active domain exceeds b.
- Add a new condition-action rule that raises a flag when $Q_{>b}$ becomes true. Hence, the flag is raised in $\mathcal{X}'$ if and only if $\mathcal{X}$ is not b-bounded.

$\mathcal{X}'$ is state-bounded, hence reachability of raising the flag is decidable.
(For decidability of checking b-run-boundedness, we can proceed analogously.)

Results on (un)decidability of verification for DCDSs



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

D: decidable with formula independent abstraction **U**: undecidable

FDA: decidable, but formula dependent abstraction