State-Boundedness in Data-Aware Dynamic Systems

Marco Montali

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

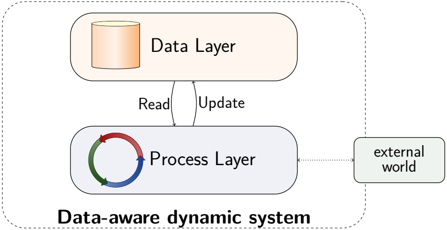
Joint work with: B. Bagheri Hariri, D. Calvanese, A. Deutsch

KR 2014

Data-Aware Dynamic System

unibz

A dynamic system that manipulate data over time.



Recall the VSL keynote by F. Baader.

- 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)

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 get 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: $\mathbf{t}(p) : \left\{ \begin{array}{lll} R(x, y) \wedge x \neq p & \rightsquigarrow & R(x, y) \\ R(p, y) & \rightsquigarrow & R(p, f(y)) \\ R(p, y) & \rightsquigarrow & Q(p) \end{array} \right\}$

Data-Centric Dynamic Systems (DCDSs)

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 get 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)

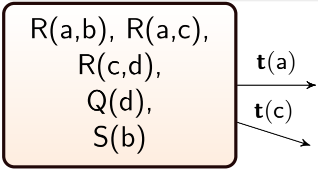
Data-Centric Dynamic Systems (DCDSs)

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 get 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)

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 get 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: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \rightsquigarrow & R(x,y) \\ R(\mathsf{a},y) & \rightsquigarrow & R(\mathsf{a}, f(y)) \\ R(\mathsf{a},y) & \rightsquigarrow & Q(\mathsf{a}) \end{array} \right\}$

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

$\xrightarrow{\mathbf{t}(\mathsf{a})}$

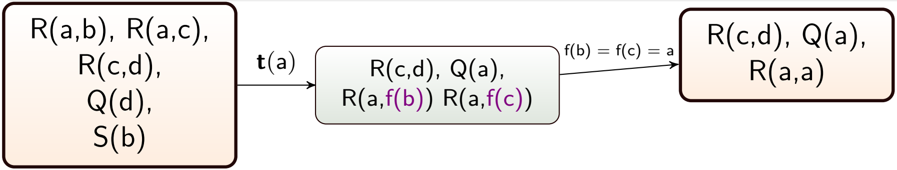
R(c,d), Q(a),
R(a,f(b)) R(a,f(c))

Data-Centric Dynamic Systems (DCDSs)

- 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 get 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: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \rightsquigarrow & R(x,y) \\ R(\mathsf{a},y) & \rightsquigarrow & R(\mathsf{a},f(y)) \\ R(\mathsf{a},y) & \rightsquigarrow & Q(\mathsf{a}) \end{array} \right\}$

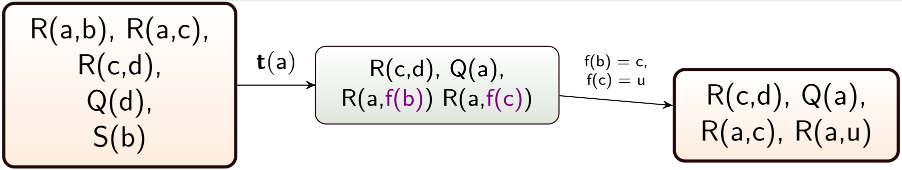


Data-Centric Dynamic Systems (DCDSs)

- 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 get 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: $\mathbf{t}(\mathsf{a}) : \left\{ \begin{array}{lcl} R(x,y) \wedge x \neq \mathsf{a} & \leadsto & R(x,y) \\ R(\mathsf{a},y) & \leadsto & R(\mathsf{a}, f(y)) \\ R(\mathsf{a},y) & \leadsto & Q(\mathsf{a}) \end{array} \right\}$



Data-Centric Dynamic Systems (DCDSs)

- 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 get new data.

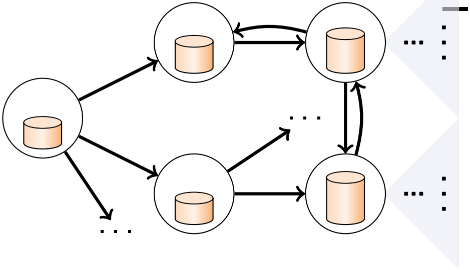
Example

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

$$\boxed{\begin{array}{c}\text{R(a,b), R(a,c),}\\ \text{R(c,d),}\\ \text{Q(d),}\\ \text{S(b)}\end{array}} \xrightarrow{\mathbf{t}(\mathsf{a})} \boxed{\begin{array}{c}\text{R(c,d), Q(a),}\\ \text{R(a,f(b)) R(a,f(c))}\end{array}} \xrightarrow{\dots} \dots$$

Verification

Execution semantics: a relational transition system, possibly with infinitely many states.

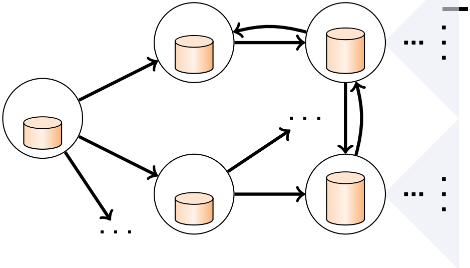


Verification Problem

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

Verification

Execution semantics: a relational transition system, possibly with infinitely many states.



Verification Problem

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

Undecidable in general!
(Even for propositional temporal logics)

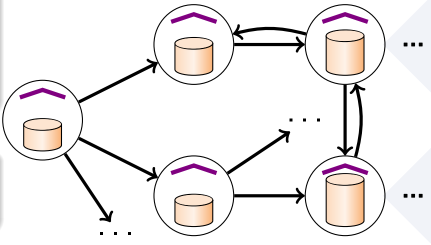
State-boundedness

State-Bounded System

Has an overall bound on the number of individuals stored **in each single state** of the system.

Structurally State-Bounded

State-Bounded for any given initial database.



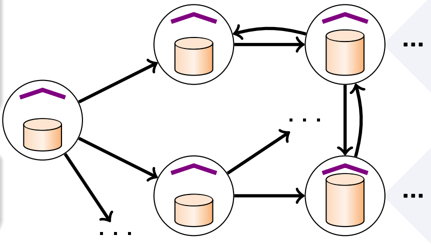
State-Boundedness

State-Bounded System

Has an overall bound on the number of individuals stored **in each single state** of the system.

Structurally State-Bounded

State-Bounded for any given initial database.



Decidability under state-boundedness

Shown in a plethora of recent works, for a variety of dynamic systems:

- Artifact-centric MASs, and FO-CTLK [BelardinelliEtAl-KR12].
- DCDSs and $\mu\mathcal{L}_p$.
- DL-based Dynamic Systems, and $\mu\mathcal{L}_p^{ECQ}$ [CalvaneseEtAl-RR13].
- Data-aware MASs with commitments, and $\mu\mathcal{L}_p^{ECQ}$ [MontaliEtAl-AAMAS14].
- Bounded situation calculus action theories, and $\mu\mathcal{L}$ [DeGiacomoEtAl-KR12].

Checking State-Boundedness

unibz

State-boundedness is a semantic property

Typically, **assumed to hold**.

Theorem ([BagheriHaririEtAl-PODS13])

Checking whether a DCDS is state-bounded is **undecidable**.

$\rightsquigarrow$ study of sufficient, syntactic conditions guaranteeing state-boundedness.

Checking State-Boundedness

unibz

State-boundedness is a semantic property

Typically, **assumed to hold**.

Theorem ([BagheriHaririEtAl-PODS13])

Checking whether a DCDS is state-bounded is **undecidable**.

$\rightsquigarrow$ study of sufficient, syntactic conditions guaranteeing state-boundedness.

Central question

Do there exist significant classes of data-aware dynamic systems for which checking state-boundedness is decidable?

Checking State-Boundedness

unibz

State-boundedness is a semantic property

Typically, **assumed to hold**.

Theorem ([BagheriHaririEtAl-PODS13])

Checking whether a DCDS is state-bounded is **undecidable**.

$\rightsquigarrow$ study of sufficient, syntactic conditions guaranteeing state-boundedness.

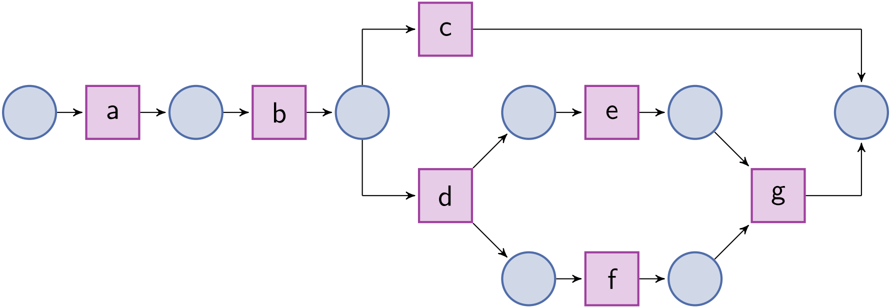
Central question

Do there exist significant classes of data-aware dynamic systems for which checking state-boundedness is decidable?

A similar question has been extensively studied in a different setting . . .

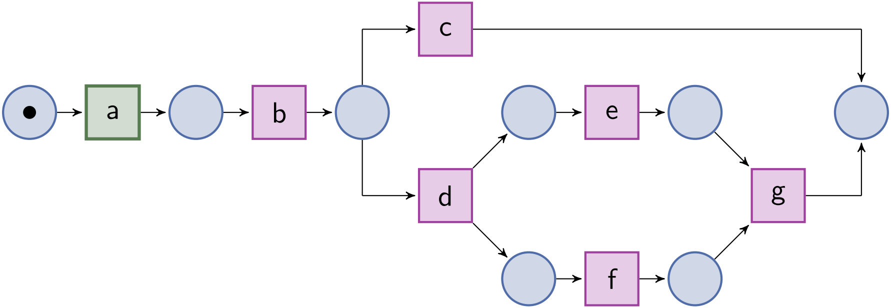
P/T nets

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



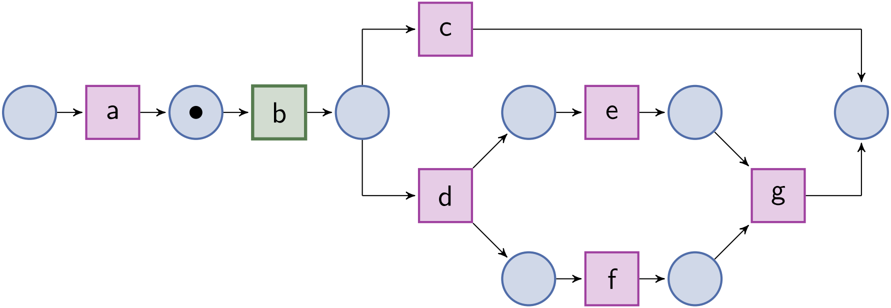
P/T nets - The Good

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



P/T nets - The Good

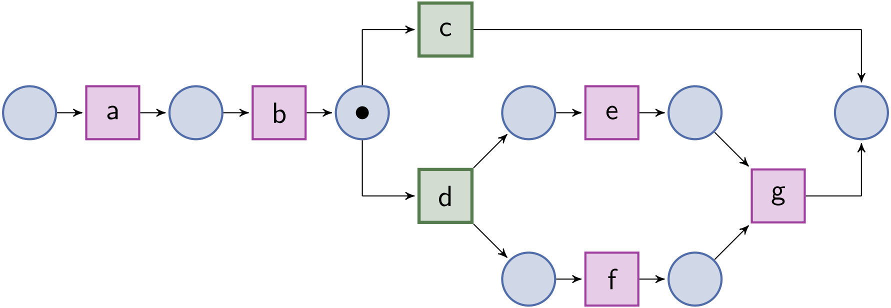
- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



P/T nets - The Good

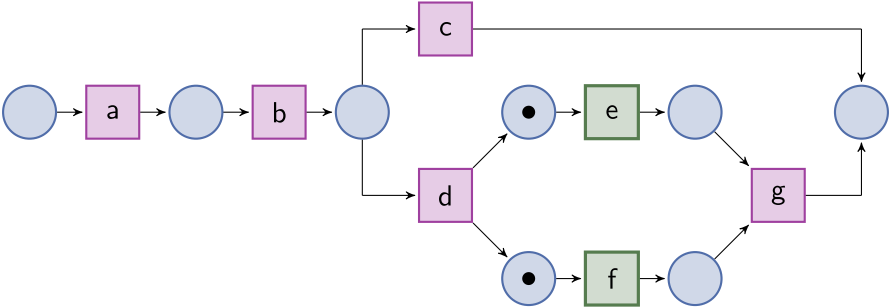
unibz

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



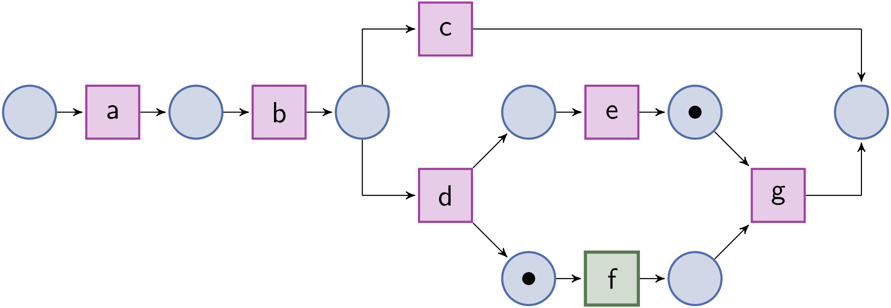
P/T nets - The Good

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, . . .
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



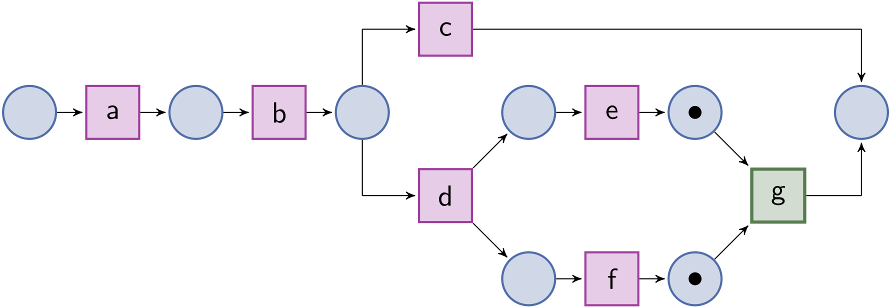
P/T nets - The Good

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



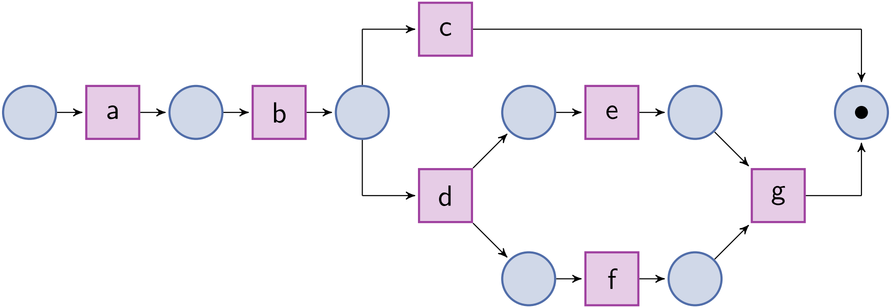
P/T nets - The Good

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



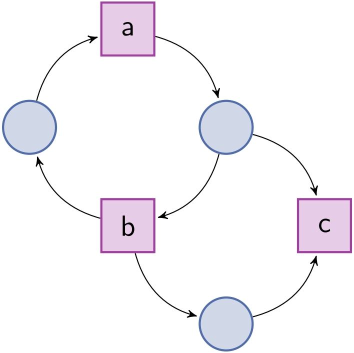
P/T nets - The Good

- Introduced by Carl Adam Petri in his PhD thesis (1962).
- Extensively used for modelling concurrent systems:
  - Distributed systems, workflows, business processes, ...
- Study of several formal properties: reachability, deadlock freedom, **boundedness**.



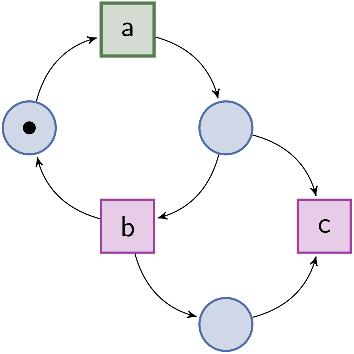
P/T Nets - The Bad

Not all marked Petri nets are bounded.



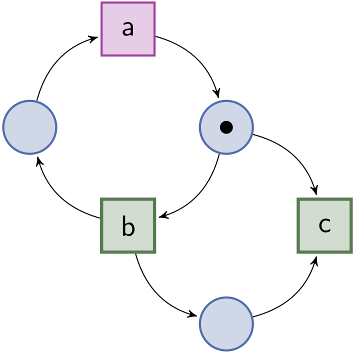
P/T Nets - The Bad

Not all marked Petri nets are bounded.



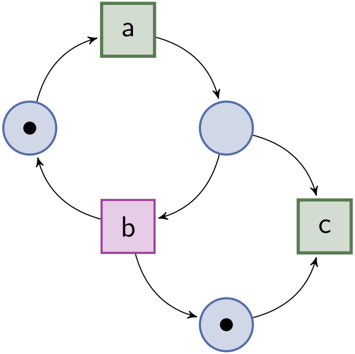
P/T Nets - The Bad

Not all marked Petri nets are bounded.



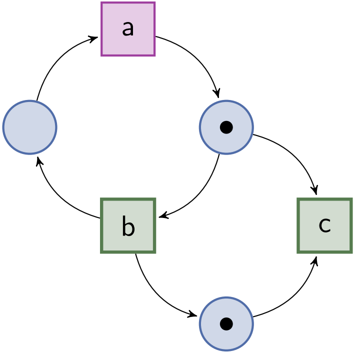
P/T Nets - The Bad

Not all marked Petri nets are bounded.



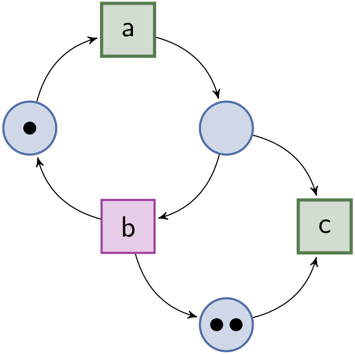
P/T Nets - The Bad

Not all marked Petri nets are bounded.



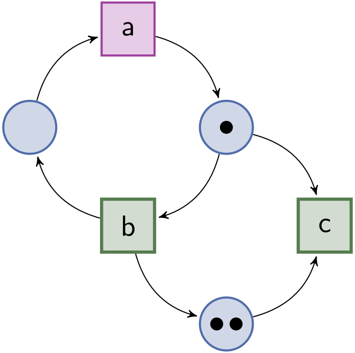
P/T Nets - The Bad

Not all marked Petri nets are bounded.



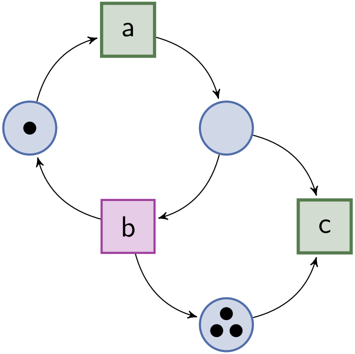
P/T Nets - The Bad

Not all marked Petri nets are bounded.



P/T Nets - The Bad

Not all marked Petri nets are bounded.



Forms of Boundedness

Boundedness

A **marked Petri net** is bounded if all executions starting from the given marking do not produce an unbounded amount of tokens.

Structural boundedness

A **Petri net** is structurally bounded if for all possible initial markings the resulting marked net is bounded.

Forms of Boundedness

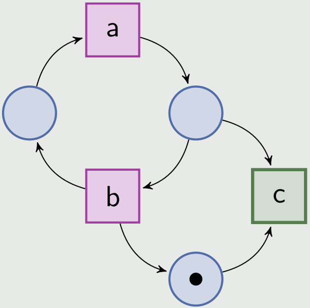
Boundedness

A **marked Petri net** is bounded if all executions starting from the given marking do not produce an unbounded amount of tokens.

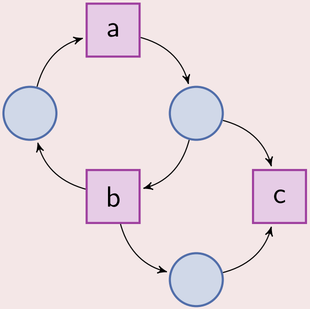
Structural boundedness

A **Petri net** is structurally bounded if for all possible initial markings the resulting marked net is bounded.

Bounded

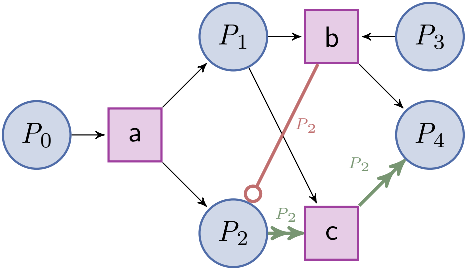


Structurally unbounded

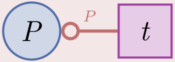


Reset Transfer Nets - The Ugly

$\boxed{\text{P/T nets}} \subset \boxed{\textbf{RT nets}} \subset \boxed{\text{Reset Post G-nets [DufourEtAlICALP98]}}$

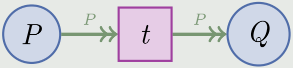


Reset arc



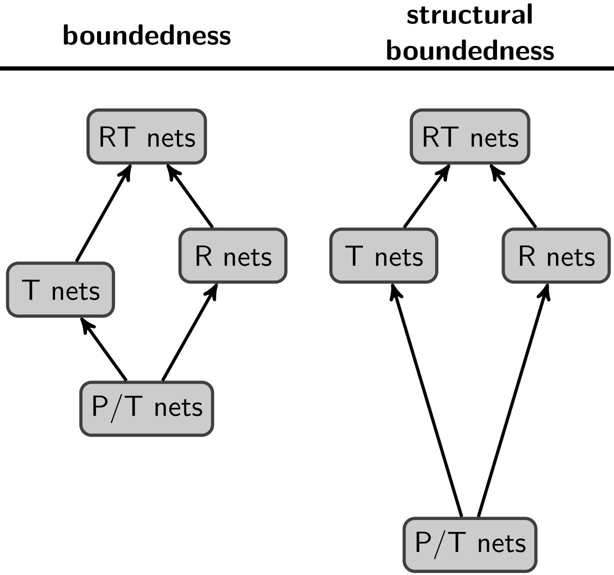
When t fires, all tokens in P are removed.

Transfer pair

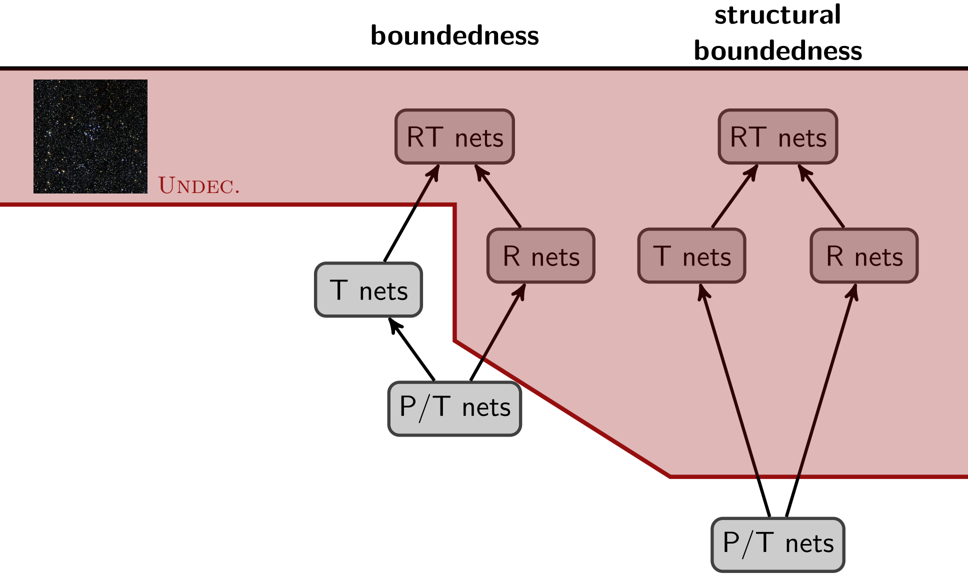


When t fires, all tokens in P are transferred to Q.

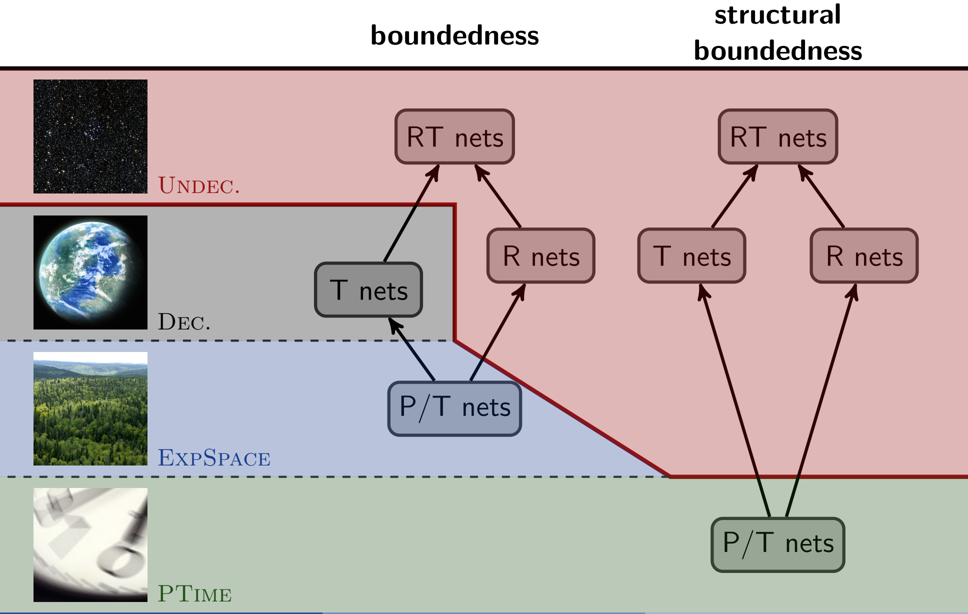
Boundedness Spectrum



Boundedness Spectrum



Boundedness Spectrum

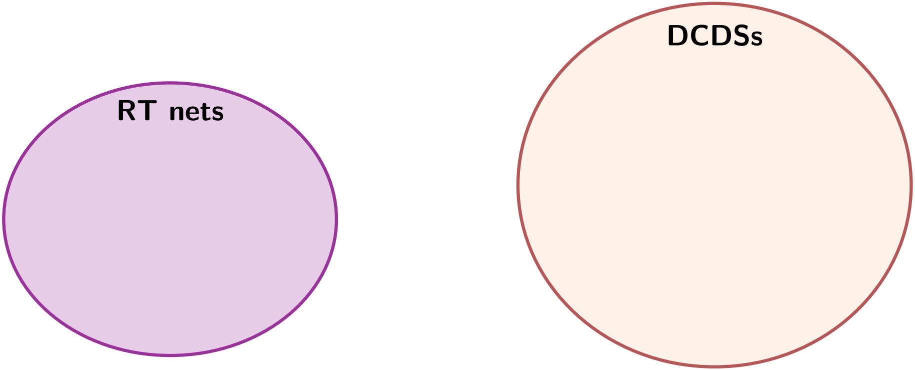


unibz

Goal

Devise a connection between RT nets and DCDSs so as to understand the state-boundedness spectrum in data-aware dynamic systems.

Main issue: set vs bag semantics.

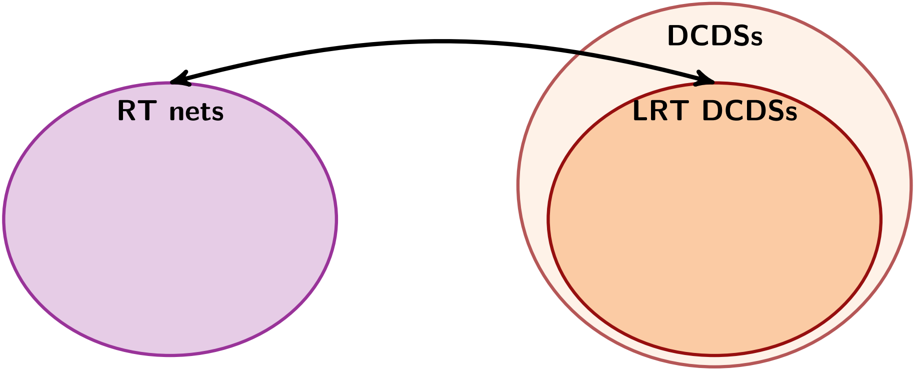


Understanding State-Boundedness

Goal

Devise a connection between RT nets and DCDSs so as to understand the state-boundedness spectrum in data-aware dynamic systems.

Main issue: set vs bag semantics.

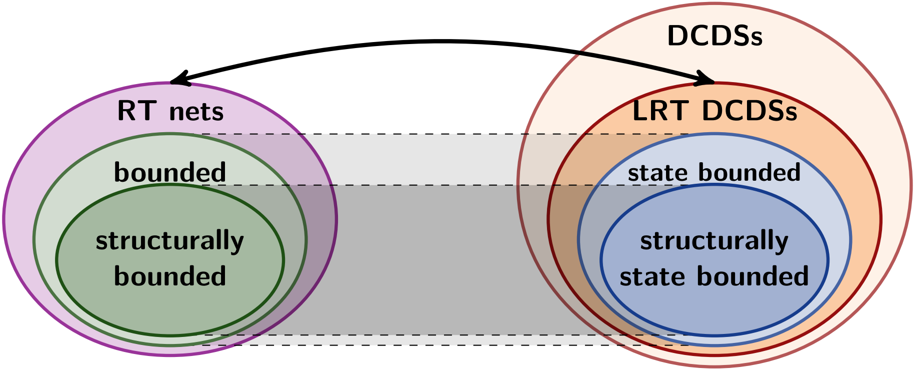


Understanding State-Boundedness

Goal

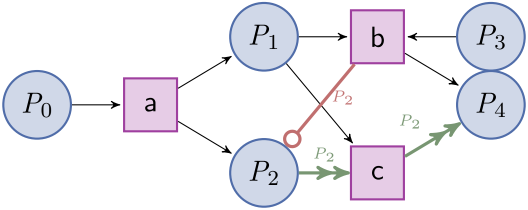
Devise a connection between RT nets and DCDSs so as to understand the state-boundedness spectrum in data-aware dynamic systems.

Main issue: set vs bag semantics.



From RT Nets to DCDSs

unibz



Idea

Tokens as distinct identifiers distributed over place relations.
Only cardinalities matter, not the data values.

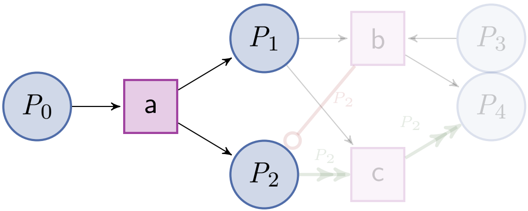
Data layer.

- Unary relations for places: $\{P_i/1 \mid i \in \{1, \ldots, 4\}\}$
- No constraints.

Process layer: each transition becomes an action + condition-action rule.

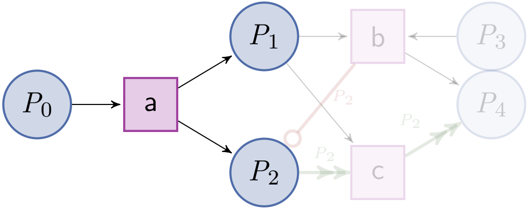
- Condition: gets tokens from input places; feeds the action with them.
- Action: moves tokens according to the firing semantics of the net.
  - Service calls to generate identifiers for new tokens.

From RT Nets to DCDSs - The P/T Case



$$P_0(x_1) \mapsto \mathbf{a}(x_1)$$

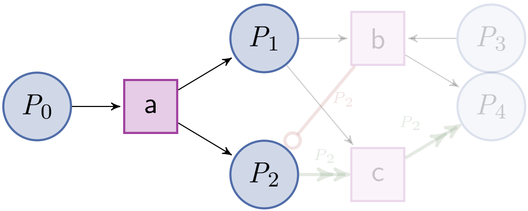
From RT Nets to DCDSs - The P/T Case



$$P_0(x_1) \mapsto \mathbf{a}(x_1)$$

$$\mathbf{a}(x_1) : \left\{ \begin{array}{ll} \mathsf{P}_0 & P_0(y) \wedge y \neq x_1 \rightsquigarrow P_0(f(y)) \\ \mathsf{P}_1 & \\ & \\ \mathsf{P}_2 & \\ & \\ \mathsf{P}_3 & \\ \mathsf{P}_4 & \end{array} \right\}$$

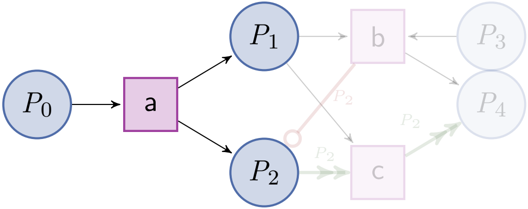
From RT Nets to DCDSs - The P/T Case



$$P_0(x_1) \mapsto \mathbf{a}(x_1)$$

$$\mathbf{a}(x_1) : \left\{ \begin{array}{llcl} \mathsf{P}_0 & P_0(y) \wedge y \neq x_1 & \rightsquigarrow & P_0(f(y)) \\ \mathsf{P}_1 & P_1(y) & \rightsquigarrow & P_1(h_1(y)) \\ & \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathsf{P}_2 & P_2(y) & \rightsquigarrow & P_2(h_2(y)) \\ & \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ \mathsf{P}_3 & & & \\ \mathsf{P}_4 & & & \end{array} \right\}$$

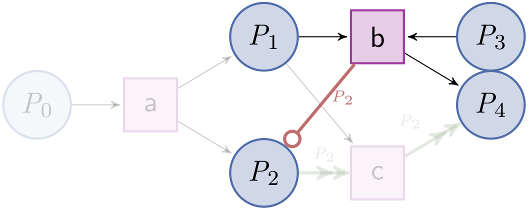
From RT Nets to DCDSs - The P/T Case



$$P_0(x_1) \mapsto \mathbf{a}(x_1)$$

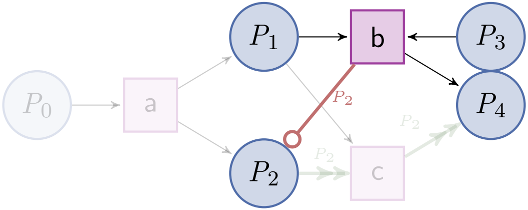
$$\mathbf{a}(x_1) : \left\{ \begin{array}{llcl} \mathsf{P}_0 & P_0(y) \wedge y \neq x_1 & \rightsquigarrow & P_0(f(y)) \\ \mathsf{P}_1 & P_1(y) & \rightsquigarrow & P_1(h_1(y)) \\ & \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathsf{P}_2 & P_2(y) & \rightsquigarrow & P_2(h_2(y)) \\ & \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ \mathsf{P}_3 & P_3(y) & \rightsquigarrow & P_3(h_3(y)) \\ \mathsf{P}_4 & P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$$

From RT Nets to DCDSs - The Reset Case



$$P_1(x_1) \wedge P_3(x_2) \mapsto \mathbf{b}(x_1, x_2)$$

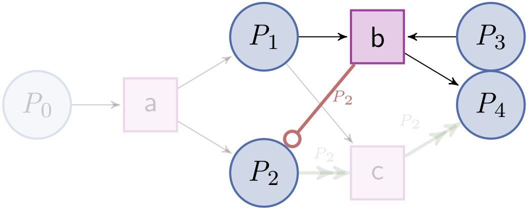
From RT Nets to DCDSs - The Reset Case



$$P_1(x_1) \wedge P_3(x_2) \mapsto \mathbf{b}(x_1, x_2)$$

$$\mathbf{b}(x_1, x_2) : \left\{ \begin{array}{lll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow \; P_0(h_0(y)) \\ \mathsf{P}_1 & & \\ \mathsf{P}_2 & & \\ \mathsf{P}_3 & & \\ \mathsf{P}_4 & & \\ & & \end{array} \right\}$$

From RT Nets to DCDSs - The Reset Case

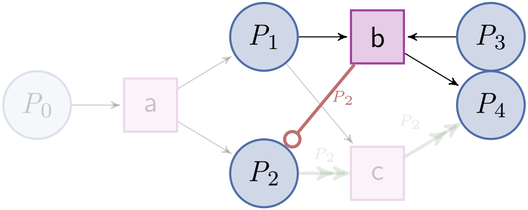


$$P_1(x_1) \wedge P_3(x_2) \mapsto \mathbf{b}(x_1, x_2)$$

$$\mathbf{b}(x_1, x_2) : \left\{ \begin{array}{llll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow & P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(h_1(y)) \\ \mathsf{P}_2 & & & \\ \mathsf{P}_3 & P_3(y) \wedge y \neq x_2 & \rightsquigarrow & P_3(h_3(y)) \\ \mathsf{P}_4 & & & \\ & & & \end{array} \right\}$$

From RT Nets to DCDSs - The Reset Case

unibz

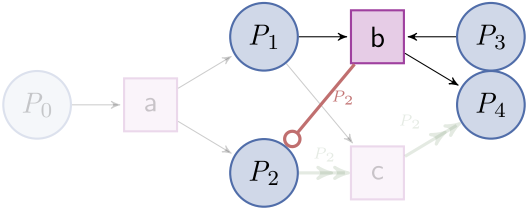


$$P_1(x_1) \wedge P_3(x_2) \mapsto \mathbf{b}(x_1, x_2)$$

$$\mathbf{b}(x_1, x_2) : \left\{ \begin{array}{lll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow P_1(h_1(y)) \\ \mathsf{P}_2 & - & \\ \mathsf{P}_3 & P_3(y) \wedge y \neq x_2 & \rightsquigarrow P_3(h_3(y)) \\ \mathsf{P}_4 & & \\ & & \end{array} \right\}$$

From RT Nets to DCDSs - The Reset Case

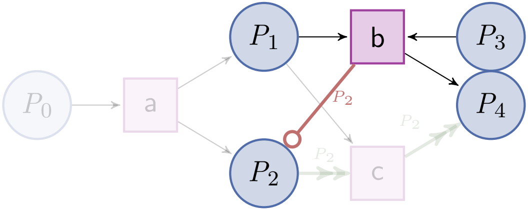
unibz



$$P_1(x_1) \wedge P_3(x_2) \mapsto \mathbf{b}(x_1, x_2)$$

$$\mathbf{b}(x_1, x_2) : \left\{ \begin{array}{lll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow P_1(h_1(y)) \\ \mathsf{P}_2 & - & \\ \mathsf{P}_3 & P_3(y) \wedge y \neq x_2 & \rightsquigarrow P_3(h_3(y)) \\ \mathsf{P}_4 & P_4(y) & \rightsquigarrow P_4(h_4(y)) \\ & \mathit{true} & \rightsquigarrow P_4(g_4()) \end{array} \right\}$$

From RT Nets to DCDSs - The Reset Case

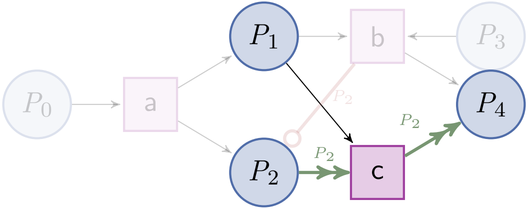


$$P_1(x_1) \wedge P_3(x_2) \mapsto \mathbf{b}(x_1, x_2)$$

$$\mathbf{b}(x_1, x_2) : \left\{ \begin{array}{lllll} \mathrm{P}_0 & P_0(y) & & \rightsquigarrow & P_0(h_0(y)) \\ \mathrm{P}_1 & P_1(y) \wedge y \neq x_1 & & \rightsquigarrow & P_1(h_1(y)) \\ \mathrm{P}_2 & - & & & \\ \mathrm{P}_3 & P_3(y) \wedge y \neq x_2 & & \rightsquigarrow & P_3(h_3(y)) \\ \mathrm{P}_4 & P_4(y) & & \rightsquigarrow & P_4(h_4(y)) \\ & \mathit{true} & & \rightsquigarrow & P_4(g_4()) \end{array} \right\}$$

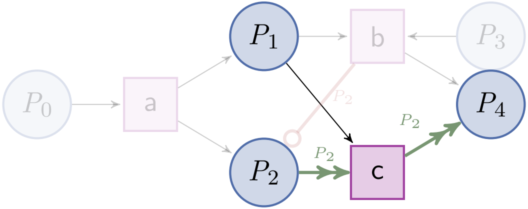
A relation (P_2) does not appear in the action!

From RT Nets to DCDSs - The Transfer Case



$$P_1(x_1) \mapsto \mathbf{c}(x_1)$$

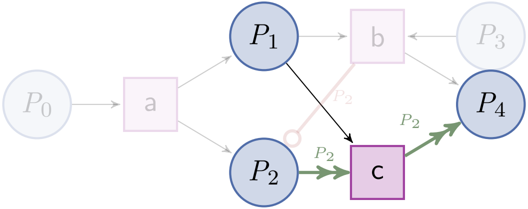
From RT Nets to DCDSs - The Transfer Case



$$P_1(x_1) \mapsto \mathbf{c}(x_1)$$

$$\mathbf{c}(x_1) : \left\{ \begin{array}{lll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow \quad P_0(h_0(y)) \\ \mathsf{P}_1 & & \\ \mathsf{P}_2 & & \\ \mathsf{P}_3 & P_3(y) & \rightsquigarrow \quad P_3(h_3(y)) \\ \mathsf{P}_4 & & \end{array} \right\}$$

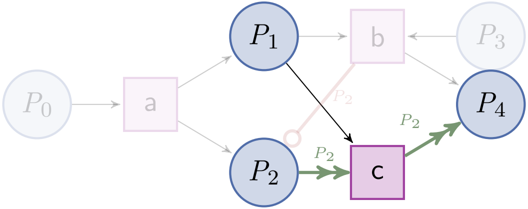
From RT Nets to DCDSs - The Transfer Case



$$P_1(x_1) \mapsto \mathbf{c}(x_1)$$

$$\mathbf{c}(x_1) : \left\{ \begin{array}{lllll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow & P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(h_1(y)) \\ \mathsf{P}_2 & & & \\ \mathsf{P}_3 & P_3(y) & \rightsquigarrow & P_3(h_3(y)) \\ \mathsf{P}_4 & & & \end{array} \right\}$$

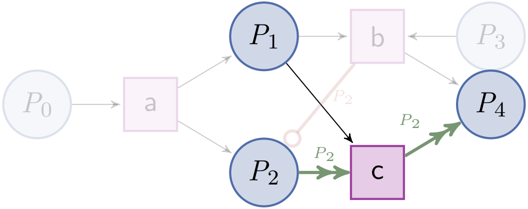
From RT Nets to DCDSs - The Transfer Case



$$P_1(x_1) \mapsto \mathbf{c}(x_1)$$

$$\mathbf{c}(x_1) : \left\{ \begin{array}{lllll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow & P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(h_1(y)) \\ \mathsf{P}_2 & P_2(y) & \rightsquigarrow & P_4(h_2(y)) \\ \mathsf{P}_3 & P_3(y) & \rightsquigarrow & P_3(h_3(y)) \\ \mathsf{P}_4 & & & \end{array} \right\}$$

From RT Nets to DCDSs - The Transfer Case

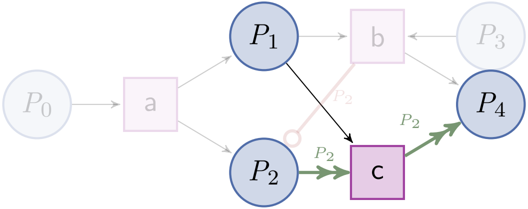


$$P_1(x_1) \mapsto \mathbf{c}(x_1)$$

$$\mathbf{c}(x_1) : \left\{ \begin{array}{lll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow & P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(h_1(y)) \\ \mathsf{P}_2 & P_2(y) & \rightsquigarrow & P_4(h_2(y)) \\ \mathsf{P}_3 & P_3(y) & \rightsquigarrow & P_3(h_3(y)) \\ \mathsf{P}_4 & P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$$

From RT Nets to DCDSs - The Transfer Case

unibz



$$P_1(x_1) \mapsto \mathbf{c}(x_1)$$

$$\mathbf{c}(x_1) : \left\{ \begin{array}{lll} \mathsf{P}_0 & P_0(y) & \rightsquigarrow P_0(h_0(y)) \\ \mathsf{P}_1 & P_1(y) \wedge y \neq x_1 & \rightsquigarrow P_1(h_1(y)) \\ \mathsf{P}_2 & P_2(y) & \rightsquigarrow P_4(h_2(y)) \\ \mathsf{P}_3 & P_3(y) & \rightsquigarrow P_3(h_3(y)) \\ \mathsf{P}_4 & P_4(y) & \rightsquigarrow P_4(h_4(y)) \end{array} \right\}$$

There is an effect involving two different relations (P_2 and P_4)!

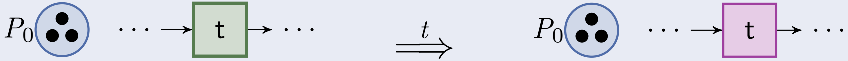
Is the Translation Correct?

NO! The resulting DCDS has a **lossy behavior**.

Is the Translation Correct?

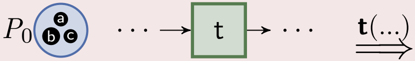
NO! The resulting DCDS has a **lossy behavior**.

Petri net

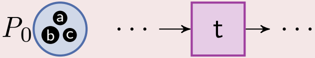


DCDS

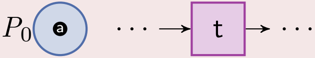
$$\mathbf{t}(\ldots) : \{\ldots, P_0(y) \rightsquigarrow P_0(h_0(y)), \ldots\}$$



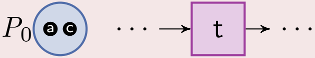
$h_0(a) = a, h_0(b) = b, h_0(c) = c$



$h_0(a) = a, h_0(b) = a, h_0(c) = a$



$h_0(a) = a, h_0(b) = a, h_0(c) = c$



...

Is the Translation Correct?

unibz

However. . .

The resulting DCDS reproduces **all behaviors** of the net (and more).

Is the Translation Correct?

unibz

However. . .

The resulting DCDS reproduces **all behaviors** of the net (and more).

Theorem

An RT net is (structurally) bounded if and only if the corresponding DCDS is (structurally) state-bounded.

LRT DCDS

unibz

Data Layer

Schema $\mathcal{R}$ with unary relations only, and no constraint.

Process

Only one rule per action, of the form $Q(\vec{x}) \mapsto \alpha(\vec{x})$, where

$$Q(x_1, \ldots, x_n) = \bigwedge_{i \in \{1,\ldots,n\},\ P_i \neq P_j \text{ for } i \neq j} P_i(x_i)$$

Shape of action $\alpha(\vec{x})$

For each $P_i \in \text{RELS}(Q)$, α **must** contain:

- $P_i(y) \wedge y \neq x_i \rightsquigarrow P_i(f_i(y))$

and **may** contain:

- true $\rightsquigarrow P_i(g_i())$

For each $P_l \in \mathcal{R} \setminus \text{RELS}(Q)$, α **may** contain:

- $P_l(y) \rightsquigarrow P_l(h_l(y))$
- *either* true $\rightsquigarrow P_j(g_j())$,
  or $P'_j(y) \rightsquigarrow P_j(h_j(y))$
  for some $P'_j \in \mathcal{R} \setminus (\text{RELS}(Q) \cup P_j)$.

From LRT DCDSs to RT Nets

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action **t** with:

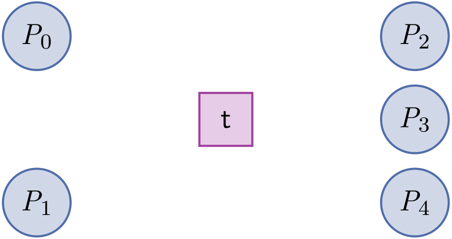
- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$

From LRT DCDSs to RT Nets

unibz

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action **t** with:

- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$

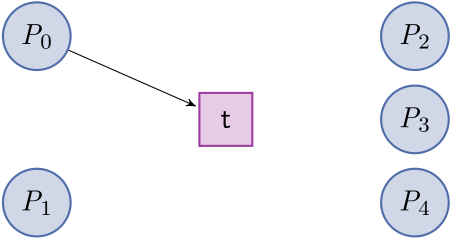


From LRT DCDSs to RT Nets

unibz

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action $\mathbf{t}$ with:

- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$

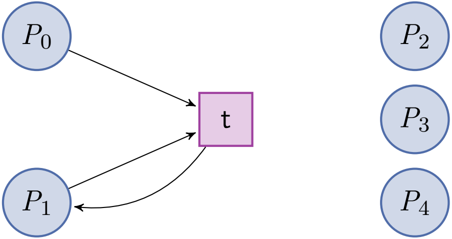


From LRT DCDSs to RT Nets

unibz

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action **t** with:

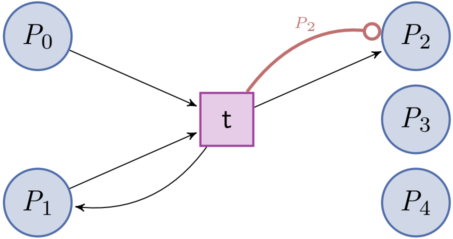
- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$



From LRT DCDSs to RT Nets

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action **t** with:

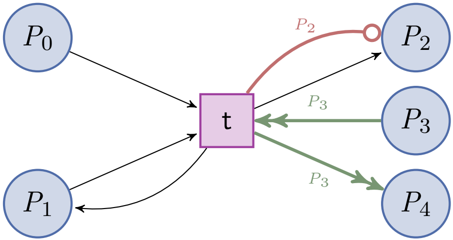
- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ true & \rightsquigarrow & P_1(g_1()) \\ true & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$



From LRT DCDSs to RT Nets

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action **t** with:

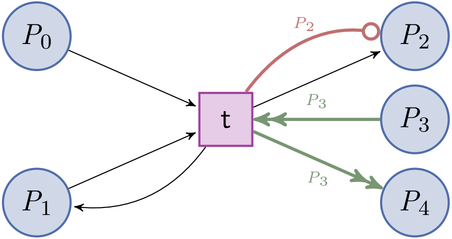
- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$



From LRT DCDSs to RT Nets

Consider schema $\mathcal{R} = \{P_0, P_1, P_2, P_3, P_4\}$, and action $\mathbf{t}$ with:

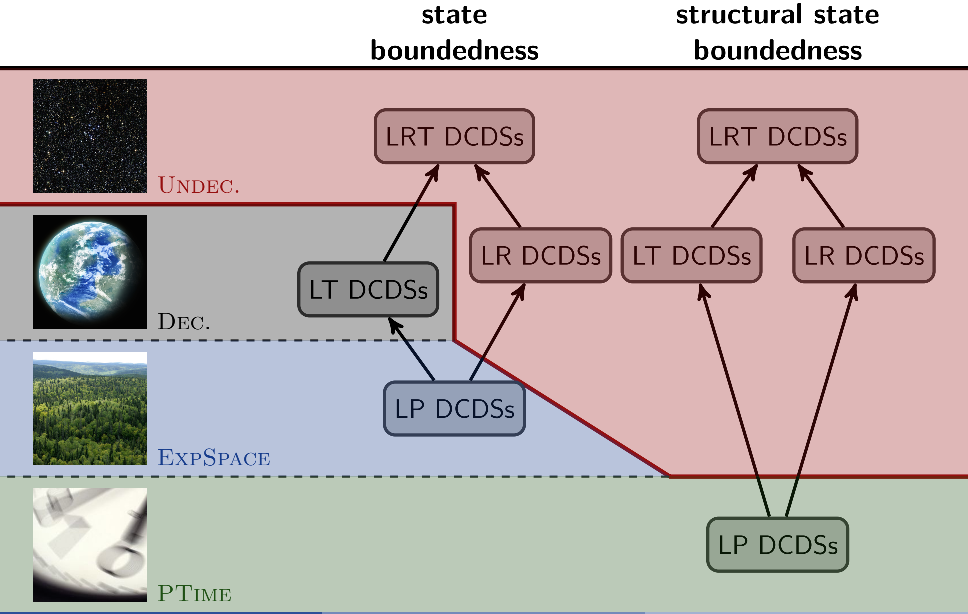
- process condition-action rule $P_0(x_0) \wedge P_1(x_1) \mapsto \mathbf{t}(x_0, x_1)$
- action $\mathbf{t}(x_0, x_1)$: $\left\{ \begin{array}{lcl} P_0(y) \wedge y \neq x_0 & \rightsquigarrow & P_0(f_0(y)) \\ P_1(y) \wedge y \neq x_1 & \rightsquigarrow & P_1(f_1(y)) \\ \mathit{true} & \rightsquigarrow & P_1(g_1()) \\ \mathit{true} & \rightsquigarrow & P_2(g_2()) \\ P_3(y) & \rightsquigarrow & P_4(h_3(y)) \\ P_4(y) & \rightsquigarrow & P_4(h_4(y)) \end{array} \right\}$



Theorem

An LRT DCDS is (structurally) state-bounded if and only if the corresponding RT net is (structurally) bounded.

State-Boundedness Spectrum



Take Home Message

LRT DCDSs are weak:

- Only unary relations.
- Only conjunctions without joins in conditions.
- Only atomic queries inside effects (possibly with a value inequality).
- Very limited use of negation (inequalities).
- No direct transfer of values from one state to the other.

Take Home Message

unibz

LRT DCDSs are weak:

- Only unary relations.
- Only conjunctions without joins in conditions.
- Only atomic queries inside effects (possibly with a value inequality).
- Very limited use of negation (inequalities).
- No direct transfer of values from one state to the other.

Still, to ensure that (structural) state-boundedness is decidable . . .

Boundedness

All relations must appear on the left-hand side of action effects, i.e., contribute to form the new state.

Structural Boundedness

Each action must be such that only a **fixed amount** of tuples is added to/removed from each relation in the schema.

Back to our Question

unibz

Central question

Do there exist significant classes of data-aware dynamic systems for which checking state-boundedness is decidable?

Answer

NO

Hence, it becomes important to provide significant sufficient, checkable syntactic conditions that guarantee structural state-boundedness.

Back to our Question

unibz

Central question

Do there exist significant classes of data-aware dynamic systems for which checking state-boundedness is decidable?

Answer

$\boxed{\textbf{NO}}$

Hence, it becomes important to provide significant sufficient, checkable syntactic conditions that guarantee structural state-boundedness.

We follow this line, focusing on DCDSs and starting from [BagheriHaririEtAl-PODS13].

GR-Acyclicity [BagheriHaririEtAl-PODS13]

Example

Consider a DCDS with process $\{\text{true} \mapsto \alpha()\}$, and

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

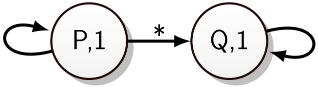
GR-Acyclicity [BagheriHaririEtAl-PODS13]

Example

Consider a DCDS with process $\{\mathsf{true} \mapsto \alpha()\}$, and

$$\alpha() : \left\{ \begin{array}{lcl} P(x) & \rightsquigarrow & P(x) \\ P(x) & \rightsquigarrow & Q(f(x)) \\ Q(x) & \rightsquigarrow & Q(x) \end{array} \right\}$$

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

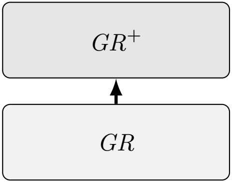


The system is not state-bounded, due to:

- 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.

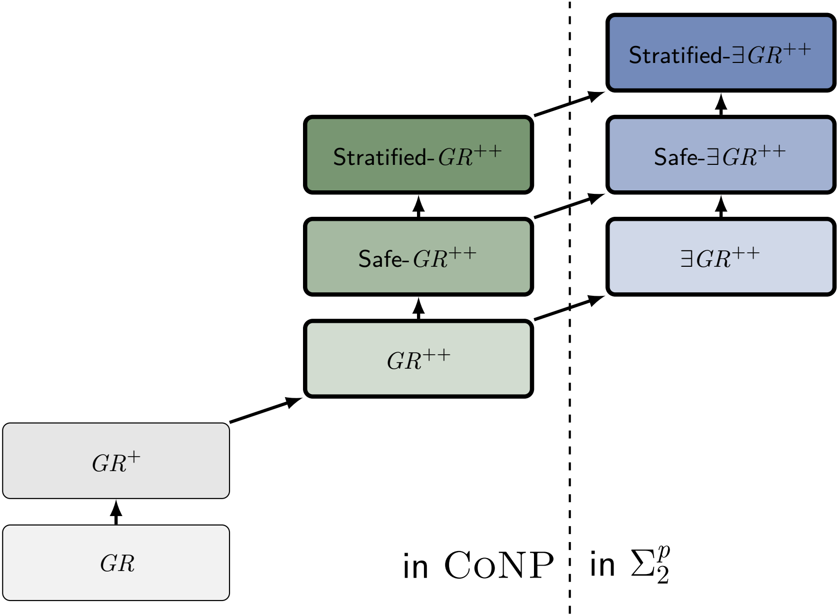
Our Contribution



in CoNP in Σ_2^p

Our Contribution

unibz



Conclusion

unibz

1.

No significant decidable classes of data-aware dynamic systems for which state-boundedness is decidable.

2.

It becomes crucial to provide checkable, sufficient conditions.

- We have built on results on chase termination for tuple-generating dependencies, providing a family of conditions for DCDSs.

Ongoing and future work

- Refine the syntactic conditions to handle `if-then-else` effects.
- Follow a different approach: provide modelling guidelines towards systems that are structurally state bounded by design.
  - Preliminary results in [SolomakhinEtAl-ICSOC13].