Model Completeness, Covers and Superposition

Diego Calvanese[1], Silvio Ghilardi[2], **Alessandro Gianola**[1], Marco Montali[1], Andrey Rivkin[1]

[1] KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

[2] Dipartimento di Matematica
Università degli Studi di Milano, Italy

June 19, 2019

Outline

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

  Advantage Software systems abstracted into **finite-state** automata.

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

  Advantage Software systems abstracted into **finite-state** automata.

  Drawback How to express manipulation of or conditions on **data**?

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

  Advantage Software systems abstracted into **finite-state** automata.

  Drawback How to express manipulation of or conditions on **data**?

- The research areas of **Data Management** and **Knowledge Representation** traditionally investigate **static** aspects of the domain of interest, disregarding dynamic aspects.

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

  Advantage Software systems abstracted into **finite-state** automata.

  Drawback How to express manipulation of or conditions on **data**?

- The research areas of **Data Management** and **Knowledge Representation** traditionally investigate **static** aspects of the domain of interest, disregarding dynamic aspects.
- Our context: **Business Processes enriched with real data**.

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

  Advantage Software systems abstracted into **finite-state** automata.

  Drawback How to express manipulation of or conditions on **data**?

- The research areas of **Data Management** and **Knowledge Representation** traditionally investigate **static** aspects of the domain of interest, disregarding dynamic aspects.
- Our context: **Business Processes enriched with real data**.
- To **bridge** the gap existing between those two approaches is *challenging*: expressing and verifying properties that *simultaneously* account for the data and the dynamic perspective.

Motivation

- Traditional **Model Checking techniques** focus on verification of temporal properties in **dynamic** finite-state systems:

  Advantage Software systems abstracted into **finite-state** automata.

  Drawback How to express manipulation of or conditions on **data**?

- The research areas of **Data Management** and **Knowledge Representation** traditionally investigate **static** aspects of the domain of interest, disregarding dynamic aspects.
- Our context: **Business Processes enriched with real data**.
- To **bridge** the gap existing between those two approaches is *challenging*: expressing and verifying properties that *simultaneously* account for the data and the dynamic perspective.
- Thanks to the presence of data, the resulting models are *intrinsically* infinite-state.

Motivation

- Infinite-state model checking requires a ***declarative*** approach: sets of (*reachable*) states and transitions are represented **symbolically**.

Motivation

- Infinite-state model checking requires a ***declarative*** approach: sets of (*reachable*) states and transitions are represented **symbolically**.
- Precise computations of the set of reachable states require some form of quantifier elimination.

Motivation

- Infinite-state model checking requires a ***declarative*** approach: sets of (*reachable*) states and transitions are represented **symbolically**.
- Precise computations of the set of reachable states require some form of quantifier elimination.
- Gulwani and Musuvathi [ESOP, 2008] introduced the notion of a ***cover***, which provides precise computation of reachable states.

Motivation

- Infinite-state model checking requires a ***declarative*** approach: sets of (*reachable*) states and transitions are represented **symbolically**.
- Precise computations of the set of reachable states require some form of quantifier elimination.
- Gulwani and Musuvathi [ESOP, 2008] introduced the notion of a ***cover***, which provides precise computation of reachable states.
- They showed that covers exist for **EUF** and proved that its computation becomes *tractable* with only unary free function symbols.

Our contributions

- We provide a new approach to verification of data-aware processes, where models are formalized using **Array-based Systems**, via *SMT-techniques*.

Our contributions

- We provide a new approach to verification of data-aware processes, where models are formalized using **Array-based Systems**, via *SMT-techniques*.
- We adapt the backward reachability procedure in order to assess **safety** properties of data-aware processes. This requires the development of Quantifier Elimination algorithms for specific theories known as **model completions**.

Our contributions

- We provide a new approach to verification of data-aware processes, where models are formalized using **Array-based Systems**, via *SMT-techniques*.
- We adapt the backward reachability procedure in order to assess **safety** properties of data-aware processes. This requires the development of Quantifier Elimination algorithms for specific theories known as **model completions**.
- We prove that ***computing covers*** for a theory is **equivalent** to ***eliminating quantifiers*** in its model completion.

Our contributions

- We provide a new approach to verification of data-aware processes, where models are formalized using **Array-based Systems**, via *SMT-techniques*.
- We adapt the backward reachability procedure in order to assess **safety** properties of data-aware processes. This requires the development of Quantifier Elimination algorithms for specific theories known as **model completions**.
- We prove that ***computing covers*** for a theory is **equivalent** to ***eliminating quantifiers*** in its model completion.
- We show that covers for **EUF** can be computed through a constrained version of the **Superposition Calculus**, equipped with appropriate settings and reduction strategies.

Outline

Artifact-Centric Systems

Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).

Artifact-Centric Systems

Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).
They **can** be formalized using three components:

Artifact-Centric Systems

Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).
They **can** be formalized using three components:

- *a read-only database (DB)*;

Artifact-Centric Systems

Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).
They **can** be formalized using three components:

- *a read-only database (DB)*;
- *an artifact working memory (e.g., artifact variables + artifact relations)*;

Artifact-Centric Systems

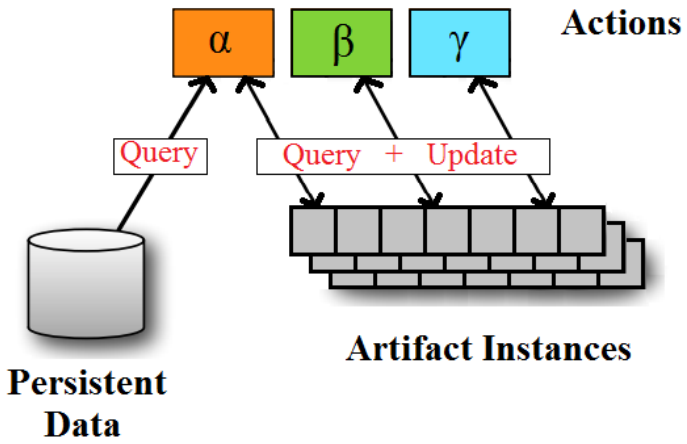
Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).
They **can** be formalized using three components:

- *a read-only database (DB)*;
- *an artifact working memory (e.g., artifact variables + artifact relations)*;
- *actions* (also called *services*).

Artifact-Centric Systems

Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).
They **can** be formalized using three components:

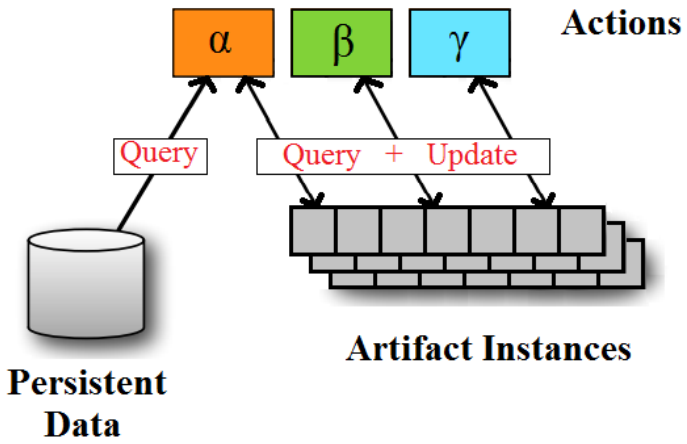
- *a read-only database (DB)*;
- *an artifact working memory (e.g., artifact variables + artifact relations)*;
- *actions* (also called *services*).



Artifact-Centric Systems

Artifact-Centric Systems enrich traditional process-centric paradigm with data (**artifact** = *information model* + *lifecycle model*).
They **can** be formalized using three components:

- *a read-only database (DB)*;
- *an artifact working memory (e.g., artifact variables + artifact relations)*;
- *actions* (also called *services*).



We formalize Artifact-Centric Systems in an array-based formal model. This encoding allows us to exploit the powerful machinery provided by SMT-techniques.

DB schemas

DB schemas model the *read-only DB* component of Artifact-Centric Systems, incorporating primary keys and foreign keys dependencies

DB schemas

DB schemas model the *read-only DB* component of Artifact-Centric Systems, incorporating primary keys and foreign keys dependencies

Definition

A DB schema is a pair (Σ, T), where:

- Σ is a *DB signature*, that is, a finite multi-sorted signature whose only symbols are equality, unary functions, and constants;
- T is a *DB theory*, that is, a set of universal Σ-sentences.

DB schemas

DB schemas model the *read-only DB* component of Artifact-Centric Systems, incorporating primary keys and foreign keys dependencies

Definition

A DB schema is a pair (Σ, T), where:

- Σ is a *DB signature*, that is, a finite multi-sorted signature whose only symbols are equality, unary functions, and constants;
- T is a *DB theory*, that is, a set of universal Σ-sentences.

We associate to a DB signature Σ a characteristic graph $G(\Sigma)$ capturing the dependencies induced by functions over sorts.

DB schemas

DB schemas model the *read-only DB* component of Artifact-Centric Systems, incorporating primary keys and foreign keys dependencies

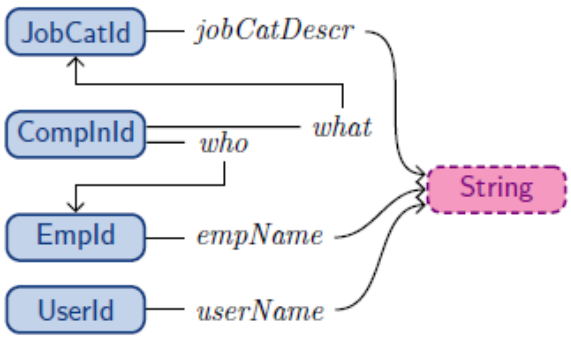
Definition

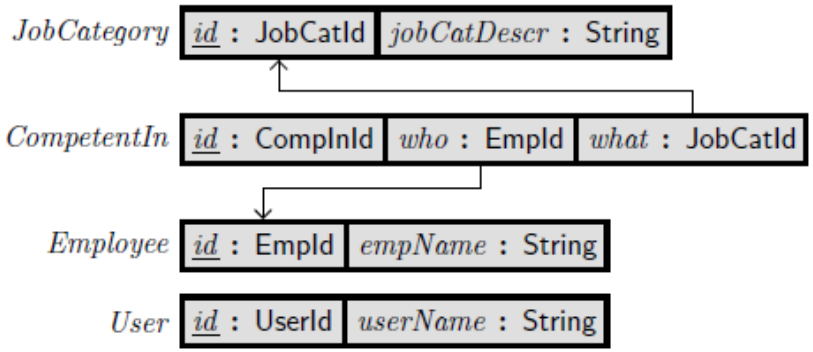
A DB schema is a pair (Σ, T), where:

- Σ is a *DB signature*, that is, a finite multi-sorted signature whose only symbols are equality, unary functions, and constants;
- T is a *DB theory*, that is, a set of universal Σ-sentences.

We associate to a DB signature Σ a characteristic graph $G(\Sigma)$ capturing the dependencies induced by functions over sorts.

Example:





Array-based Artifact-Centric Systems: a simplified version

Simplified version of Artifact-Centric Systems in the array-based setting: Simple Artifact Systems (SAS).

Array-based Artifact-Centric Systems: a simplified version

Simplified version of Artifact-Centric Systems in the array-based setting: Simple Artifact Systems (SAS).
A **SAS** is a tuple

$$\mathcal{S} \;=\; \langle \Sigma, T, \underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}') \rangle$$

where (Σ, T) is a **(read-only) DB schema**, $\underline{x}$ are *individual* variables (for **artifact variables**, i.e. the working memory).
The Σ-formula ι represents the ***initialization*** of $\mathcal{S}$, whereas the Σ-formula τ models the ***transitions*** (actions) of $\mathcal{S}$.

Array-based Artifact-Centric Systems: a simplified version

Simplified version of Artifact-Centric Systems in the array-based setting: Simple Artifact Systems (SAS).
A **SAS** is a tuple

$$\mathcal{S} \;=\; \langle \Sigma, T, \underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}') \rangle$$

where (Σ, T) is a **(read-only) DB schema**, $\underline{x}$ are *individual* variables (for **artifact variables**, i.e. the working memory).
The Σ-formula ι represents the ***initialization*** of $\mathcal{S}$, whereas the Σ-formula τ models the ***transitions*** (actions) of $\mathcal{S}$.

Example of ι and τ in a SAS:

$$\iota := (\mathsf{Applicant} = undef \wedge \mathsf{JobPos} = undef)$$

$$\tau := \exists \mathsf{UserID}, \mathsf{JobID} \begin{pmatrix} \mathsf{UserID} \neq undef \wedge \mathsf{JobID} \neq undef \wedge \mathsf{Applicant} = undef \wedge \\ \mathsf{JobPos} = undef \wedge \mathsf{Applicant}' := \mathsf{UserID} \wedge \mathsf{JobPos}' := \mathsf{JobID}) \end{pmatrix}$$

Array-based Artifact-Centric Systems: a simplified version

Simplified version of Artifact-Centric Systems in the array-based setting: Simple Artifact Systems (SAS).
A **SAS** is a tuple

$$\mathcal{S} \;=\; \langle \Sigma, T, \underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}') \rangle$$

where (Σ, T) is a **(read-only) DB schema**, $\underline{x}$ are *individual* variables (for **artifact variables**, i.e. the working memory).
The Σ-formula ι represents the ***initialization*** of $\mathcal{S}$, whereas the Σ-formula τ models the ***transitions*** (actions) of $\mathcal{S}$.

Example of ι and τ in a SAS:

$$\iota := (\mathsf{Applicant} = undef \wedge \mathsf{JobPos} = undef)$$

$$\tau := \exists \mathsf{UserID}, \mathsf{JobID} \begin{pmatrix} \mathsf{UserID} \neq undef \wedge \mathsf{JobID} \neq undef \wedge \mathsf{Applicant} = undef \wedge \\ \mathsf{JobPos} = undef \wedge \mathsf{Applicant}' := \mathsf{UserID} \wedge \mathsf{JobPos}' := \mathsf{JobID}) \end{pmatrix}$$

Array-based Artifact-Centric Systems: a simplified version

Simplified version of Artifact-Centric Systems in the array-based setting: Simple Artifact Systems (SAS).
A **SAS** is a tuple

$$\mathcal{S} \;=\; \langle \Sigma, T, \underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x'}) \rangle$$

where (Σ, T) is a **(read-only) DB schema**, $\underline{x}$ are *individual* variables (for **artifact variables**, i.e. the working memory).
The Σ-formula ι represents the ***initialization*** of $\mathcal{S}$, whereas the Σ-formula τ models the ***transitions*** (actions) of $\mathcal{S}$.

Example of ι and τ in a SAS:

$$\iota := (\mathsf{Applicant} = undef \wedge \mathsf{JobPos} = undef)$$

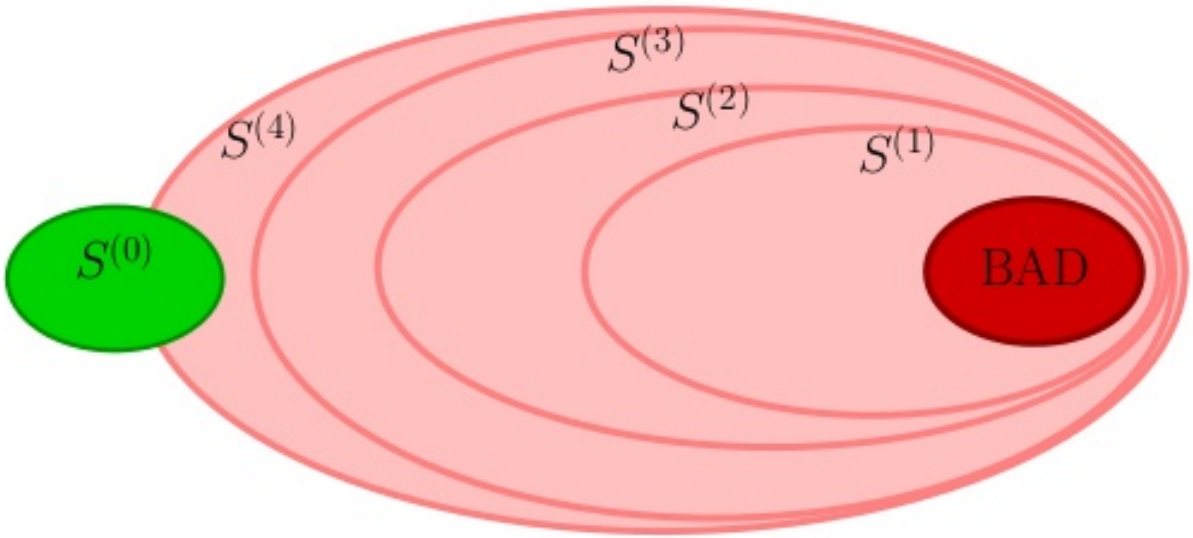
$$\tau := \exists \mathsf{UserID}, \mathsf{JobID} \begin{pmatrix} \mathsf{UserID} \neq undef \wedge \mathsf{JobID} \neq undef \wedge \mathsf{Applicant} = undef \wedge \\ \mathsf{JobPos} = undef \wedge \mathsf{Applicant}' := \mathsf{UserID} \wedge \mathsf{JobPos}' := \mathsf{JobID}) \end{pmatrix}$$

Outline

Backward Reachability: intuition

Backward-Reachability ($S^{(0)} \equiv$ "bad states")

Safety Check	If $S^{(i)}$ contains an initial, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T^{-1}(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Verification of safety in a SAS $\mathcal{S}$

A *safety* formula for $\mathcal{S}$ is a *generic* quantifier-free formula $\upsilon(\underline{x})$ (intuitively, $\upsilon(\underline{x})$ describes *undesired states* of $\mathcal{S}$).

Verification of safety in a SAS $\mathcal{S}$

A *safety* formula for $\mathcal{S}$ is a *generic* quantifier-free formula $\upsilon(\underline{x})$ (intuitively, $\upsilon(\underline{x})$ describes *undesired states* of $\mathcal{S}$).

We say that $\mathcal{S}$ is safe with respect to υ iff there is no DB-instance $\mathcal{M}$ of (Σ, T), no $k \geq 0$ and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \ldots, \underline{x}^k$ s.t.

$$\iota(\underline{x}^0) \wedge \tau(\underline{x}^0, \underline{x}^1) \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{x}^k) \wedge \upsilon(\underline{x}^k)$$

is true in $\mathcal{M}$ (here the $\underline{x}^i$ are renamed copies of the $\underline{x}$).

Verification of safety in a SAS $\mathcal{S}$

A *safety* formula for $\mathcal{S}$ is a *generic* quantifier-free formula $\upsilon(\underline{x})$ (intuitively, $\upsilon(\underline{x})$ describes *undesired states* of $\mathcal{S}$).

We say that $\mathcal{S}$ is safe with respect to υ iff there is no DB-instance $\mathcal{M}$ of (Σ, T), no $k \geq 0$ and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \ldots, \underline{x}^k$ s.t.

$$\iota(\underline{x}^0) \wedge \tau(\underline{x}^0, \underline{x}^1) \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{x}^k) \wedge \upsilon(\underline{x}^k)$$

is true in $\mathcal{M}$ (here the $\underline{x}^i$ are renamed copies of the $\underline{x}$).

Safety problem for $\mathcal{S}$: *given a safety formula υ, decide if $\mathcal{S}$ is safe wrt υ.*

Verification of safety in a SAS $\mathcal{S}$

A *safety* formula for $\mathcal{S}$ is a *generic* quantifier-free formula $\upsilon(\underline{x})$ (intuitively, $\upsilon(\underline{x})$ describes *undesired states* of $\mathcal{S}$).

We say that $\mathcal{S}$ is safe with respect to υ iff there is no DB-instance $\mathcal{M}$ of (Σ, T), no $k \geq 0$ and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \ldots, \underline{x}^k$ s.t.

$$\iota(\underline{x}^0) \wedge \tau(\underline{x}^0, \underline{x}^1) \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{x}^k) \wedge \upsilon(\underline{x}^k)$$

is true in $\mathcal{M}$ (here the $\underline{x}^i$ are renamed copies of the $\underline{x}$).

Safety problem for $\mathcal{S}$: *given a safety formula υ, decide if $\mathcal{S}$ is safe wrt υ.*

Theorem

Backward search is effective, correct and complete (the last one w.r.t. detecting unsafety) for solving safety problems for SASs

Verification of safety in a SAS $\mathcal{S}$

A *safety* formula for $\mathcal{S}$ is a *generic* quantifier-free formula $\upsilon(\underline{x})$ (intuitively, $\upsilon(\underline{x})$ describes *undesired states* of $\mathcal{S}$).

We say that $\mathcal{S}$ is safe with respect to υ iff there is no DB-instance $\mathcal{M}$ of (Σ, T), no $k \geq 0$ and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \ldots, \underline{x}^k$ s.t.

$$\iota(\underline{x}^0) \wedge \tau(\underline{x}^0, \underline{x}^1) \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{x}^k) \wedge \upsilon(\underline{x}^k)$$

is true in $\mathcal{M}$ (here the $\underline{x}^i$ are renamed copies of the $\underline{x}$).

Safety problem for $\mathcal{S}$: *given a safety formula υ, decide if $\mathcal{S}$ is safe wrt υ.*

Theorem

Backward search is effective, correct and complete (the last one w.r.t. detecting unsafety) for solving safety problems for SASs

The proof requires **Quantifier Elimination**!

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:
  - A state formula has the form $\phi := \psi(\underline{x})$;

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:
  - A state formula has the form $\phi := \psi(\underline{x})$;
  - A transition formula has the form
    $\tau := \exists \underline{d} \left(\gamma(\underline{d}, \underline{x}) \wedge \bigwedge_i x'_i = F_i(\underline{d}, \underline{x}) \right)$;

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:
  - A state formula has the form $\phi := \psi(\underline{x})$;
  - A transition formula has the form

    $\tau := \exists \underline{d} \left(\gamma(\underline{d}, \underline{x}) \wedge \bigwedge_i x'_i = F_i(\underline{d}, \underline{x}) \right)$;
  - $Pre(\tau, \phi) := \exists \underline{x}' (\tau(\underline{x}, \underline{x}') \wedge \phi(\underline{x}'))$.

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:
  - A state formula has the form $\phi := \psi(\underline{x})$;
  - A transition formula has the form
    $\tau := \exists \underline{d} \left(\gamma(\underline{d}, \underline{x}) \wedge \bigwedge_i x'_i = F_i(\underline{d}, \underline{x})\right)$;
  - $Pre(\tau, \phi) := \exists \underline{x}'(\tau(\underline{x}, \underline{x}') \wedge \phi(\underline{x}'))$.
- Therefore, during the run of the algorithm the tail of existential quantifiers can grow dramatically: this can affect not only the **performance**, but also **correctness** (*regressability*) and **termination**.

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:
  - A state formula has the form $\phi := \psi(\underline{x})$;
  - A transition formula has the form
    $\tau := \exists \underline{d} \left(\gamma(\underline{d}, \underline{x}) \wedge \bigwedge_i x'_i = F_i(\underline{d}, \underline{x})\right)$;
  - $Pre(\tau, \phi) := \exists \underline{x}'(\tau(\underline{x}, \underline{x}') \wedge \phi(\underline{x}'))$.

- Therefore, during the run of the algorithm the tail of existential quantifiers can grow dramatically: this can affect not only the **performance**, but also **correctness** (*regressability*) and **termination**.

- Hence, we need to handle these quantifiers: in order to do that, we compute quantifier elimination in the **model completion** $T^\star$ of DB theories T.

Why is Quantifier Elimination needed?

- Backward search computes symbolic pre-images of sets of states. This computation produces new existentially quantified **"data"** variables:
  - A state formula has the form $\phi := \psi(\underline{x})$;
  - A transition formula has the form
    $\tau := \exists \underline{d} \left(\gamma(\underline{d}, \underline{x}) \wedge \bigwedge_i x'_i = F_i(\underline{d}, \underline{x})\right)$;
  - $Pre(\tau, \phi) := \exists \underline{x}'(\tau(\underline{x}, \underline{x}') \wedge \phi(\underline{x}'))$.
- Therefore, during the run of the algorithm the tail of existential quantifiers can grow dramatically: this can affect not only the **performance**, but also **correctness** (*regressability*) and **termination**.
- Hence, we need to handle these quantifiers: in order to do that, we compute quantifier elimination in the **model completion** $T^\star$ of DB theories T.

Remark

*Detecting **unsafe conditions** in T or in $T^\star$ are equivalent problems!*

Model completions, Covers and Uniform Interpolation

Fix a theory T and an existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.

- We call a *residue* of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ any quantifier-free formula in $Res(\exists \underline{e}\, \phi) = \{\theta(\underline{y}, \underline{z}) \mid T \models \phi(\underline{e}, \underline{y}) \rightarrow \theta(\underline{y}, \underline{z})\}$. A quantifier-free formula $\psi(\underline{y})$ is a T-***cover*** of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ iff $\psi(\underline{y}) \in Res(\exists \underline{e}\, \phi)$ and $\psi(\underline{y})$ implies (modulo T) all the other formulae in $Res(\exists \underline{e}\, \phi)$.

Model completions, Covers and Uniform Interpolation

Fix a theory T and an existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.

- We call a *residue* of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ any quantifier-free formula in $Res(\exists \underline{e}\, \phi) = \{\theta(\underline{y}, \underline{z}) \mid T \models \phi(\underline{e}, \underline{y}) \rightarrow \theta(\underline{y}, \underline{z})\}$. A quantifier-free formula $\psi(\underline{y})$ is a T-***cover*** of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ iff $\psi(\underline{y}) \in Res(\exists \underline{e}\, \phi)$ and $\psi(\underline{y})$ implies (modulo T) all the other formulae in $Res(\exists \underline{e}\, \phi)$.

- We say that a theory T has *uniform quantifier-free interpolation* iff every existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ has a T-cover.

Model completions, Covers and Uniform Interpolation

Fix a theory T and an existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.

- We call a *residue* of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ any quantifier-free formula in $Res(\exists \underline{e}\, \phi) = \{\theta(\underline{y}, \underline{z}) \mid T \models \phi(\underline{e}, \underline{y}) \rightarrow \theta(\underline{y}, \underline{z})\}$. A quantifier-free formula $\psi(\underline{y})$ is a T-***cover*** of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ iff $\psi(\underline{y}) \in Res(\exists \underline{e}\, \phi)$ and $\psi(\underline{y})$ implies (modulo T) all the other formulae in $Res(\exists \underline{e}\, \phi)$.

- We say that a theory T has *uniform quantifier-free interpolation* iff every existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ has a T-cover. It is clear that if T has uniform quantifier-free interpolation, then it has ordinary quantifier-free interpolation.

Model completions, Covers and Uniform Interpolation

Fix a theory T and an existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.

- We call a *residue* of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ any quantifier-free formula in $Res(\exists \underline{e}\, \phi) = \{\theta(\underline{y}, \underline{z}) \mid T \models \phi(\underline{e}, \underline{y}) \rightarrow \theta(\underline{y}, \underline{z})\}$. A quantifier-free formula $\psi(\underline{y})$ is a T-***cover*** of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ iff $\psi(\underline{y}) \in Res(\exists \underline{e}\, \phi)$ and $\psi(\underline{y})$ implies (modulo T) all the other formulae in $Res(\exists \underline{e}\, \phi)$.

- We say that a theory T has *uniform quantifier-free interpolation* iff every existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ has a T-cover. It is clear that if T has uniform quantifier-free interpolation, then it has ordinary quantifier-free interpolation.

Theorem

Suppose that T is a universal theory. Then, T has a model completion T^* ***iff*** *T has uniform quantifier-free interpolation. If this happens, T^* is axiomatized by the infinitely many sentences $\forall \underline{y}\, (\psi(\underline{y}) \rightarrow \exists \underline{e}\, \phi(\underline{e}, \underline{y}))$, where $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ is a primitive formula and ψ is a* ***cover*** *of it.*

Outline

Preprocessing of formulae

- **Flattening** of terms:

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:
  - $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:
  - $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);
  - $E(t, u)$ fails if $t \equiv e_i$ and (either $t \equiv f(t_1, \ldots, t_k)$ or $u \equiv e_j$ for $i \neq j$);

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:
  - $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);
  - $E(t, u)$ fails if $t \equiv e_i$ and (either $t \equiv f(t_1, \ldots, t_k)$ or $u \equiv e_j$ for $i \neq j$);
  - $E(t, u) = \emptyset$ if $t \equiv u$;

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:
  - $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);
  - $E(t, u)$ fails if $t \equiv e_i$ and (either $t \equiv f(t_1, \ldots, t_k)$ or $u \equiv e_j$ for $i \neq j$);
  - $E(t, u) = \emptyset$ if $t \equiv u$;
  - $E(t, u) = \{t = u\}$ if t and u are different but both $\underline{e}$-free;

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:
  - $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);
  - $E(t, u)$ fails if $t \equiv e_i$ and (either $t \equiv f(t_1, \ldots, t_k)$ or $u \equiv e_j$ for $i \neq j$);
  - $E(t, u) = \emptyset$ if $t \equiv u$;
  - $E(t, u) = \{t = u\}$ if t and u are different but both $\underline{e}$-free;
  - $E(t, u)$ fails if none of t, u is $\underline{e}$-free, $t \equiv f(t_1, \ldots, t_k)$ and $u \equiv g(u_1, \ldots, u_l)$ for $f \not\equiv g$;

Preprocessing of formulae

- **Flattening** of terms:
  - definition of $\underline{e}$-free terms;
  - definition of $\underline{e}$-flat terms;
  - an $\underline{e}$-flat literal is a literal of the form $t = a$ or $a \neq b$, where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

  For example, $f(t(y), e) = e$, where f is a function symbol and t a generic term, is an $\underline{e}$-flat literal.
- Given two $\underline{e}$-flat terms t, u, $E(t, u)$ is defined as follows:
  - $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);
  - $E(t, u)$ fails if $t \equiv e_i$ and (either $t \equiv f(t_1, \ldots, t_k)$ or $u \equiv e_j$ for $i \neq j$);
  - $E(t, u) = \emptyset$ if $t \equiv u$;
  - $E(t, u) = \{t = u\}$ if t and u are different but both $\underline{e}$-free;
  - $E(t, u)$ fails if none of t, u is $\underline{e}$-free, $t \equiv f(t_1, \ldots, t_k)$ and $u \equiv g(u_1, \ldots, u_l)$ for $f \not\equiv g$;
  - $E(t, u) = E(t_1, u_1) \cup \cdots \cup E(t_k, u_k)$ if none of t, u is $\underline{e}$-free, $t \equiv f(t_1, \ldots, t_k)$, $u \equiv f(u_1, \ldots, u_k)$ and none of the $E(t_i, u_i)$ fails.

Constrained Superposition Calculus

The rules of our Constrained Superposition Calculus (**constr-SC**) follow (each rule applies provided the E subprocedure called by it does not fail):

Constrained Superposition Calculus

The rules of our Constrained Superposition Calculus (**constr-SC**) follow (each rule applies provided the E subprocedure called by it does not fail):

Superposition Right (Constrained)

$$\frac{l = r \parallel C \qquad s = t \parallel D}{s[r]_p = t \parallel C \cup D \cup E(s_{|p}, l)} \quad \text{if } l > r \text{ and } s > t$$

Superposition Left (Constrained)

$$\frac{l = r \parallel C \qquad s \neq t \parallel D}{s[r]_p \neq t \parallel C \cup D \cup E(s_{|p}, l)} \quad \text{if } l > r \text{ and } s > t$$

Reflexion (Constrained)

$$\frac{t \neq u \parallel C}{\bot \parallel C \cup E(t, u)}$$

Demodulation (Constrained)

$$\frac{L \parallel C, \qquad l = r \parallel D}{L[r]_p \parallel C} \quad \text{if } l > r, L_{|p} \equiv l \text{ and } C \supseteq D$$

Constrained Superposition Calculus

The rules of our Constrained Superposition Calculus (**constr-SC**) follow (each rule applies provided the E subprocedure called by it does not fail):

Superposition Right (Constrained)

$$\frac{l = r \parallel C \qquad s = t \parallel D}{s[r]_p = t \parallel C \cup D \cup E(s_{|p}, l)} \quad \text{if } l > r \text{ and } s > t$$

Superposition Left (Constrained)

$$\frac{l = r \parallel C \qquad s \neq t \parallel D}{s[r]_p \neq t \parallel C \cup D \cup E(s_{|p}, l)} \quad \text{if } l > r \text{ and } s > t$$

Reflexion (Constrained)

$$\frac{t \neq u \parallel C}{\bot \parallel C \cup E(t, u)}$$

Demodulation (Constrained)

$$\frac{L \parallel C, \qquad l = r \parallel D}{L[r]_p \parallel C} \quad \text{if } l > r, L_{|p} \equiv l \text{ and } C \supseteq D$$

Remark: termination requires a precise ***application strategy*** for the rules.

Termination and Correctness

There are in principle infinitely many $\underline{e}$-flat terms that can be generated during saturation. However, thanks to the application strategy, we prove:

Termination and Correctness

There are in principle infinitely many $\underline{e}$-flat terms that can be generated during saturation. However, thanks to the application strategy, we prove:

Proposition

The saturation of the initial set of $\underline{e}$-flat constrained literals always terminates after finitely many steps.

Termination and Correctness

There are in principle infinitely many $\underline{e}$-flat terms that can be generated during saturation. However, thanks to the application strategy, we prove:

Proposition

The saturation of the initial set of $\underline{e}$-flat constrained literals always terminates after finitely many steps.

Relying on model-theoretic techniques and the completeness theorem of standard SC, we get:

Termination and Correctness

There are in principle infinitely many $\underline{e}$-flat terms that can be generated during saturation. However, thanks to the application strategy, we prove:

Proposition

The saturation of the initial set of $\underline{e}$-flat constrained literals always terminates after finitely many steps.

Relying on model-theoretic techniques and the completeness theorem of standard SC, we get:

Theorem

*Suppose that **constr-SC**, taking as input the primitive $\underline{e}$-flat formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$, gives as output the quantifier-free formula $\psi(\underline{y})$. Then the latter is a **cover** of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.*

Termination and Correctness

There are in principle infinitely many $\underline{e}$-flat terms that can be generated during saturation. However, thanks to the application strategy, we prove:

Proposition

The saturation of the initial set of $\underline{e}$-flat constrained literals always terminates after finitely many steps.

Relying on model-theoretic techniques and the completeness theorem of standard SC, we get:

Theorem

*Suppose that **constr-SC**, taking as input the primitive $\underline{e}$-flat formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$, gives as output the quantifier-free formula $\psi(\underline{y})$. Then the latter is a **cover** of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.*

This also proves the existence of the model completion of EUF. Notice also that EUF suffices to represent DB theories of our Artifact Systems

Outline

Conclusions

- We formalized Artifact-centric Systems in array-based systems in order to exploit a suitable version of the backward reachability procedure to attack **verification** problem of safety properties.

Conclusions

- We formalized Artifact-centric Systems in array-based systems in order to exploit a suitable version of the backward reachability procedure to attack **verification** problem of safety properties.
- We developed **quantifier elimination** procedures in model completions in order to retain soundness and completeness of the backward search.

Conclusions

- We formalized Artifact-centric Systems in array-based systems in order to exploit a suitable version of the backward reachability procedure to attack **verification** problem of safety properties.
- We developed **quantifier elimination** procedures in model completions in order to retain soundness and completeness of the backward search.
- We showed how **quantifier elimination in model completions** and **covers** are strictly related.

Conclusions

- We formalized Artifact-centric Systems in array-based systems in order to exploit a suitable version of the backward reachability procedure to attack **verification** problem of safety properties.
- We developed **quantifier elimination** procedures in model completions in order to retain soundness and completeness of the backward search.
- We showed how **quantifier elimination in model completions** and **covers** are strictly related.
- We computed covers for **EUF** adopting a **constrained version of the Superposition Calculus** with appropriate application strategies.

THANKS FOR YOUR ATTENTION!