Verification of Data-Aware Processes

Giuseppe De Giacomo

Sapienza Università di Roma
Rome, Italy

11th International Workshop on Web Services and Formal Methods:
Formal Aspects of Service-Oriented and Cloud Computing
WS-FM:FASOCC 2014

Eindhoven, The Netherlands, September 11-12, 2014
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Data and Processes

The information assets of an organization are constituted by:

- data, and
- processes, that determine how data changes and evolves over time.

“Conceptual Modeling”

Both aspects are modelled “conceptually”, but:

- Using different modeling tools
- By different teams with different competences
- Connection between the two is NOT modelled conceptually

Consequence

Full reasoning support, e.g., for verification taking into account both process and data, is not possible!
Data and Processes

The information assets of an organization are constituted by:

- **data**, and
- **processes**, that determine how data changes and evolves over time.

"Conceptual Modeling"

Both aspects are modelled “conceptually”, but:

- Using different modeling tools
- By different teams with different competences
- Connection between the two is NOT modelled conceptually

Consequence

Full reasoning support, e.g., for verification taking into account both process and data, is not possible!
Data and Processes

The information assets of an organization are constituted by:

- **data**, and
- **processes**, that determine how data changes and evolves over time.

### “Conceptual Modeling”

Both aspects are modelled “conceptually”, but:

- Using different modeling tools
- By different teams with different competences
- **Connection between the two is NOT modelled conceptually**

### Consequence

Full reasoning support, e.g., for verification taking into account both process and data, is not possible!
Data Modeling

Example: (UML class diagram)

- Focus is on entities, relations, and static constraints that are relevant for the domain of interest.
- ER-diagrams, UML Class Diagrams, DBs, Data Integration.
- Knowledge Bases, Ontologies, OBDA, OBDI, Automated Reasoning

But how do data evolve?
Data Modeling

Example: (UML class diagram)

- Focus is on entities, relations, and static constraints that are relevant for the domain of interest.
- ER-diagrams, UML Class Diagrams, DBs, Data Integration.
- Knowledge Bases, Ontologies, OBDA, OBDI, Automated Reasoning

But how do data evolve?
Data Modeling

Example: (UML class diagram)

- Focus is on entities, relations, and static constraints that are relevant for the domain of interest.
- ER-diagrams, UML Class Diagrams, DBs, Data Integration.
- Knowledge Bases, Ontologies, OBDA, OBDI, Automated Reasoning

But how do data evolve?
Data Modeling

Example: (UML class diagram)

- Focus is on entities, relations, and static constraints that are relevant for the domain of interest.
- ER-diagrams, UML Class Diagrams, DBs, Data Integration.
- Knowledge Bases, Ontologies, OBDA, OBDI, Automated Reasoning

But how do data evolve?
Data Modeling

Example: (UML class diagram)

- Focus is on entities, relations, and static constraints that are relevant for the domain of interest.
- ER-diagrams, UML Class Diagrams, DBs, Data Integration.
- Knowledge Bases, Ontologies, OBDA, OBDI, Automated Reasoning

But how do data evolve?
Process Modeling

Example: (BPMN diagram)

- Focus is on control flow of activities that realize the business objectives.
- High level (Conceptual) models, BPMN diagrams, UML Activity Diagrams,
- Execution, monitoring, verification (some hacking needed).

But how are data manipulated?
Process Modeling

Example: (BPMN diagram)

- Focus is on control flow of activities that realize the business objectives.
- High level (Conceptual) models, BPMN diagrams, UML Activity Diagrams,
- Execution, monitoring, verification (some hacking needed).

But how are data manipulated?
Process Modeling

Example: (BPMN diagram)

- Focus is on **control flow** of activities that realize the business objectives.
- High level (Conceptual) models, **BPMN diagrams**, UML Activity Diagrams,
- **Execution, monitoring, verification** (some hacking needed).

But how are data manipulated?
**Process Modeling**

**Example: (BPMN diagram)**

- Focus is on control flow of activities that realize the business objectives.
- High level (Conceptual) models, BPMN diagrams, UML Activity Diagrams,
- Execution, monitoring, verification (some hacking needed).

But how are data manipulated?
Process Modeling

Example: (BPMN diagram)

- Focus is on control flow of activities that realize the business objectives.
- High level (Conceptual) models, BPMN diagrams, UML Activity Diagrams,
- Execution, monitoring, verification (some hacking needed).

But how are data manipulated?
Data and Processes

Data-Process Dichotomy in IT management [Forrester Report 2009]

- Business process management professionals: view data as subsidiary to processes manipulating them, and neglect importance of data quality.
- Data management experts: consider data as the driver of the organizational processes and are concerned about data quality only.

Data-Process Dichotomy has a negative impact

- Little collaboration between the teams
  - running the master data management initiatives, and
  - managing the business processes.
  Forrester: 83% . . . no interaction at all

- Little attention on the side of tool vendors to address the combined requirements:
  - Data management tools consider only the processes directly affecting the data in the tools, but not the actual business processes using the data,
  - Business process modeling suites do not allow for direct connection of data.
Data and Processes

Data-Process Dichotomy in IT management [Forrester Report 2009]

- **Business process management professionals**: view data as subsidiary to processes manipulating them, and neglect importance of data quality.
- **Data management experts**: consider data as the driver of the organizational processes and are concerned about data quality only.

Data-Process Dichotomy has a negative impact

- Little collaboration between the teams
  - running the master data management initiatives, and
  - managing the business processes.

  Forrester: 83% ... no interaction at all

- Little attention on the side of tool vendors to address the combined requirements:
  - Data management tools consider only the processes directly affecting the data in the tools, but not the actual business processes using the data,
  - Business process modeling suites do not allow for direct connection of data.
Data and Processes

The need for overcoming this dichotomy is recognized by the business process modeling community as well [Reichert 2012]:

“Process and data are just two sides of the same coin”

Two key areas where explicit representation of data in business processes is important [Meyer et al. 2011]:

1. Modeling the core assets of an organization.
   - Data is crucial for the execution of business processes that create value.
   - Hence the business processes need to access the data, and this should be accounted for explicitly.

2. Business process controlling.
   - Key performance indicators and business goals are defined in terms of data.
   - To evaluate and control them, the data objects relevant for the activities contributing to the goals need to be identified.
Data and Processes

The need for overcoming this dichotomy is recognized by the business process modeling community as well [Reichert 2012]:

“Process and data are just two sides of the same coin”

Two key areas where explicit representation of data in business processes is important [Meyer et al. 2011]:

1. **Modeling the core assets of an organization.**
   - Data is crucial for the execution of business processes that create value.
   - Hence the business processes need to access the data, and this should be accounted for explicitly.

2. **Business process controlling.**
   - Key performance indicators and business goals are defined in terms of data.
   - To evaluate and control them, the data objects relevant for the activities contributing to the goals need to be identified.
Data and Processes

- Conventional modeling: separation between data and processes.
- Connection is handled at the technological level.
  - Lack of a conceptual view of data+processs that is holistic and coherent.
  - It becomes difficult to manage and maintain the system.

Example: build-to-order

![Diagram showing the process flow from Decompose Customer PO to Manage Material POs to Assemble to Ship to Manage Cancelation. The data and activities are represented in different colored blocks alongside various databases for Customers, Customer POs, Work Orders, Material POs, and Suppliers&Catalogues.]
Data and Processes

- Conventional modeling: separation between data and processes.
- Connection is handled at the technological level.

  ≈ Lack of a conceptual view of data+process that is **holistic** and **coherent**.
  ≈ It becomes difficult to manage and maintain the system.

**Example: build-to-order**

![Diagram](image-url)
Overcoming the dichotomy

Strong need for:

- suitable modeling formalisms supporting the integrated management of processes and data;
- methodologies for the design of systems based on such formalisms;
- systems and tools that implement these languages and methodologies.

This, in turn requires a foundational approach to:

- provide a clear understanding of (data-aware) process models w.r.t. semantic properties, and computational properties;
- enable analysis of such formalisms.
Overcoming the dichotomy

Strong need for:

- suitable modeling formalisms supporting the integrated management of processes and data;
- methodologies for the design of systems based on such formalisms;
- systems and tools that implement these languages and methodologies.

This, in turn requires a foundational approach to:

- provide a clear understanding of (data-aware) process models w.r.t.
  - semantic properties, and
  - computational properties;
- enable analysis of such formalisms.
Analysis

Data-related analysis is well-established in DB theory:

- intensional reasoning over queries: containment, equivalence
- database dependencies: axiomatization, satisfaction, implication, ...
- semantic and conceptual data models
- reasoning over views
- ...

Analysis of system dynamics is also of great interest: verification of software and hardware systems via model checking [2007 Turing Award: Clarke, Emerson, Sifakis]

- Dynamic properties of interest are formulated in a temporal logic (LTL, CTL, μ-calculus, ...).
- The transition system mathematically capturing the dynamics of the system of interest is (implicitly or explicitly) represented.
- The temporal logic formulas are checked (i.e., evaluated) over the transition system.

Model checking technology requires transition systems to be finite.
Analysis

Data-related analysis is well-established in DB theory:

- intensional reasoning over queries: containment, equivalence
- database dependencies: axiomatization, satisfaction, implication, ...
- semantic and conceptual data models
- reasoning over views
- ...

Analysis of system dynamics is also of great interest: verification of software and hardware systems via model checking [2007 Turing Award: Clarke, Emerson, Sifakis]

- Dynamic properties of interest are formulated in a temporal logic (LTL, CTL, μ-calculus, ...).
- The transition system mathematically capturing the dynamics of the system of interest is (implicitly or explicitly) represented.
- The temporal logic formulas are checked (i.e., evaluated) over the transition system.

Model checking technology requires transition systems to be finite.
**Analysis**

Data-related analysis is well-established in DB theory:

- intensional reasoning over queries: containment, equivalence
- database dependencies: axiomatization, satisfaction, implication, ...
- semantic and conceptual data models
- reasoning over views
- ...

Analysis of system dynamics is also of great interest: verification of software and hardware systems via model checking [2007 Turing Award: Clarke, Emerson, Sifakis]

- Dynamic properties of interest are formulated in a temporal logic (LTL, CTL, $\mu$-calculus, ...).
- The transition system mathematically capturing the dynamics of the system of interest is (implicitly or explicitly) represented.
- The temporal logic formulas are checked (i.e., evaluated) over the transition system.

Model checking technology requires transition systems to be finite.
Analysis in BPM

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

However:

- Data is abstracted away.
- Emphasis is on the control-flow dimension:
  \[ \sim \] sophisticated techniques for absence of deadlocks, boundedness, soundness, or domain-dependent properties expressed in LTL, CTL, \( \mu \)-calculus.

Basic assumption: control-flow is captured by a transition system:

- labels on transitions represent the process tasks/activities
- concurrency is represented by interleaving
- transition system usually not represented explicitly, but typically is implicitly “folded” into a Petri net
In BPM, process model analysis is considered the second most influential topic in the last decade (after process modeling languages) [van der Aalst 2012].

However:

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

Basic assumption: control-flow is captured by a transition system:

- labels on transitions represent the process tasks/activities
- concurrency is represented by interleaving
- transition system usually not represented explicitly, but typically is implicitly “folded” into a Petri net
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

We need first-order variants of temporal logics.

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

\[ \Rightarrow \text{We need first-order variants of temporal logics} \]

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

→ We need first-order variants of temporal logics

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

\( \rightarrow \) We need first-order variants of temporal logics

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

We need first-order variants of temporal logics

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

We need first-order variants of temporal logics.

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

We need first-order variants of temporal logics

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

• the expressiveness of the temporal property language, and
• the identification of classes of data-aware processes,

such that

1 we are to capture notable, real-world scenarios, but
2 analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

\( \rightsquigarrow \) We need first-order variants of temporal logics

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

\[\leadsto\text{We need first-order variants of temporal logics}\]

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

1. we are to capture notable, real-world scenarios, but
2. analysis stays decidable, and possibly efficient.
Analysis of data-aware processes

The presence of data complicates analysis significantly:

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

\[ \sim \text{We need first-order variants of temporal logics} \]

Model checking data-aware processes becomes immediately undecidable!

How can we mediate between:

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

such that

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

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Artifact-Centric Approach

• In early 2000, the artifact-centric approach emerged as a foundational proposal for merging data and processes together.
  ▶ The emphasis is on data, which are modeled taking into account that they will be manipulated by processes.
  ▶ Processes are modeled by considering how they manipulate data.

• Initial proposals by IBM [NigamCaswell03], followed by [Bhattacharya et al. 07]; [Deutsch et al. 07]; [KünzleReichert11], . . .

• Key results within the EU project ACSI, 2010–2013

![ASI](Artifact-Centric Service Interoperation)
What is an artifact?

**Definition**
A key, business-relevant conceptual dynamic entity that is used in guiding the operation of a business.

Consists of:
- **Information model** - relevant data maintained by the artifact
- **Lifecycle model** - (implicit) description of the allowed information model evolutions through the execution of a process.

Goal: unified, end-to-end view of relevant entities and their possible evolutions.
Concrete models for artifacts

Key questions:

- How and where to store data maintained by their information models?
- How to specify the lifecycle of such artifacts?
- At which level of abstraction?

Some concrete information models:

- Relational database (with nested records).
- Knowledge base, e.g., expressed in some ontology language.

Some concrete lifecycle models:

- Finite-state machines. State = phase; events trigger transitions. ▶ Implemented in the *Siena* prototype by IBM.
- Guard-Stage-Milestone lifecycles, based on declarative (event-condition-action)-like rules. ▶ Implemented in the *Barcelona* prototype by IBM.
- Proclets (interacting Petri nets). ▶ Emphasise many-to-many relationships between artifacts.
Concrete models for artifacts

Key questions:

• How and where to store data maintained by their information models?
• How to specify the lifecycle of such artifacts?
• At which level of abstraction?

Some concrete information models:

• Relational database (with nested records).
• Knowledge base, e.g., expressed in some ontology language.

Some concrete lifecycle models:

• Finite-state machines. State = phase; events trigger transitions.
  ▶ Implemented in the Siena prototype by IBM.
• Guard-Stage-Milestone lifecycles, based on declarative (event-condition-action)-like rules.
  ▶ Implemented in the Barcelona prototype by IBM.
• Proclets (interacting Petri nets).
  ▶ Emphasise many-to-many relationships between artifacts.
Concrete models for artifacts

Key questions:

• How and where to store data maintained by their information models?
• How to specify the lifecycle of such artifacts?
• At which level of abstraction?

Some concrete information models:

• Relational database (with nested records).
• Knowledge base, e.g., expressed in some ontology language.

Some concrete lifecycle models:

• Finite-state machines. State = phase; events trigger transitions.
  ▶ Implemented in the Siena prototype by IBM.
• Guard-Stage-Milestone lifecycles, based on declarative (event-condition-action)-like rules.
  ▶ Implemented in the Barcelona prototype by IBM.
• Proclets (interacting Petri nets).
  ▶ Emphasise many-to-many relationships between artifacts.
Reasoning about artifacts as dynamic entities

We want to provide a formal foundation for artifact-centric systems, and provide corresponding reasoning facilities for their trustworthy design.

In particular, we want to decide whether dynamic/temporal properties of interest hold over the life of such systems.

- Verification of temporal formulae.
- Dominance/simulation/bisimulation/containment properties.
- Automated composition of artifacts-based systems.
- Automated process synthesis from dynamic/temporal specifications.
Verification of artifacts is tough

What is an example of a finite-state control process manipulating possibly unbounded data? A Turing Machine

Verification of the propositional CTL / LTL reachability property “eventually milestone $\textit{Halt}$ achieved” is undecidable.
Verification of artifacts is tough

What is an example of a finite-state control process manipulating possibly unbounded data? A Turing Machine

Verification of the propositional CTL / LTL reachability property “eventually milestone Halt achieved” is undecidable.
Artifact formal foundations

Is there hope to find interesting decidable cases?

• This requires to identify “classes of systems” that enjoy verifiability.
• First step: devise a minimal, clean mathematical framework as the basis of investigation.
• Many research groups working on this:
  ▶ University California San Diego,
  ▶ University California Santa Barbara,
  ▶ IBM Watson,
  ▶ Imperial College,
  ▶ Sapienza Università di Roma,
  ▶ Free University of Bozen-Bolzano.
• Starting from previous work, we have defined a rich but “pristine” formal framework: Data-Centric Dynamic Systems.
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Data-Centric Dynamic Systems (DCDS)

- **Abstract model** underlying variants of artifact-centric systems.
- **Semantically equivalent to the most expressive models** for business process systems (e.g., GSM).

- **Data Layer**: Relational databases
  - Data schema, specifying constraints on the allowed states
  - Data instance: state of the DCDS

- **Process Layer**: key elements are
  - Atomic actions
  - Condition-action-rules for application of actions
  - Service calls: communication with external environment, new data!
Data-Centric Dynamic Systems (DCDS)

- Abstract model underlying variants of artifact-centric systems.
- Semantically equivalent to the most expressive models for business process systems (e.g., GSM).

- Data Layer: Relational databases
  - Data schema, specifying constraints on the allowed states
  - Data instance: state of the DCDS

- Process Layer: key elements are
  - Atomic actions
  - Condition-action-rules for application of actions
  - Service calls: communication with external environment, new data!
Data-Centric Dynamic Systems (DCDS)

- Abstract model underlying variants of artifact-centric systems.
- Semantically equivalent to the most expressive models for business process systems (e.g., GSM).

- Data Layer: Relational databases
  - Data schema, specifying constraints on the allowed states
  - Data instance: state of the DCDS

- Process Layer: key elements are
  - Atomic actions
  - Condition-action-rules for application of actions
  - Service calls: communication with external environment, new data!
Data-Centric Dynamic Systems (DCDS)

- **Abstract model** underlying variants of artifact-centric systems.
- **Semantically equivalent to the most expressive models** for business process systems (e.g., GSM).

### Data Layer: Relational databases
- Data schema, specifying constraints on the allowed states
- Data instance: state of the DCDS

### Process Layer: key elements are
- Atomic actions
- Condition-action-rules for application of actions
- Service calls: communication with external environment, new data!
Data-Centric Dynamic Systems (DCDS)

- **Abstract model** underlying variants of artifact-centric systems.
- **Semantically equivalent to the most expressive models** for business process systems (e.g., GSM).

- **Data Layer**: Relational databases
  - Data schema, specifying constraints on the allowed states
  - Data instance: state of the DCDS

- **Process Layer**: key elements are
  - Atomic actions
  - Condition-action-rules for application of actions
  - **Service calls**: communication with external environment, new data!
DCDS: Example

Schema

peer

Customer

In Debt Customer

Loan

Gold Customer

Data Layer

Instance

Cust(ann)
peer(mark, john)
Gold(john)

...
DCDS: Example

Data Layer

Schema

peer

Customer

In Debt Customer

Loan

Gold Customer

Instance

Cust(ann)
peer(mark, john)
Gold(john)

...
DCDS: Example

Process Layer

Condition Action Rules

peer\(x, y\) \land Gold(y)
\iff GetLoan(x)

Service Calls

\(UInput(x)\)

Data Layer

Schema

peer

Customer

In Debt Customer

Loan

Gold Customer

Instance

Cust(ann)
peer(mark, john)
Gold(john)

\ldots

Actions

GetLoan(x):

\exists y. peer(x, y) \iff \{ owes(x, UInput(x)) \},

Cust(z) \iff \{ Cust(z) \},

Loan(z) \iff \{ Loan(z) \},

InDebt(z) \iff \{ InDebt(z) \},

Gold(z) \iff \{ Gold(z) \}
DCDS: Example

Process Layer

Condition Action Rules

peer(x, y) ∧ Gold(y) → GetLoan(x)

Service Calls

UInput(x)

Data Layer

Schema

peer

Customer

In Debt Customer

Loan

Gold Customer

Instance

Cust(ann)
peer(mark, john)
Gold(john)

...
DCDS: Example

**Data Layer**
- **Schema**
  - peer
    - Customer
    - In Debt Customer
    - Loan
    - Gold Customer

- **Instance**
  - Cust(ann)
  - peer(mark, john)
  - Gold(john)

**Process Layer**

**Condition Action Rules**

- peer\((x, y) \land Gold(y)\) \implies GetLoan\((x)\)

**Actions**

- GetLoan\((x)\):
  - \(\exists y.\) peer\((x, y) \implies \{\text{owes}(x, UInput(x))\}\),
  - Cust\((z) \implies \{\text{Cust}(z)\}\),
  - Loan\((z) \implies \{\text{Loan}(z)\}\),
  - InDebt\((z) \implies \{\text{InDebt}(z)\}\),
  - Gold\((z) \implies \{\text{Gold}(z)\}\)

**Service Calls**

- UInput\((x)\)
DCDS: Example

Data Layer

Schema

peer

Customer

In Debt Customer

Loan

Gold Customer

Instance

Cust(ann)

peer(mark, john)

Gold(john)

owes(mark, @25)

...

Process Layer

Condition Action Rules

peer(x, y) \land Gold(y)

\implies GetLoan(x)

Service Calls

UInput(x)

GetLoan(x) :

\exists y. peer(x, y) \implies \{ owes(x, UInput(x)), Cust(z), Loan(z), InDebt(z), Gold(z) \}
Deterministic vs. non-deterministic services

DCDSs admit two different semantics for service-execution:

**Deterministic services semantics**

Along a run, when the same service is called again with the same arguments, it returns the same result as in the previous call.

Are used to model an environment whose behavior is completely determined by the parameters.

**Example:** temperature, given the location and the date and time

**Non-deterministic services semantics**

Along a run, when the same service is called again with the same arguments, it may return a different value than in the previous call.

Are used to model:

- an environment whose behavior is determined by parameters that are outside the control of the system;
- input of external users, whose choices depend on external factors.

**Example:** current temperature, given the location
Deterministic vs. non-deterministic services

DCDSs admit two different semantics for service-execution:

**Deterministic services semantics**

Along a run, when the same service is called again with the same arguments, it returns the same result as in the previous call.

Are used to model an environment whose behavior is completely determined by the parameters.

Example: temperature, given the location and the date and time

**Non-deterministic services semantics**

Along a run, when the same service is called again with the same arguments, it may return a different value than in the previous call.

Are used to model:

- an environment whose behavior is determined by parameters that are outside the control of the system;
- input of external users, whose choices depend on external factors.

Example: current temperature, given the location
Semantics via transition systems

Semantics of a DCDS $S$ is given in terms of a transition system $\Upsilon_S$:

- each state of $\Upsilon_S$ has an associated DB over a common schema;
- the initial state is associated to the initial DB of the DCDS.

Note: $\Upsilon_S$ is in general infinite state:

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

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

- infinite branching
- infinite runs.

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

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

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are **bisimilar** if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s^A_1$ of $A$ such that $s^A \xrightarrow{A} s^A_1$, then there exists a state $s^B_1$ of $B$ such that $s^B \xrightarrow{B} s^B_1$, and $s^A_1$ and $s^B_1$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are bisimilar if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_1^A$ of $A$ such that $s^A \Rightarrow_A s_1^A$, then there exists a state $s_1^B$ of $B$ such that $s^B \Rightarrow_B s_1^B$, and $s_1^A$ and $s_1^B$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are **bisimilar** if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_1^A$ of $A$ such that $s^A \Rightarrow_A s_1^A$, then there exists a state $s_1^B$ of $B$ such that $s^B \Rightarrow_B s_1^B$, and $s_1^A$ and $s_1^B$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are bisimilar if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_1^A$ of $A$ such that $s^A \Rightarrow_A s_1^A$, then there exists a state $s_1^B$ of $B$ such that $s^B \Rightarrow_B s_1^B$, and $s_1^A$ and $s_1^B$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.

\[ \text{Theorem (bisimulation invariance)} \]

Bisimilar transition systems satisfy the same $\mu_L$ formulas, where $\mu_L$ is a FO variant of $\mu$-calculus with quantification ranging on active domain only, and no quantification across states.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are bisimilar if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_1^A$ of $A$ such that $s^A \Rightarrow_A s_1^A$, then there exists a state $s_1^B$ of $B$ such that $s^B \Rightarrow_B s_1^B$, and $s_1^A$ and $s_1^B$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are bisimilar if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_A^1$ of $A$ such that $s^A \xrightarrow{A} s_A^1$, then there exists a state $s_B^1$ of $B$ such that $s^B \xrightarrow{B} s_B^1$, and $s_A^1$ and $s_B^1$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are **bisimilar** if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_1^A$ of $A$ such that $s^A \Rightarrow_A s_1^A$, then there exists a state $s_1^B$ of $B$ such that $s^B \Rightarrow_B s_1^B$, and $s_1^A$ and $s_1^B$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.
Bisimulation between Transition Systems

States $s^A$ and $s^B$ of transition systems $A$ and $B$ are bisimilar if:

1. $s^A$ and $s^B$ are isomorphic;
2. If there exists a state $s_1^A$ of $A$ such that $s^A \xrightarrow{A} s_1^A$, then there exists a state $s_1^B$ of $B$ such that $s^B \xrightarrow{B} s_1^B$, and $s_1^A$ and $s_1^B$ are bisimilar;
3. The other direction!

$A$ and $B$ are bisimilar, if their initial states are bisimilar.

Theorem (bisimulation invariance)

Bisimilar transition systems satisfy the same $\mu\mathcal{L}$ formulas, where $\mu\mathcal{L}$ is a FO variant of $\mu$-calculus with quantification ranging on active domain only, and no quantification across states.
Two variants of $\mu \mathcal{L}$ are of particular interest: $\mu \mathcal{L}_A$ and $\mu \mathcal{L}_P$

**History-preserving $\mu$-calculus: $\mu \mathcal{L}_A$**

FO quantification across states ranges over current active domain only.

Examples:

$LTL_{FO}$: $\forall x. \text{Customer}(x) \supset F \text{Gold}(x)$

$\mu \mathcal{L}_A$: $\forall x. \text{Customer}(x) \supset \mu Z. \text{Gold}(x) \lor [\neg] Z$

**Persistence-preserving $\mu$-calculus: $\mu \mathcal{L}_P$**

FO quantification across states ranges over persisting individuals only.

Examples:

$LTL_{FO}$: $\forall x. \text{Gold}(x) \supset G \text{Gold}(x)$

$\mu \mathcal{L}_P$: $\forall x. \text{Gold}(x) \supset v Z. \text{Gold}(x) \land [\neg] Z$
History-Preserving and Persistence-Preserving Mu-Calculus

Two variants of $\mu\mathcal{L}$ are of particular interest: $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$

**History-preserving $\mu$-calculus: $\mu\mathcal{L}_A$**

FO quantification across states ranges over current active domain only.

Examples:

$LTL_{FO} : \forall x. \text{Customer}(x) \supset F \text{Gold}(x)$

$\mu\mathcal{L}_A : \forall x. \text{Customer}(x) \supset \mu Z.\text{Gold}(x) \lor [\neg]Z$

**Persistence-preserving $\mu$-calculus: $\mu\mathcal{L}_P$**

FO quantification across states ranges over persisting individuals only.

Examples:

$LTL_{FO} : \forall x. \text{Gold}(x) \supset G \text{Gold}(x)$

$\mu\mathcal{L}_P : \forall x. \text{Gold}(x) \supset \nu Z.\text{Gold}(x) \land [\neg]Z$
History-Preserving and Persistence-Preserving Mu-Calculus

Two variants of $\mu\mathcal{L}$ are of particular interest: $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$

### History-preserving $\mu$-calculus: $\mu\mathcal{L}_A$

FO quantification across states ranges over current active domain only.

Examples:

- $LTL_{FO}$: $\forall x. \text{Customer}(x) \supset F \text{Gold}(x)$
- $\mu\mathcal{L}_A$: $\forall x. \text{Customer}(x) \supset \mu Z. \text{Gold}(x) \vee [\neg]Z$

### Persistence-preserving $\mu$-calculus: $\mu\mathcal{L}_P$

FO quantification across states ranges over persisting individuals only.

Examples:

- $LTL_{FO}$: $\forall x. \text{Gold}(x) \supset G \text{Gold}(x)$
- $\mu\mathcal{L}_P$: $\forall x. \text{Gold}(x) \supset \nu Z. \text{Gold}(x) \land [\neg]Z$
Bisimulation between Transition Systems

History-preserving bisimilarity

**History-preserving bisimilarity** requires that isomorphism in the new states extends old ones.

Persistence-preserving bisimilarity

**Persistence-preserving bisimilarity** requires that isomorphism in the new states extends old ones only on objects in the active domain.

Theorem (bisimulation invariance [PODS13])

- **History-preserving bisimilar transition systems** satisfy the same $\mu\mathcal{L}_A$ (history-preserving) formulas.
- **Persistence-preserving bisimilar transition systems** satisfy the same $\mu\mathcal{L}_P$ (persistence-preserving) formulas.
Bisimulation between Transition Systems

### History-preserving bisimilarity

**History-preserving bisimilarity** requires that isomorphism in the new states **extends** old ones.

### Persistence-preserving bisimilarity

**Persistence-preserving bisimilarity** requires that isomorphism in the new states **extends** old ones only on objects in the active domain.

### Theorem (bisimulation invariance [PODS13])

- **History-preserving bisimilar transition systems** satisfy the same $\mu \mathcal{L}_A$ (history-preserving) formulas.
- **Persistence-preserving bisimilar transition systems** satisfy the same $\mu \mathcal{L}_P$ (persistence-preserving) formulas.
Bisimulation between Transition Systems

History-preserving bisimilarity

History-preserving bisimilarity requires that isomorphism in the new states extends old ones.

Persistence-preserving bisimilarity

Persistence-preserving bisimilarity requires that isomorphism in the new states extends old ones only on objects in the active domain.

Theorem (bisimulation invariance [PODS13])

- History-preserving bisimilar transition systems satisfy the same $\mu\mathcal{L}_A$ (history-preserving) formulas.
- Persistence-preserving bisimilar transition systems satisfy the same $\mu\mathcal{L}_P$ (persistence-preserving) formulas.
Two Key Semantical Conditions for Decidability

**Run-boundedness**

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \[\sim\] Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10,BPM11,DL11,ECAI11,JAIR13,PODS13]

**State-boundedness**

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \[\sim\] Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11,KR12,PODS13,RR13,DL13]
Two Key Semantical Conditions for Decidability

**Run-boundedness**

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

- Transition system accumulates finite information along a run.
- **But** accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
- ~ Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10, BPM11, DL11, ECAI11, JAIR13, PODS13]

**State-boundedness**

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
- ~ Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11, KR12, PODS13, RR13, DL13]
Two Key Semantical Conditions for Decidability

**Run-boundedness**

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \[ \Rightarrow \text{Easy to check syntactic conditions needed: Weak-acyclicity.} \]
- See [ACSI-ICSOC10, BPM11, DL11, ECAI11, JAIR13, PODS13]

**State-boundedness**

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \[ \Rightarrow \text{Easy to check syntactic conditions needed: GR-acyclicity.} \]
- See [ACSI-ICSOC11, KR12, PODS13, RR13, DL13]
Two Key Semantical Conditions for Decidability

### Run-boundedness

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \( \sim \) Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10,BPM11,DL11,ECAI11,JAIR13,PODS13]

### State-boundedness

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \( \sim \) Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11,KR12,PODS13,RR13,DL13]
Two Key Semantical Conditions for Decidability

Run-boundedness

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \[\rightsquigarrow\] Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10,BPM11,DL11,ECAI11,JAIR13,PODS13]

State-boundedness

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \[\rightsquigarrow\] Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11,RR13,PODS13,RR13,DL13]
Two Key Semantical Conditions for Decidability

Run-boundedness

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \(\sim\) Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10,BPM11,DL11,ECAI11,JAIR13,PODS13]

State-boundedness

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \(\sim\) Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11,KR12,PODS13,RR13,DL13]
Two Key Semantical Conditions for Decidability

### Run-boundedness

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \(\sim\) Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10,BPM11,DL11,ECAI11,JAIR13,PODS13]

### State-boundedness

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \(\sim\) Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11,KR12,PODS13,RR13,DL13]
Two Key Semantical Conditions for Decidability

Run-boundedness

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \(\sim\) Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10,BPM11,DL11,ECAI11,JAIR13,PODS13]

State-boundedness

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \(\sim\) Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11,KR12,PODS13,RR13,DL13]
Two Key Semantical Conditions for Decidability

Run-boundedness

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

- Transition system accumulates finite information along a run.
- But accumulates infinite information through infinite branching.
- This is a semantic condition, whose checking is undecidable.
  \[ \sim \] Easy to check syntactic conditions needed: Weak-acyclicity.
- See [ACSI-ICSOC10, BPM11, DL11, ECAI11, JAIR13, PODS13]

State-boundedness

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

- Relaxation of run-boundedness.
- Infinite accumulation of information along each run is possible.
- This is a semantic condition, whose checking is undecidable.
  \[ \sim \] Easy to check syntactic conditions needed: GR-acyclicity.
- See [ACSI-ICSOC11, KR12, PODS13, RR13, DL13]
Decidability Results for Run-Bounded Transition Systems:

Theorem

Verification of $\mu L_A$ over run-bounded transition systems is decidable and can be reduced to model checking of propositional $\mu$-calculus over a finite transition system.

Idea: use isomorphic types instead of actual values.

Remember: runs are bounded!
Decidability Results for Run-Bounded Transition Systems:

Theorem

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

Idea: use isomorphic types instead of actual values.

Remember: runs are bounded!
Decidability Results for State-Bounded Transition System

**Theorem**

Verification of $\mu L_P$ over state-bounded transition systems is decidable and can be reduced to model checking of propositional $\mu$-calculus over a finite transition system.

**Steps:**

1. **Prune** infinite branching (isomorphic types).
2. Finite abstraction along the runs:
   - $\mu L_P$ looses track of previous values that do not exist anymore.
   - New values can be replaced with old, non-persisting ones.
   - This eventually leads to **recycle** the old values without generating new ones.
Decidability Results for State-Bounded Transition System

**Theorem**

Verification of $\mu\mathcal{L}_P$ over **state-bounded** transition systems is decidable and can be reduced to model checking of propositional $\mu$-calculus over a finite transition system.

Steps:

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

Theorem

Verification of $\mu L_P$ over state-bounded transition systems is decidable and can be reduced to model checking of propositional $\mu$-calculus over a finite transition system.

Steps:

1. **Prune** infinite branching (isomorphic types).
2. Finite abstraction along the runs:
   - $\mu L_P$ looses track of previous values that do not exist anymore.
   - New values can be replaced with old, non-persisting ones.
   - This eventually leads to recycle the old values without generating new ones.
Decidability Results for State-Bounded Transition System

Theorem

Verification of $\mu L_P$ over state-bounded transition systems is decidable and can be reduced to model checking of propositional $\mu$-calculus over a finite transition system.

Steps:

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

1 Data and Processes
2 Artifact-Centric Approach
3 Data-Centric Dynamic Systems
4 Situation Calculus
5 Description Logics
6 SitCalc and DLs
7 Levesque’s Functional Approach to KBs
8 Knowledge and Action Bases
9 Conclusion
Situation Calculus

SitCalc [McCarthy63], [McCarthyHayes69], [Reiter01] Very well-developed formalism for reasoning about actions.

SitCalc (Reiter’s version)

- First-Order multi sorted language - but over inductively defined situations (i.e. situations are defined in Second-Order Logic). Sorts:
  - Objects - standard names, in our case
  - Situations: denote the current state and the history that lead to that state.
  - Actions: progress the systems - we assume finite action types (but with infinite many possible object parameters)
  - Fluents: assert property of objects in situations - only predicates in our case

- Precondition axioms: define when (in which situations) actions (with parameters) can be executed
  \[
  Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s)
  \]

- Successor state axioms: define the effects of action execution - include solution to the frame problem
  \[
  F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s)
  \]

- Initial situation \( S_0 \) description: description of the initial state of the system
Situation Calculus

SitCalc \cite{McCarthy63},\cite{McCarthyHayes69},\cite{Reiter01} Very well-developed formalism for reasoning about actions.

SitCalc (Reiter’s version)

- **First-Order** multi sorted language - but over inductively defined situations (i.e. situations are defined in **Second-Order** Logic). Sorts:
  - Objects - standard names, in our case
  - Situations: denote the current state and the history that lead to that state.
  - Actions: progress the systems - we assume finite action types (but with infinite many possible object parameters)
  - Fluents: assert property of objects in situations - only predicates in our case

- **Precondition axioms:** define when (in which situations) actions (with parameters) can be executed
  \[
  Pos\!(a(\vec{x}), s) \equiv \Phi(\vec{x}, s)\]

- **Successor state axioms:** define the effects of action execution - include solution to the frame problem
  \[
  F(\vec{x}, do(a, s)) \equiv \Phi^+_F(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi^-_F(\vec{x}, a, s)\]

- **Initial situation** $S_0$ description: description of the initial state of the system
Situation Calculus

SitCalc [McCarthy63], [McCarthyHayes69], [Reiter01] Very well-developed formalism for reasoning about actions.

SitCalc (Reiter’s version)

- **First-Order** multi sorted language - but over inductively defined situations (i.e. situations are defined in **Second-Order** Logic). Sorts:
  - **Objects**: standard names, in our case
  - **Situations**: denote the current state and the history that lead to that state.
  - **Actions**: progress the systems - we assume finite action types (but with infinite many possible object parameters)
  - **Fluents**: assert property of objects in situations - only predicates in our case

- **Precondition axioms**: define when (in which situations) actions (with parameters) can be executed
  \[ \text{Poss}(a(\vec{x}), s) \equiv \Phi(\vec{x}, s) \]

- **Successor state axioms**: define the effects of action execution - include solution to the frame problem
  \[ F(\vec{x}, \text{do}(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \]

- **Initial situation** \( S_0 \) description: description of the initial state of the system
Situation Calculus

SitCalc [McCarthy63],[McCarthyHayes69],[Reiter01] Very well-developed formalism for reasoning about actions.

**SitCalc (Reiter’s version)**

- **First-Order** multi sorted language - but over inductively defined situations (i.e. situations are defined in **Second-Order** Logic). Sorts:
  - Objects -standard names, in our case
  - Situations: denote the current state and the history that lead to that state.
  - Actions: progress the systems - we assume finite action types (but with infinite many possible object parameters)
  - Fluents: assert property of objects in situations - only predicates in our case

- **Precondition axioms**: define when (in which situations) actions (with parameters) can be executed
  \[\text{Poss}(a(\vec{x}), s) \equiv \Phi(\vec{x}, s)\]

- **Successor state axioms**: define the effects of action execution - include solution to the frame problem
  \[F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s)\]

- **Initial situation** $S_0$ description: description of the initial state of the system
Situation Calculus

SitCalc [McCarthy63],[McCarthyHayes69],[Reiter01] Very well-developed formalism for reasoning about actions.

SitCalc (Reiter’s version)

- **First-Order** multi sorted language - but over inductively defined situations (i.e. situations are defined in **Second-Order** Logic). Sorts:
  - **Objects** -standard names, in our case
  - **Situations**: denote the current state and the history that lead to that state.
  - **Actions**: progress the systems - we assume finite action types (but with infinite many possible object parameters)
  - **Fluents**: assert property of objects in situations - only predicates in our case

- **Precondition axioms**: define when (in which situations) actions (with parameters) can be executed
  \[ Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s) \]

- **Successor state axioms**: define the effects of action execution - include solution to the frame problem
  \[ F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \]

- **Initial situation** $S_0$ **description**: description of the initial state of the system
SitCalc: Reasoning

Key feature: Regression

Regression allows for reducing reasoning about a given future situation to reasoning about the initial situation — greatly simplifies reasoning as progression/executability.

However, more sophisticated temporal properties are also of interest:

- There exists a future situation such that $\alpha$
- For all (future) situations $\alpha$ holds
- Eventually whatever actions we do we have $\alpha$
- Always when $\alpha$ then eventually $\beta$
- ...

When we deal with such properties virtually all decidability results are based on assuming a finite number of states:

- Propositional SitCalc
- Assume finite object domain
- Exception: incomplete fixpoint approximation-based methods
SitCalc: Reasoning

Key feature: Regression

Regression allows for reducing reasoning about a given future situation to reasoning about the initial situation — greatly simplifies reasoning as progression/executability.

However, more sophisticated temporal properties are also of interest:

- There exists a future situation such that $\alpha$
- For all (future) situations $\alpha$ holds
- Eventually whatever actions we do we have $\alpha$
- Always when $\alpha$ then eventually $\beta$
- ...

When we deal with such properties virtually all decidability results are based on assuming a finite number of states:

- Propositional SitCalc
- Assume finite object domain
- Exception: incomplete fixpoint approximation-based methods
SitCalc: Reasoning

Key feature: Regression

Regression allows for reducing reasoning about a given future situation to reasoning about the initial situation — greatly simplifies reasoning as progression/executability.

However, more sophisticated temporal properties are also of interest:

- There exists a future situation such that $\alpha$
- For all (future) situations $\alpha$ holds
- Eventually whatever actions we do we have $\alpha$
- Always when $\alpha$ then eventually $\beta$
- ...

When we deal with such properties virtually all decidability results are based on assuming a finite number of states:

- Propositional SitCalc
- Assume finite object domain
- Exception: incomplete fixpoint approximation-based methods
Bounded Action Theories

**Key Observation:**

FOL can express being **bounded** by $b$:

$$\#\{\vec{x} \mid \phi(\vec{x})\} < b \equiv \neg (\exists \vec{x}_1, \ldots, \vec{x}_b. \phi(\vec{x}_1) \land \cdots \land \phi(\vec{x}_b) \land \bigwedge_{i,j \in \{1, \ldots, b\}, i \neq j} \vec{x}_i \neq \vec{x}_j)$$

- Fluent $F(\vec{x}, s)$ in situation $s$ is **bounded** by $b$:
  $$Bounded_{F,b}(s) \equiv \#\{\vec{x} \mid F(\vec{x}, s)\} < b.$$

- Situation $s$ is bounded by $b$:
  $$Bounded_b(s) \equiv \bigwedge_{F \in \mathcal{F}} Bounded_{F,b}(s).$$

- Action theory $\mathcal{D}$ is **bounded** by $b$:
  $$\mathcal{D} \models \forall s. Executable(s) \supset Bounded_b(s).$$

**Main result**

Verification of temporal properties on bounded action theories is decidable.
Bounded Action Theories

Key Observation:

FOL can express being bounded by $b$:

$$\#\{\vec{x} \mid \phi(\vec{x})\} < b \equiv \neg(\exists \vec{x}_1, \ldots, \vec{x}_b. \phi(\vec{x}_1) \land \cdots \land \phi(\vec{x}_b) \land \bigwedge_{i,j\in\{1,\ldots,b\}, i\neq j} \vec{x}_i \neq \vec{x}_j)$$

- Fluent $F(\vec{x}, s)$ in situation $s$ is bounded by $b$:

$$Bounded_{F,b}(s) \equiv \#\{\vec{x} \mid F(\vec{x}, s)\} < b.$$ 

- Situation $s$ is bounded by $b$:

$$Bounded_b(s) \equiv \bigwedge_{F\in\mathcal{F}} Bounded_{F,b}(s).$$

- Action theory $\mathcal{D}$ is bounded by $b$:

$$\mathcal{D} \models \forall s. Executable(s) \supset Bounded_b(s).$$

Main result

Verification of temporal properties on bounded action theories is decidable.
**Bounded Action Theories**

**Key Observation:**

FOL can express being bounded by $b$:

$$\#\{\vec{x} \mid \phi(\vec{x})\} < b \equiv \neg(\exists \vec{x}_1, \ldots, \vec{x}_b. \phi(\vec{x}_1) \land \cdots \land \phi(\vec{x}_b) \land \bigwedge_{i,j \in \{1,\ldots,b\}, i \neq j} \vec{x}_i \neq \vec{x}_j)$$

- Fluent $F(\vec{x}, s)$ in situation $s$ is **bounded** by $b$:
  $$\text{Bounded}_{F,b}(s) \equiv \#\{\vec{x} \mid F(\vec{x}, s)\} < b.$$

- Situation $s$ is bounded by $b$:
  $$\text{Bounded}_b(s) \equiv \bigwedge_{F \in \mathcal{F}} \text{Bounded}_{F,b}(s).$$

- Action theory $\mathcal{D}$ is **bounded** by $b$:
  $$\mathcal{D} \models \forall s. \text{Executable}(s) \supset \text{Bounded}_b(s).$$

**Main result**

Verification of temporal properties on bounded action theories is decidable.
Obtaining Bounded Action Theories by Blocking Execution

Observe:

\[ Bounded_b(s) \] is an FOL formula uniform in \( s \) and hence regressable.

Bounded Action Theories by Blocking Execution

From any basic action theory, we obtain a bounded action theory by replacing

\[
Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s)
\]

with

\[
Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s) \land R[Bounded_b(do(a(\vec{x}), s)]
\]

Theorem

Let \( \mathcal{D} \) be a basic action theory with the initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models Bounded_b(S_0) \), for some \( b \), and let \( \mathcal{D}_b \) be the basic action theory obtained as discussed above. Then, \( \mathcal{D}_b \) is bounded by \( b \).
Obtaining Bounded Action Theories by Blocking Execution

Observe:

\( Bounded_b(s) \) is an FOL formula uniform in \( s \) and hence regressable.

Bounded Action Theories by Blocking Execution

From any basic action theory, we obtain a bounded action theory by replacing

\[
Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s)
\]

with

\[
Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s) \land R[Bounded_b(do(a(\vec{x}), s)]
\]

Theorem

Let \( \mathcal{D} \) be a basic action theory with the initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models Bounded_b(S_0) \), for some \( b \), and let \( \mathcal{D}_b \) be the basic action theory obtained as discussed above. Then, \( \mathcal{D}_b \) is bounded by \( b \).
Observe:

\( Bounded_b(s) \) is an FOL formula uniform in \( s \) and hence regressable.

Bounded Action Theories by Blocking Execution

From any basic action theory, we obtain a bounded action theory by replacing

\[
Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s)
\]

with

\[
Poss(a(\vec{x}), s) \equiv \Phi(\vec{x}, s) \land R[Bounded_b(do(a(\vec{x}), s)]
\]

Theorem

Let \( \mathcal{D} \) be a basic action theory with the initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models Bounded_b(S_0) \), for some \( b \), and let \( \mathcal{D}_b \) be the basic action theory obtained as discussed above. Then, \( \mathcal{D}_b \) is bounded by \( b \).
Effect Bounded Action Theories

Recall successor state axioms:

\[ F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \]

Effect bounded

Fluent \( F \) is **effect bounded** if for every action and situation, the number of tuples added to the fluent is less than or equal to that deleted:

\[ \#\{\vec{x} \mid \Phi_F^+(\vec{x}, a, s) \land \neg F(\vec{x}, s)\} \leq \#\{\vec{x} \mid F(\vec{x}, s) \land \Phi_F^-(\vec{x}, a, s)\} \]

Basic action theory is effect bounded if every fluent \( F \in \mathcal{F} \) is effect bounded.

**Theorem**

Let \( \mathcal{D} \) be an effect bounded basic action theory with the initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models Bounded_b(S_0) \), for some \( b \). Then \( \mathcal{D} \) is bounded by \( b \).

Observe: effect boundedness cannot be expressed as a FOL formula!
**Effect Bounded Action Theories**

Recall successor state axioms:

\[
F(\vec{x}, \text{do}(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s)
\]

---

**Effect bounded**

Fluent \( F \) is **effect bounded** if for every action and situation, the number of tuples added to the fluent is less than or equal to that deleted:

\[
\#\{\vec{x} \mid \Phi_F^+(\vec{x}, a, s) \land \neg F(\vec{x}, s)\} \leq \#\{\vec{x} \mid F(\vec{x}, s) \land \Phi_F^-(\vec{x}, a, s)\}
\]

Basic action theory is effect bounded if every fluent \( F \in \mathcal{F} \) is effect bounded.

---

**Theorem**

Let \( \mathcal{D} \) be an effect bounded basic action theory with the initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models \text{Bounded}_b(S_0) \), for some \( b \). Then \( \mathcal{D} \) is bounded by \( b \).

Observe: effect boundedness cannot be expressed as a FOL formula!
Effect Bounded Action Theories

Recall successor state axioms:

\[ F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \]

Effect bounded

Fluent \( F \) is **effect bounded** if for every action and situation, the number of tuples added to the fluent is less than or equal to that deleted:

\[ \#\{\vec{x} \mid \Phi_F^+(\vec{x}, a, s) \land \neg F(\vec{x}, s)\} \leq \#\{\vec{x} \mid F(\vec{x}, s) \land \Phi_F^-(\vec{x}, a, s)\} \]

Basic action theory is effect bounded if every fluent \( F \in \mathcal{F} \) is effect bounded.

Theorem

*Let \( \mathcal{D} \) be an effect bounded basic action theory with the initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models \text{Bounded}_b(S_0) \), for some \( b \). Then \( \mathcal{D} \) is bounded by \( b \).*

Observe: effect boundedness cannot be expressed as a FOL formula!
Fading Fluents Action Theories

Idea: information over time fades away unless explicitly reinforced!

Recall: normally fluents have successor state axioms of the form:

\[
F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s)
\]

Fading Fluents Action Theories

A fading fluent \( F(\vec{x}, s) \) \( \equiv \bigvee_{0 \leq i \leq \ell} F_i(\vec{x}, s) \), where the auxiliary fluents \( F_i \) have successor state axioms of a special form:

\[
\begin{align*}
F_\ell(\vec{x}, do(a, s)) & \equiv \Phi_F^+(\vec{x}, a, s) \land \#\{\vec{x} \mid \exists a. \Phi_F^+(\vec{x}, a, s)\} < b \\
F_i(\vec{x}, do(a, s)) & \equiv F_{i+1}(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \land \neg \Phi_F^+(\vec{x}, a, s) \quad (0 \leq i < \ell)
\end{align*}
\]

Rationale: tuples are initially added to \( F_\ell \), and progressively lose their strength, moving from \( F_i \) to \( F_{i-1} \), until eventually they move out of \( F_0 \) and forgotten, unless an action re-add (or delete) them during the fading.

Theorem

Let \( \mathcal{D} \) be a fading fluents action theory with fading length \( \ell \) and initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models \text{Bounded}_b(S_0) \), for some \( b \). Then, \( \mathcal{D} \) is bounded by \( b \).
Fading Fluents Action Theories

Idea: information over time fades away unless explicitly reinforced!

Recall: normally fluents have successor state axioms of the form:

\[ F(\vec{x}, do(a, s)) \equiv \Phi^+_F(\vec{x}, a, s) \lor F(\vec{x}, s) \lor \neg \Phi^-_F(\vec{x}, a, s) \]

Fading Fluents Action Theories

A fading fluent \( F(\vec{x}, s) \) is defined as:

\[ F(\vec{x}, s) \triangleq \bigvee_{0 \leq i \leq \ell} F_i(\vec{x}, s) \]

where the auxiliary fluents \( F_i \) have successor state axioms of a special form:

\[
\begin{align*}
F_\ell(\vec{x}, do(a, s)) & \equiv \Phi^+_F(\vec{x}, a, s) \land \#\{\vec{x} \mid \exists a. \Phi^+_F(\vec{x}, a, s)\} < b \\
F_i(\vec{x}, do(a, s)) & \equiv F_{i+1}(\vec{x}, s) \land \neg \Phi^-_F(\vec{x}, a, s) \land \neg \Phi^+_F(\vec{x}, a, s) & (0 \leq i < \ell)
\end{align*}
\]

Rationale: tuples are initially added to \( F_\ell \), and progressively lose their strength, moving from \( F_i \) to \( F_{i-1} \), until eventually they move out of \( F_0 \) and forgotten, unless an action re-add (or delete) them during the fading.

Theorem

Let \( \mathcal{D} \) be a fading fluents action theory with fading length \( \ell \) and initial database \( D_0 \) such that \( D_0 \models Bounded_b(S_0) \), for some \( b \). Then, \( \mathcal{D} \) is bounded by \( b \).
Fading Fluents Action Theories

Idea: information over time fades away unless explicitly reinforced!

Recall: normally fluents have successor state axioms of the form:

\[ F(\vec{x}, do(a, s)) \equiv \Phi^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg\Phi^-(\vec{x}, a, s) \]

Fading Fluents Action Theories

A fading fluent \( F(\vec{x}, s) \equiv \bigvee_{0 \leq i \leq \ell} F_i(\vec{x}, s) \), where the auxiliary fluents \( F_i \) have successor state axioms of a special form:

\[
F_\ell(\vec{x}, do(a, s)) \equiv \Phi^+_F(\vec{x}, a, s) \land \#\{\vec{x} | \exists a.\Phi^+_F(\vec{x}, a, s)\} < b \\
F_i(\vec{x}, do(a, s)) \equiv F_{i+1}(\vec{x}, s) \land \neg\Phi^-_F(\vec{x}, a, s) \land \neg\Phi^+_F(\vec{x}, a, s) \quad (0 \leq i < \ell)
\]

Rationale: tuples are initially added to \( F_\ell \), and progressively lose their strength, moving from \( F_i \) to \( F_{i-1} \), until eventually they move out of \( F_0 \) and forgotten, unless an action re-add (or delete) them during the fading.

Theorem

Let \( D \) be a fading fluents action theory with fading length \( \ell \) and initial database \( D_0 \) such that \( D_0 \models Bounded_b(S_0) \), for some \( b \). Then, \( D \) is bounded by \( b \).
Fading Fluents Action Theories

Idea: information over time fades away unless explicitly reinforced!

Recall: normally fluents have successor state axioms of the form:

\[ F(\vec{x}, do(a, s)) \equiv \Phi_F^+(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \]

Fading Fluents Action Theories

A fading fluent \( F(\vec{x}, s) \equiv \bigvee_{0 \leq i \leq \ell} F_i(\vec{x}, s) \), where the auxiliary fluents \( F_i \) have successor state axioms of a special form:

\[
\begin{align*}
F_{\ell}(\vec{x}, do(a, s)) & \equiv \Phi_F^+(\vec{x}, a, s) \land \#\{\vec{x} \mid \exists a.\Phi_F^+(\vec{x}, a, s)\} < b \\
F_i(\vec{x}, do(a, s)) & \equiv F_{i+1}(\vec{x}, s) \land \neg \Phi_F^-(\vec{x}, a, s) \land \neg \Phi_F^+(\vec{x}, a, s) \quad (0 \leq i < \ell)
\end{align*}
\]

Rationale: tuples are initially added to \( F_\ell \), and progressively lose their strength, moving from \( F_i \) to \( F_{i-1} \), until eventually they move out of \( F_0 \) and forgotten, unless an action re-add (or delete) them during the fading.

Theorem

Let \( \mathcal{D} \) be a fading fluents action theory with fading length \( \ell \) and initial database \( \mathcal{D}_0 \) such that \( \mathcal{D}_0 \models \text{Bounded}_b(S_0) \), for some \( b \). Then, \( \mathcal{D} \) is bounded by \( b \).
Example

Example ("Vacuum cleaner world")

- A robotic vacuum cleaner may clean a room or region \( r \). If a room/region is used, then it becomes unclean.

- We could model this using a fluent \( IsClean(r, s) \) with the following successor state axiom:

\[
IsClean(r, do(a, s)) \equiv a = clean(r) \lor IsClean(r, s) \land a \neq use(r)
\]

- Clearly, cleanliness is a property that fades over time. By applying the proposed transformation to this specification, we obtain the following:

\[
IsClean_\ell(r, do(a, s)) \equiv a = clean(r) \quad (we \ assume \ b \geq 1)
\]

\[
IsClean_i(r, do(a, s)) \equiv IsClean_{i+1}(r, s) \land a \neq use(r) \land a \neq clean(r) \quad (0 \leq i < \ell)
\]

NB: after \( \ell \) steps robot forgets about a room being clean.
Example

Example ("Vacuum cleaner world")

- A robotic vacuum cleaner may clean a room or region \( r \). If a room/region is used, then it becomes unclean.
- We could model this using a fluent \( \text{IsClean}(r, s) \) with the following successor state axiom:

\[
\text{IsClean}(r, \text{do}(a, s)) \equiv a = \text{clean}(r) \lor \text{IsClean}(r, s) \land a \neq \text{use}(r)
\]

- Clearly, cleanliness is a property that fades over time. By applying the proposed transformation to this specification, we obtain the following:

\[
\text{IsClean}_\ell(r, \text{do}(a, s)) \equiv a = \text{clean}(r) \quad (\text{we assume } b \geq 1)
\]

\[
\text{IsClean}_i(r, \text{do}(a, s)) \equiv \text{IsClean}_{i+1}(r, s) \land a \neq \text{use}(r) \land a \neq \text{clean}(r)
\]

\((0 \leq i < \ell)\)

NB: after \( \ell \) steps robot forgets about a room being clean.
Example

Example ("Vacuum cleaner world")

• A robotic vacuum cleaner may clean a room or region $r$. If a room/region is used, then it becomes unclean.

• We could model this using a fluent $IsClean(r, s)$ with the following successor state axiom:

$$IsClean(r, do(a, s)) \equiv a = clean(r) \lor IsClean(r, s) \land a \neq use(r)$$

• Clearly, cleanliness is a property that fades over time. By applying the proposed transformation to this specification, we obtain the following:

$$IsClean_\ell(r, do(a, s)) \equiv a = clean(r) \quad (we \ assume \ b \geq 1)$$

$$IsClean_i(r, do(a, s)) \equiv IsClean_{i+1}(r, s) \land a \neq use(r) \land a \neq clean(r) \quad (0 \leq i < \ell)$$

NB: after $\ell$ steps robot forgets about a room being clean.
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu\mathcal{L}_P$ as discussed before
- Combinining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- ...

Giuseppe De Giacomo (Sapienza)  Verification of Data-Aware Processes  WS-FM:FASOCC 2014 41 / 81
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu\mathcal{L}_P$ as discussed before
- Combinining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- …
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu\mathcal{L}_P$ as discussed before
- Combinining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- ...
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for CTL_A [PODS13], for μLP as discussed before
- Combinining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- ...

Giuseppe De Giacomo (Sapienza)  Verification of Data-Aware Processes  WS-FM:FASOCCE 2014
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu\mathcal{L}_P$ as discussed before
- Combining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- ...
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations
cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu\mathcal{L}_P$ as discussed before
- Combinining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- ...
Discussion on Bounded SitCalc

Summary
Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu\mathcal{L}_P$ as discussed before
- Combining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
Discussion on Bounded SitCalc

Summary

Boundness action theories are a truly non propositional case of SitCalc where verification is decidable!

Extension

- Boundeness itself is verifiable (for SitCalc Action Theories with bounded initial state) [AAMAS14]
- Bounded theories are always progressable(!) [AAMAS14]
- Boundeness can be applied to online (vs offline) executions as well [AAMAS14,ECAI14]
- Verification formalism with object quantification across situations cf. [Belardinelli,Lomuscio,PatriziKR12] for $CTL_A$ [PODS13], for $\mu L_P$ as discussed before
- Combinining with knowledge [IJCAI13,AAMAS14,ECAI14].
- Golog/ConGolog/IndiGolog programs running over bounded action theories give rise to bounded state transition systems! [To appear]
- ...

Giuseppe De Giacomo (Sapienza)
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Description Logics

In modeling an application domain we typically need to represent the domain of interest in terms of:

- objects
- classes
- relations (or associations)

and to reason about the representation

Description Logics (DLs) are logics specifically designed to represent and reason on:

- objects
- classes – called “concepts” in DLs
- (binary) relations – called “roles” in DLs
Brief history of DLs

DLs are one of the main formalisms developed in AI Knowledge Representation [DLBOOK03].

- [late '70s, early '80s] – early days of KR formalisms
  - Semantic Networks: graph-based formalism, used to represent the meaning of sentences
  - Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

Problems: no clear semantics, reasoning not well understood

- [mid '80s, '90s] – Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid '80s, with the aim of providing semantics and inference techniques to knowledge representation systems

- [Today] DLs are at the base of the whole research on ontology, and formalization of data conceptual modeling.
Brief history of DLs

DLs are one of the main formalisms developed in AI Knowledge Representation [DLBOOK03].

- [late ’70s, early ’80s] – early days of KR formalisms
  - Semantic Networks: graph-based formalism, used to represent the meaning of sentences
  - Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

Problems: no clear semantics, reasoning not well understood

- [mid ’80s, ’90s] – Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid ’80s, with the aim of providing semantics and inference techniques to knowledge representation systems

- [Today] DLs are at the base of the whole research on ontology, and formalization of data conceptual modeling.
Brief history of DLs

DLs are one of the main formalisms developed in AI Knowledge Representation [DLBOOK03].

- **[late '70s, early '80s] – early days of KR formalisms**
  - Semantic Networks: graph-based formalism, used to represent the meaning of sentences
  - Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

Problems: no clear semantics, reasoning not well understood

- **[mid '80s, '90s] – Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid '80s, with the aim of providing semantics and inference techniques to knowledge representation systems**

- **[Today] DLs are at the base of the whole research on ontology, and formalization of data conceptual modeling.**
Brief history of DLs

DLs are one of the main formalisms developed in AI Knowledge Representation (DLBOOK03).

- **[late ’70s, early ’80s]** – early days of KR formalisms
  - Semantic Networks: graph-based formalism, used to represent the meaning of sentences
  - Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

Problems: no clear semantics, reasoning not well understood

- **[mid ’80s, ’90s]** – Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid ’80s, with the aim of providing semantics and inference techniques to knowledge representation systems

- **[Today]** DLs are at the base of the whole research on ontology, and formalization of data conceptual modeling.
Brief history of DLs

DLs are one of the main formalisms developed in AI Knowledge Representation [DLBOOK03].

- [late '70s, early '80s] – early days of KR formalisms
  - Semantic Networks: graph-based formalism, used to represent the meaning of sentences
  - Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

  Problems: no clear semantics, reasoning not well understood

- [mid '80s, '90s] – Description Logics (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid '80s, with the aim of providing semantics and inference techniques to knowledge representation systems

- [Today] DLs are at the base of the whole research on ontology, and formalization of data conceptual modeling.
Current applications of DLs

DLs have evolved from being used “just” in KR

Found applications in:

• Databases:
  ▶ schema design, schema evolution
  ▶ query optimization
  ▶ integration of heterogeneous data sources, data warehousing
• Conceptual modeling
• Foundation for the semantic web
• Ontology-Based Data Access (OBDA)
• ...
OBDA and *DL-Lite*

**OBDA**

Ontology-Based Data Access is the problem of accessing and querying external data sources through an ontology.

**DL-Lite family**

- A family of DLs optimized according to the tradeoff between expressive power and complexity of query answering, with emphasis on data [CDeGLLRAAAI05], [CDeGLLRAIJ13].

- Carefully design to capture most UML/ER/Ontology constructs

- Carefully designed to have nice computational properties for answering conjunctive queries (CQs, and UCQs):
  - The same complexity as relational databases.
  - In fact, query answering can be delegated to a relational DB engine.
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties.
Capturing basic ontology constructs in $DL-Lite_A$

<table>
<thead>
<tr>
<th>ISA between classes</th>
<th>$A_1 \sqsubseteq A_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disjointness between classes</td>
<td>$A_1 \sqsubseteq \neg A_2$</td>
</tr>
<tr>
<td>Domain and range of properties</td>
<td>$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$</td>
</tr>
<tr>
<td>Mandatory participation ($min\ card = 1$)</td>
<td>$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$</td>
</tr>
<tr>
<td>Functionality of relations ($max\ card = 1$)</td>
<td>$(\text{funct } P) \quad (\text{funct } P^-)$</td>
</tr>
<tr>
<td>ISA between properties</td>
<td>$Q_1 \sqsubseteq Q_2$</td>
</tr>
<tr>
<td>Disjointness between properties</td>
<td>$Q_1 \sqsubseteq \neg Q_2$</td>
</tr>
</tbody>
</table>

Note without loosing its nice computational features:

- $DL-Lite_A$ cannot capture completeness of a hierarchy. This would require disjunction (i.e., reasoning by cases).
- $DL-Lite_A$ cannot capture subset constraints on association with max multiplicity different from “*”. This may again introduce an hidden form of disjunction (i.e., reasoning by cases).
- $DL-Lite_A$ can be extended to capture also min cardinality constraints “$A \sqsubseteq \leq nQ$” and max cardinality constraints “$A \sqsubseteq \geq nQ$”.
- $DL-Lite_A$ can be extended to capture also identification constraints “$(\text{id } C \ Q_1, \ldots, Q_n)$.”
Capturing basic ontology constructs in $\textit{DL-Lite}_A$

<table>
<thead>
<tr>
<th>Construct</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA between classes</td>
<td>$A_1 \sqsubseteq A_2$</td>
</tr>
<tr>
<td>Disjointness between classes</td>
<td>$A_1 \sqsubseteq \neg A_2$</td>
</tr>
<tr>
<td>Domain and range of properties</td>
<td>$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$</td>
</tr>
<tr>
<td>Mandatory participation ($\textit{min card} = 1$)</td>
<td>$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$</td>
</tr>
<tr>
<td>Functionality of relations ($\textit{max card} = 1$)</td>
<td>(funct $P$) (funct $P^-$)</td>
</tr>
<tr>
<td>ISA between properties</td>
<td>$Q_1 \sqsubseteq Q_2$</td>
</tr>
<tr>
<td>Disjointness between properties</td>
<td>$Q_1 \sqsubseteq \neg Q_2$</td>
</tr>
</tbody>
</table>

Note without losing its nice computational features:

- $\textit{DL-Lite}_A$ cannot capture \textbf{completeness} of a hierarchy. This would require \textbf{disjunction} (i.e., reasoning by cases).
- $\textit{DL-Lite}_A$ cannot capture \textbf{subset constraints on association} with max multiplicity different from “*”. This may again introduce an hidden form of \textbf{disjunction} (i.e., reasoning by cases).
- $\textit{DL-Lite}_A$ can be extended to capture also \textbf{min cardinality constraints} “$A \sqsubseteq \leq nQ$” and \textbf{max cardinality constraints} “$A \sqsubseteq \geq nQ$”.
- $\textit{DL-Lite}_A$ can be extended to capture also \textbf{identification constraints} “$(\text{id} C Q_1, \ldots, Q_n)$”.
Translating UML Class Diagrams in $DL$-$Lite_A$ KBs: 
example

```
Professor ⊑ Faculty
AssocProf ⊑ Professor
Dean ⊑ Professor
AssocProf ⊑ ¬Dean
Faculty ⊑ ∃age
∃age⁻ ⊑ xsd:integer
(funct age)
∃worksFor ⊑ Faculty
∃worksFor⁻ ⊑ College
Faculty ⊑ ∃worksFor
College ⊑ ∃worksFor⁻
∃isHeadOf ⊑ Dean
∃isHeadOf⁻ ⊑ College
Dean ⊑ ∃isHeadOf
College ⊑ ∃isHeadOf⁻
isHeadOf ⊑ worksFor
(funct isHeadOf)
(funct isHeadOf⁻)
...
Observations on $DL$-$Lite_A$

- Captures all the basic constructs of UML Class Diagrams and of the ER Model …
- … except covering constraints in generalizations.
- Is the logical underpinning of OWL2 QL, one of the OWL 2 Profiles.
- Extends (the DL fragment of) the ontology language RDFS.
- Is completely symmetric w.r.t. direct and inverse properties.
Observations on $DL-Lite_A$

- Captures all the basic constructs of UML Class Diagrams and of the ER Model . . .
- . . . except covering constraints in generalizations.
- Is the logical underpinning of OWL2 QL, one of the OWL 2 Profiles.
- Extends (the DL fragment of) the ontology language RDFS.
- Is completely symmetric w.r.t. direct and inverse properties.
Data complexity of query answering in ontologies

QA has been studied extensively for (unions of) CQs in the context of Description Logic-based ontology languages, which can be though of as specific FOL formalisms for class-based representation (cf. UML class diagrams or ER):

<table>
<thead>
<tr>
<th>CQ query answering</th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete databases</td>
<td>NP-complete</td>
<td>in AC$^0$(1)</td>
</tr>
<tr>
<td>Incomplete databases (naive tables, OWA) no TBox</td>
<td>NP-complete</td>
<td>in AC$^0$</td>
</tr>
<tr>
<td>UML Class Diagrams or OWL2* TBoxes</td>
<td>ExpTime [2]-hard</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

* OWL 2 is a W3C standard based on Description Logics (DLs).

(1) AC$^0$ ⊆ LOGSPACE is the cost of evaluating FOL over relational DBs – This is what we need to scale with the data.

The key question at the base of development of DL-Lite

How do we find interesting logics for expressing ontologies for which the query answering

- can be solved in AC$^0$, and
- we leverage on evaluation and relational database technology for query answering?
Data complexity of query answering in ontologies

QA has been studied extensively for (unions of) CQs in the context of **Description Logic-based ontology languages**, which can be though of as specific FOL formalisms for class-based representation (cf. UML class diagrams or ER):

<table>
<thead>
<tr>
<th>CQ query answering</th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete databases</td>
<td>NP-complete</td>
<td>in $\text{AC}^0(1)$</td>
</tr>
<tr>
<td>Incomplete databases (naive tables, OWA) no TBox</td>
<td>NP-complete</td>
<td>in $\text{AC}^0$</td>
</tr>
<tr>
<td>UML Class Diagrams or $\text{OWL2}^\ast$ TBoxes</td>
<td>ExpTime [2]-hard</td>
<td>$\text{coNP}$-hard</td>
</tr>
</tbody>
</table>

* $\text{OWL2}$ is a W3C standard based on Description Logics (DLs).

(1) $\text{AC}^0 \subseteq \text{LogSpace}$ is the cost of evaluating FOL over relational DBs — This is what we need to scale with the data.

---

**The key question at the base of development of $\text{DL-Lite}$**

How do we find interesting logics for expressing ontologies for which the query answering

- can be solved in $\text{AC}^0$, and
- we leverage on evaluation and relational database technology for query answering?
Data complexity of query answering in ontologies

QA has been studied extensively for (unions of) CQs in the context of Description Logic-based ontology languages, which can be though of as specific FOL formalisms for class-based representation (cf. UML class diagrams or ER):

<table>
<thead>
<tr>
<th>CQ query answering</th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete databases</td>
<td>NP-complete</td>
<td>in AC⁰(1)</td>
</tr>
<tr>
<td>Incomplete databases (naive tables, OWA) no TBox</td>
<td>NP-complete</td>
<td>in AC⁰</td>
</tr>
<tr>
<td>UML Class Diagrams or OWL²* TBoxes</td>
<td>ExpTime [2]-hard</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

* OWL 2 is a W3C standard based on Description Logics (DLs).

(1) $AC⁰ \subseteq \text{LogSpace}$ is the cost of evaluating FOL over relational DBs – This is what we need to scale with the data.

The key question at the base of development of DL-Lite

How do we find interesting logics for expressing ontologies for which the query answering

- can be solved in $AC⁰$, and

- we leverage on evaluation and relational database technology for query answering?
Data complexity of query answering in ontologies

QA has been studied extensively for (unions of) CQs in the context of Description Logic-based ontology languages, which can be thought of as specific FOL formalisms for class-based representation (cf. UML class diagrams or ER):

<table>
<thead>
<tr>
<th>CQ query answering</th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete databases</td>
<td>NP-complete</td>
<td>in AC^0 (1)</td>
</tr>
<tr>
<td>Incomplete databases (naive tables, OWA) no TBox</td>
<td>NP-complete</td>
<td>in AC^0</td>
</tr>
<tr>
<td>UML Class Diagrams or OWL2* TBoxes</td>
<td>ExpTime [2]-hard</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

*OWL 2 is a W3C standard based on Description Logics (DLs).

(1) AC^0 ⊆ LOGSPACE is the cost of evaluating FOL over relational DBs – This is what we need to scale with the data.

The key question at the base of development of DL-Lite

How do we find interesting logics for expressing ontologies for which the query answering

- can be solved in AC^0, and
- we leverage on evaluation and relational database technology for query answering?
Compiling inference into evaluation for query answering

\[ q \xrightarrow{} \text{Logical inference} \xrightarrow{} \text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) \]

To be able to deal with data efficiently, we need to separate the contribution of \( \mathcal{A} \) from the contribution of \( q \) and \( \mathcal{T} \) and use evaluation.

\( \sim \) Query answering by query rewriting.
Query rewriting

Query answering can always be thought as done in two phases:

1. **Perfect rewriting**: produce from \( q \) and the TBox \( T \) a new query \( \text{rew} q, T \) (called the perfect rewriting of \( q \) w.r.t. \( T \)).

2. **Query evaluation**: evaluate \( \text{rew} q, T \) over the ABox \( A \) seen as a complete database (and without considering the TBox \( T \)).

\[ \rightsquigarrow \text{Produces } \text{cert}(q, \langle T, A \rangle). \]

Note: The “always” holds if we pose no restriction on the language in which to express the rewriting \( \text{rew} q, T \).
Language of the rewriting

The expressiveness of the KB language affects the query language into which we are able to rewrite CQs:

- When we can rewrite into **FOL/SQL**.
  - Query evaluation can be done in SQL, i.e., via an RDBMS *(Note: FOL is in \(AC^0\)).*

- When we can rewrite into an **NLogSpace-hard** language.
  - Query evaluation requires (at least) linear recursion.

- When we can rewrite into a **PTime-hard** language.
  - Query evaluation requires full recursion (e.g., Datalog).

- When we can rewrite into a **coNP-hard** language.
  - Query evaluation requires (at least) power of Disjunctive Datalog.
## Language of the rewriting

The expressiveness of the KB language affects the query language into which we are able to rewrite CQs:

- **When we can rewrite into FOL/SQL.**
  $\rightarrow$ Query evaluation can be done in SQL, i.e., via an RDBMS (Note: FOL is in $\mathsf{AC}^0$).

- **When we can rewrite into an $\mathsf{NLogSpace}$-hard language.**
  $\rightarrow$ Query evaluation requires (at least) linear recursion.

- **When we can rewrite into a $\mathsf{PTime}$-hard language.**
  $\rightarrow$ Query evaluation requires full recursion (e.g., Datalog).

- **When we can rewrite into a $\mathsf{coNP}$-hard language.**
  $\rightarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.
Language of the rewriting

The expressiveness of the KB language affects the query language into which we are able to rewrite CQs:

- When we can rewrite into FOL/SQL.
  \(\leadsto\) Query evaluation can be done in SQL, i.e., via an RDBMS (Note: FOL is in AC\(^0\)).

- When we can rewrite into an NLogSpace-hard language.
  \(\leadsto\) Query evaluation requires (at least) linear recursion.

- When we can rewrite into a PTime-hard language.
  \(\leadsto\) Query evaluation requires full recursion (e.g., Datalog).

- When we can rewrite into a coNP-hard language.
  \(\leadsto\) Query evaluation requires (at least) power of Disjunctive Datalog.
Language of the rewriting

The expressiveness of the KB language affects the query language into which we are able to rewrite CQs:

- When we can rewrite into **FOL/SQL**.
  - Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note: FOL is in AC⁰*).

- When we can rewrite into an **NLogSpace-hard** language.
  - Query evaluation requires (at least) linear recursion.

- When we can rewrite into a **PTime-hard** language.
  - Query evaluation requires full recursion (e.g., **Datalog**).

- When we can rewrite into a **coNP-hard** language.
  - Query evaluation requires (at least) power of **Disjunctive Datalog**.
Beyond $DL\text{-}Lite_A$: results on data complexity

<table>
<thead>
<tr>
<th>Lhs</th>
<th>rhs</th>
<th>funct.</th>
<th>Prop. incl.</th>
<th>Data complexity of query answering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DL\text{-}Lite_A$</td>
<td>$A$</td>
<td>yes*</td>
<td>yes*</td>
<td>in $AC^0$</td>
</tr>
<tr>
<td>$A \mid \exists P.A$</td>
<td>$A$</td>
<td>no</td>
<td>no</td>
<td>$N\text{LogSpace}$-hard</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \mid \forall P.A$</td>
<td>no</td>
<td>no</td>
<td>$N\text{LogSpace}$-hard</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \mid \exists P.A$</td>
<td>yes</td>
<td>no</td>
<td>$N\text{LogSpace}$-hard</td>
</tr>
<tr>
<td>$A \mid \exists P.A \mid A_1 \cap A_2$</td>
<td>$A$</td>
<td>no</td>
<td>no</td>
<td>$P\text{Time}$-hard</td>
</tr>
<tr>
<td>$A \mid A_1 \cap A_2$</td>
<td>$A \mid \forall P.A$</td>
<td>no</td>
<td>no</td>
<td>$P\text{Time}$-hard</td>
</tr>
<tr>
<td>$A \mid A_1 \cap A_2$</td>
<td>$A \mid \exists P.A$</td>
<td>yes</td>
<td>no</td>
<td>$P\text{Time}$-hard</td>
</tr>
<tr>
<td>$A \mid \exists P.A \mid \exists P^- . A$</td>
<td>$A \mid \exists P$</td>
<td>no</td>
<td>no</td>
<td>$P\text{Time}$-hard</td>
</tr>
<tr>
<td>$A \mid \exists P \mid \exists P^-$</td>
<td>$A \mid \exists P \mid \exists P^-$</td>
<td>yes</td>
<td>yes</td>
<td>$P\text{Time}$-hard</td>
</tr>
<tr>
<td>$A \mid \neg A$</td>
<td>$A$</td>
<td>no</td>
<td>no</td>
<td>$coNP$-hard</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \mid A_1 \sqcup A_2$</td>
<td>no</td>
<td>no</td>
<td>$coNP$-hard</td>
</tr>
<tr>
<td>$A \mid \forall P.A$</td>
<td>$A$</td>
<td>no</td>
<td>no</td>
<td>$coNP$-hard</td>
</tr>
</tbody>
</table>

Notes:

- * with the “proviso” of not specializing functional properties.
- $N\text{LogSpace}$ and $P\text{Time}$ hardness holds already for instance checking.
- For $coNP$-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\sim$ No hope of including covering constraints.
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
SitCalc and DLs

A natural idea

- Description Logics (DLs) capture virtually all conceptual data models;
- SitCalc is possibly the best known formalism in Reasoning about Actions.

⇒
- Represent data at the conceptual level with DLs ontologies
- Represent processes in SitCalc, possibly using Golog/ConGolog.

More precisely

- DL KB describing an ontology can be seen as FOL “static constraints” in SitCalc.
- Use the single theory obtained by combining the DL KB and the SitCalc action theory to represent and reason on actions over the ontology.

Unfortunately, DLs + SitCalc action theories are deeply undecidable.
SitCalc and DLs

A natural idea

- **Description Logics (DLs)** capture virtually all conceptual data models;
- **SitCalc** is possibly the best known formalism in Reasoning about Actions.
- **⇒**
  - Represent data at the conceptual level with **DLs** ontologies
  - Represent processes in **SitCalc**, possibly using Golog/ConGolog.

More precisely

- DL KB describing an ontology can be seen as FOL “static constraints” in SitCalc.
- Use the single theory obtained by combining the DL KB and the SitCalc action theory to represent and reason on actions over the ontology.

Unfortunately, DLs + SitCalc action theories are deeply undecidable.
SitCalc and DLs

A natural idea

- **Description Logics** (DLs) capture virtually all conceptual data models;
- **SitCalc** is possibly the best known formalism in Reasoning about Actions.

\[ \rightarrow \]
- Represent data at the conceptual level with **DLs** ontologies
- Represent processes in **SitCalc**, possibly using Golog/ConGolog.

More precisely

- **DL KB** describing an ontology can be seen as FOL “static constraints” in **SitCalc**.
- Use the single theory obtained by combining the DL KB and the SitCalc action theory to represent and reason on actions over the ontology.

Unfortunately, DLs + SitCalc action theories are deeply undecidable.
SitCalc and DLs

A natural idea

- **Description Logics** (DLs) capture virtually all conceptual data models;
- **SitCalc** is possibly the best known formalism in Reasoning about Actions.
- \[\Rightarrow\]
  - Represent **data** at the conceptual level with **DLs** ontologies
  - Represent **processes** in **SitCalc**, possibly using Golog/ConGolog.

More precisely

- **DL KB** describing an ontology can be seen as FOL “static constraints” in **SitCalc**.
- Use the **single theory** obtained by combining the DL KB and the SitCalc action theory to represent and reason on actions over the ontology.

Unfortunately, **DLs + SitCalc action theories** are deeply undecidable.
SitCalc and DLs

A natural idea

- **Description Logics** (DLs) capture virtually all conceptual data models;
- **SitCalc** is possibly the best known formalism in Reasoning about Actions.

⇒

- Represent **data** at the conceptual level with **DLs** ontologies
- Represent **processes** in **SitCalc**, possibly using Golog/ConGolog.

More precisely

- **DL KB** describing an ontology can be seen as FOL “static constraints” in **SitCalc**.

- Use the **single theory** obtained by combining the DL KB and the SitCalc action theory to **represent and reason on actions over the ontology**.

Unfortunately, **DLs + SitCalc action theories** are deeply undecidable.
SitCalc and DLs

A natural idea
- Description Logics (DLs) capture virtually all conceptual data models;
- SitCalc is possibly the best known formalism in Reasoning about Actions.

⇒
- Represent data at the conceptual level with DLs ontologies
- Represent processes in SitCalc, possibly using Golog/ConGolog.

More precisely
- DL KB describing an ontology can be seen as FOL “static constraints” in SitCalc.

- Use the single theory obtained by combining the DL KB and the SitCalc action theory to represent and reason on actions over the ontology.

Unfortunately, DLs + SitCalc action theories are deeply undecidable.
Deep Undecidability DLs + Actions Theory

DLs + SitCalc undecidability holds already for:

- The simplest kind of DLs:
  - $\text{DL-Lite}_{\text{core}}$ (i.e., OWL-QL profile of OWL2)
  - $\mathcal{EL}$ (i.e., OWL-EL profile of OWL2)

- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - “Local Effect”, where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - “Context Free”, where successor state axioms changes do not depend on properties holding in the current situation

Theorem

Satisfiability of $\text{DL-Lite}_{\text{core}}/\mathcal{EL}$ KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

DLs + SitCalc undecidability holds already for:

- The simplest kind of DLs:
  - $DL$-Lite$_{core}$ (i.e., OWL-QL profile of OWL2)
  - $\mathcal{E}\mathcal{L}$ (i.e., OWL-EL profile of OWL2)
- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - “Local Effect”, where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - “Context Free”, where successor state axioms changes do not depend on properties holding in the current situation

Theorem

Satisfiability of $DL$-Lite$_{core}$/$\mathcal{E}\mathcal{L}$ KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

DLs + SitCalc undecidability holds already for:

- The simplest kind of DLs:
  - DL-Lite$_{\text{core}}$ (i.e., OWL-QL profile of OWL2)
  - $\mathcal{EL}$ (i.e., OWL-EL profile of OWL2)

- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - “Local Effect”, where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - “Context Free”, where successor state axioms changes do not depend on properties holding in the current situation

Theorem

Satisfiability of DL-Lite$_{\text{core}}$/EL KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

**DLs + SitCalc undecidability holds already for:**

- The simplest kind of DLs:
  - \( DL\text{-Lite}_{\text{core}} \) (i.e., OWL-QL profile of OWL2)
  - \( \mathcal{EL} \) (i.e., OWL-EL profile of OWL2)
- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - “Local Effect”, where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - “Context Free”, where successor state axioms changes do not depend on properties holding in the current situation

**Theorem**

Satisfiability of \( DL\text{-Lite}_{\text{core}}/\mathcal{EL} \) KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

DLs + SitCalc undecidability holds already for:

- The simplest kind of DLs:
  - \(DL-\text{Lite}_{\text{core}}\) (i.e., OWL-QL profile of OWL2)
  - \(\mathcal{EL}\) (i.e., OWL-EL profile of OWL2)

- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - “Local Effect”, where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - “Context Free”, where successor state axioms changes do not depend on properties holding in the current situation

Theorem

Satisfiability of \(DL-\text{Lite}_{\text{core}}/\mathcal{EL}\) KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

### DLs + SitCalc undecidability holds already for:

- **The simplest kind of DLs:**
  - *DL-Lite*$_{core}$ (i.e., OWL-QL profile of OWL2)
  - *EL* (i.e., OWL-EL profile of OWL2)
- **The simplest kind of SitCalc action theories, those satisfying these 2 properties:**
  - *“Local Effect”,* where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - *“Context Free”,* where successor state axioms changes do not depend on properties holding in the current situation

### Theorem

Satisfiability of *DL-Lite*$_{core}$/*EL* KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

DLs + SitCalc undecidability holds already for:

- The simplest kind of DLs:
  - $DL\text{-}$$\text{Lite}_{\text{core}}$ (i.e., OWL-QL profile of OWL2)
  - $\mathcal{EL}$ (i.e., OWL-EL profile of OWL2)

- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - "Local Effect", where successor state axioms change only the properties of objects mentioned explicitly in the action parameters"
  - "Context Free", where successor state axioms changes do not depend on properties holding in the current situation

Theorem

Satisfiability of $DL\text{-}$$\text{Lite}_{\text{core}}/\mathcal{EL}$ KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Deep Undecidability DLs + Actions Theory

DLs + SitCalc undecidability holds already for:

- The simplest kind of DLs:
  - $DL-\text{Lite}_{\text{core}}$ (i.e., OWL-QL profile of OWL2)
  - $\mathcal{EL}$ (i.e., OWL-EL profile of OWL2)
- The simplest kind of SitCalc action theories, those satisfying these 2 properties:
  - “Local Effect”, where successor state axioms change only the properties of objects mentioned explicitly in the action parameters
  - “Context Free”, where successor state axioms changes do not depend on properties holding in the current situation

Theorem

Satisfiability of $DL-\text{Lite}_{\text{core}}/\mathcal{EL}$ KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.
Theorem

Satisfiability of DL-Lite$\text{core}/\mathcal{EL}$ KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.

Proof: crux of the reduction (in pictures)
Deep Undecidability DLs + Actions Theory

**Theorem**

Satisfiability of DL-Lite$_{core}$/$\mathcal{EL}$ KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.

**Proof: crux of the reduction (in pictures)**

[Diagram of a tiling grid and a Turing machine computation]
Deep Undecidability DLs + Actions Theory

Theorem

Satisfiability of DL-Lite\textsubscript{core}/\mathcal{E}\mathcal{L} KBs + SitCalc “Local Effect” and “Context Free” SitCalc action theories is undecidable.

Proof: crux of the reduction (in pictures)

Tiling Grid

Turing Machine Computation
Deep Undecidability of DLs + Actions Theory

Rationale

DLs + Actions theories are undecidable even in the simplest cases.

(In fact related to, undecidability of multi-dimensional modal logics, which is computationally nasty [BaaderLauxIJCAI95], [WolterZakharyaschevKR98, IJCAI99, FroCoS99, Fl99], [GabbayEtAl03].)

How to regain decidability?

- Allow changes only of concepts (not roles)
  e.g., [GabbayEtal03],[ArtaleFranconi99],[BasultoJungLutz12],[Jamroga12]

- Drop TBox (or make it acyclic)
  e.g., [BaaderLutzMilicicSattlerWolterAAAI05,GuSoutchanskiIJCAI07]

- Drop persistence of TBox (ontology is not maintained by actions)
  e.g., [GuSoutchanskiAMAI10]

All these restrictions are unsuitable for conceptual models of data + processes!

Are there other options?

YES: adopt “Levesque’s functional approach to KBs”
Deep Undecidability of DLs + Actions Theory

Rationale

DLs + Actions theories are undecidable even in the simplest cases.

(In fact related to, undecidability of multi-dimensional modal logics, which is computationally nasty [BaaderLauxIJCAI95], [WolterZakharyaschevKR98, IJCAI99, FroCoS99, FI99], [GabbayEtAlt03].)

How to regain decidability?

- **Allow changes only of concepts (not roles)**
  
e.g., [GabbayEtal03],[ArtaleFranconi99],[BasultoJungLutz12],[Jamroga12]

- **Drop TBox (or make it acyclic)**
  
e.g., [BaaderLutzMilicicSattlerWolterAAAI05,GuSoutchanskiIJCAI07]

- **Drop persistence of TBox (ontology is not maintained by actions)**
  
e.g., [GuSoutchanskiAMAI10]

All these restrictions are unsuitable for conceptual models of data + processes!

Are there other options?

YES: adopt “Levesque’s functional approach to KBs”
Deep Undecidability of DLs + Actions Theory

Rationale
DLs + Actions theories are undecidable even in the simplest cases.

(In fact related to, undecidability of multi-dimensional modal logics, which is computationally nasty [BaaderLauxIJCAI95], [WolterZakharyaschevKR98, IJCAI99, FroCoS99, Fl99], [GabbayEtAlt03].)

How to regain decidability?

- Allow changes only of concepts (not roles)
  e.g., [GabbayEtal03],[ArtaleFranconi99],[BasultoJungLutz12],[Jamroga12]

- Drop TBox (or make it acyclic)
  e.g., [BaaderLutzMilicicSattlerWolterAAAI05,GuSoutchanskiIJCAI07]

- Drop persistence of TBox (ontology is not maintained by actions)
  e.g., [GuSoutchanskiAMAI10]

All these restrictions are unsuitable for conceptual models of data + processes!

Are there other options?
YES: adopt “Levesque’s functional approach to KBs”
Deep Undecidability of DLs + Actions Theory

Rationale

DLs + Actions theories are undecidable even in the simplest cases.

(In fact related to, undecidability of multi-dimensional modal logics, which is computationally nasty [BaaderLauxIJCAI95], [WolterZakharyaschevKR98, IJCAI99, FroCoS99, FL99], [GabbayEtAl03].)

How to regain decidability?

- Allow changes only of concepts (not roles)
  e.g., [GabbayEtal03],[ArtaleFranconi99],[BasultoJungLutz12],[Jamroga12]

- Drop TBox (or make it acyclic)
  e.g., [BaaderLutzMilicicSattlerWolterAAAI05,GuSoutchanskiIJCAI07]

- Drop persistence of TBox (ontology is not maintained by actions)
  e.g., [GuSoutchanskiAMAI10]

All these restrictions are unsuitable for conceptual models of data + processes!

Are there other options?

YES: adopt “Levesque’s functional approach to KBs”
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Levesque’s Functional Approach to KBs


Levesque’s functional approach

View KB as systems that allow for two kinds of operations

- **ASK**(*q*, *s*), which returns the answers to a query *q* that are logically implied by the ontology *s*,
- **TELL**(*a*, *s*), which produces a new ontology *s*′ as a result of the application of an action *a* to the ontology *s*.

**NB1:** Essentially it amounts to applying actions/temporal operators to DL axioms only.

**NB2:** Also related to update [LiuLutzMilicicWolterKR06]. But result of TELL must remain in the same language of the original ontology.
Levesque’s Functional Approach

Major advantage:

• It strongly **decouples** reasoning on the **static knowledge** from the one on the **dynamics of the computations** over such knowledge

• As a result, we can lift to DLs many notions and results developed over the years in Reasoning about Actions, Process Modeling, and Verification [CalvaneseDeGiacomoLenzeriniRosatiDL07], [BaaderGhilardiLutzKR08], [ACSI-ECAI12], [ACSI-JAIR12], [ACSI-RR12], [ACSI-RR13].

Disadvantages:

• We **don’t have a single theory** anymore for representing and reasoning on actions over ontologies
  - **the ontology** represents what is known
  - **actions** change what is known (ie, the ontology), but they are not represented in the (same) ontology

• We lose the possibility of distinguishing between “knowledge” and “truth”, see [SardinaDeGiacomoLesperanceLevesqueKR06], and [ECAI14].
Levesque’s Functional Approach

Major advantage:

- It strongly **decouples** reasoning on the **static knowledge** from the one on the **dynamics of the computations** over such knowledge.

- As a result, we can lift to DLs many notions and results developed over the years in Reasoning about Actions, Process Modeling, and Verification: [CalvaneseDeGiacomoLenzeriniRosatiDL07], [BaaderGhilardiLutzKR08], [ACSI-ECAI12], [ACSI-JAIR12], [ACSI-RR12], [ACSI-RR13].

Disadvantages:

- We don’t have a **single theory** anymore for representing and reasoning on actions over ontologies:
  - the **ontology** represents what is known
  - **actions** change what is known (i.e., the ontology), but they are not represented in the (same) ontology

- We lose the possibility of distinguishing between “**knowledge**” and “**truth**”, see [SardinaDeGiacomoLesperanceLevesqueKR06], and [ECAI14].
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Knowledge and Action Bases (KAB)

**Idea:** follow Levesque’s functional approach and build artifact systems whose information model is a KB/ontology, to:

- **better capture the domain semantics at the conceptual level;**
- **take into account incomplete information.**

**Ontology**

- **Data Layer:** Description logic KB
  - Data schema: TBox
  - Data instance: ABox *(possibly virtual in ontology-based data access, OBDA, systems)*

**Process**

- **Process Layer:**
  - Atomic actions: access and update data;
  - Process: finite state control over conditional action invocation;
  - External calls: communication with external environment
  - * Insert new data objects possibly depending on already present objects.

**KAB**
Knowledge and Action Bases (KAB)

Idea: follow Levesque’s functional approach and build artifact systems whose information model is a KB/ontology, to:

- better capture the domain semantics at the conceptual level;
- take into account incomplete information.

![Ontology + Process = KAB](image)

**KAB**

- **Data Layer**: Description logic KB
  - Data schema: TBox
  - Data instance: ABox *(possibly virtual in ontology-based data access, OBDA, systems)*

- **Process Layer**:
  - Atomic actions: access and update data;
  - Process: finite state control over conditional action invocation;
  - External calls: communication with external environment
    * Insert new data objects possibly depending on already present objects.*
Knowledge and Action Base (KAB)

**Actions**
- Have input parameters.
- Action execution results in a new KB: ABox changes while TBox remains fixed.
- Resulting KBs may contain new objects that come from the environment outside the system.

**Processes**
A process over a KAB is a specification of when actions can be executed.
- Must be based on querying the current ontology.
- With additional non-deterministic finite control state programs.
Knowledge and Action Base (KAB)

Actions

- Have input parameters.
- Action execution results in a new KB: ABox changes while TBox remains fixed.
- Resulting KBs may contain new objects that come from the environment outside the system.

Processes

A process over a KAB is a specification of when actions can be executed.

- Must be based on querying the current ontology.
- With additional non-deterministic finite control state programs.
KAB Transition System

KAB behavior captured by THE infinite-state transition system $\Upsilon$, defined as follows:

1. Start from initial state $A_0$.
2. Use process specification to choose instantiated action $\rho\sigma$ to execute.
3. If resulting state/ABox is consistent wrt TBox, it becomes a new state.
4. Repeat forever.
KAB Transition System

KAB behavior captured by THE infinite-state transition system \( \Upsilon \), defined as follows:

1. Start from initial state \( A_0 \).
2. Use process specification to choose instantiated action \( \rho\sigma \) to execute.
3. If resulting state/AX is consistent wrt TBox, it becomes a new state.
4. Repeat forever.
KAB Transition System

KAB behavior captured by THE infinite-state transition system \( \Upsilon \), defined as follows:

1. Start from initial state \( A_0 \).
2. Use process specification to choose instantiated action \( \rho \sigma \) to execute.
3. If resulting state/\( \text{ABox} \) is consistent \( \text{wrt TBox} \), it becomes a new state.
4. Repeat forever.
KAB Transition System

KAB behavior captured by THE infinite-state transition system $\Upsilon$, defined as follows:

1. Start from initial state $A_0$.
2. Use process specification to choose instantiated action $\rho\sigma$ to execute.
3. If resulting state/ABox is consistent wrt TBox, it becomes a new state.
4. Repeat forever.
The data layer in an artifact system might be very complex, and difficult to “govern”.

Semantically-Enhanced Artifact Systems

We exploit ontology-based technology and ontology-based data access (OBDA) techniques to support users:

- We install “on top” of an artifact system an ontology, capturing the domain of interest at a higher level of abstraction.
- We connect the ontology to the underlying artifact system via declarative mappings.
- Data remain at the sources, but are virtually viewed through the ontology.
- Transitions remain defined over the sources.
The data layer in an artifact system might be very complex, and difficult to “govern”.

Semantically-Enhanced Artifact Systems

We exploit ontology-based technology and ontology-based data access (OBDA) techniques to support users:

- We install “on top” of an artifact system an ontology, capturing the domain of interest at a higher level of abstraction.
- We connect the ontology to the underlying artifact system via declarative mappings.
- Data remain at the sources, but are virtually viewed through the ontology
- Transitions remain defined over the sources.
The data layer in an artifact system might be very complex, and difficult to “govern”.

**Semantically-Enhanced Artifact Systems**

We exploit ontology-based technology and ontology-based data access (OBDA) techniques to support users:

- We install “on top” of an artifact system an **ontology**, capturing the domain of interest at a higher level of abstraction.
- We connect the ontology to the underlying artifact system via **declarative mappings**.
- Data remain at the sources, but are virtually viewed through the ontology
- Transitions remain defined over the sources.
The data layer in an artifact system might be very complex, and difficult to “govern”.

**Semantically-Enhanced Artifact Systems**

We exploit ontology-based technology and ontology-based data access (OBDA) techniques to support users:

- We install “on top” of an artifact system an **ontology**, capturing the domain of interest at a higher level of abstraction.
- We connect the ontology to the underlying artifact system via **declarative mappings**.
- Data remain at the sources, but are **virtually** viewed through the ontology.
- Transitions remain defined over the sources.
Knowledge and Action Bases (KAB)

Artifacts System conceptual schema (TBox) composed of semantic constraints that define the “data boundaries” of the artifact system.
Semantic layer and snapshots

Actual data are concretely maintained at the artifact layer. **Snapshot**: database instances of artifacts.
Mappings

Each snapshot is conceptualized in the ontology as instance data. Mappings define how to obtain the virtual ABox from the source data.
Action execution to evolve the system

The system evolves thanks to actions/process executed over the artifact layer, invoking external services to inject new data.
Understanding the evolution

Semantic layer used to **understand** the evolution at the conceptual level, by posing queries over the ontology.

![Diagram showing the evolution of an artifact system with semantic layers and queries](image-url)
Semantic Governance

Semantic layer used to regulate the execution of actions at the artifact layer by rejecting actions that lead to violations of constraints in the ontology.
Temporal Verification over Semantic Layer

Temporal properties expressed as:

- queries over the ontology combined with
- temporal operators to talk about the dynamics of the system.

System evolves at the Artifact Layer.

Rewriting of temporal properties

- The temporal part is maintained unaltered, because the system evolves at the Artifact Layer.
- Faithful transformation of a temporal property over Semantic Layer:
  1. Rewriting of ontology queries to compile away the TBox.
  2. Unfolding of temporal property wrt mappings to obtain a corresponding temporal property over the Artifact Layer.

Hence, verification of temporal properties expressed over the ontology is reduced to verification of temporal properties over the artifacts.
Temporal Verification over Semantic Layer

Temporal properties expressed as:
- queries over the ontology combined with
- temporal operators to talk about the dynamics of the system.

System evolves at the Artifact Layer.

Rewriting of temporal properties

- The temporal part is maintained unaltered, because the system evolves at the Artifact Layer.
- Faithful transformation of a temporal property over Semantic Layer:
  1. **Rewriting** of ontology queries to compile away the TBox.
  2. **Unfolding** of temporal property wrt mappings to obtain a corresponding temporal property over the Artifact Layer.

Hence, verification of temporal properties expressed over the ontology is reduced to verification of temporal properties over the artifacts.
Decidability of verification

Verification technique

We compile the transition system generated by the KAB into a transition system generated by a DCDS. Then we do faithful abstraction getting decidability under run bounded and state bounded contions for $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$ respectively.

Semantic Transition System

Relational Transition System

Abstract Transition System

$$\Phi = \text{UNFOLD}(\text{REW}(\Phi, T), M)$$
Outline

1. Data and Processes
2. Artifact-Centric Approach
3. Data-Centric Dynamic Systems
4. Situation Calculus
5. Description Logics
6. SitCalc and DLs
7. Levesque’s Functional Approach to KBs
8. Knowledge and Action Bases
9. Conclusion
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies

- Automated synthesis also possible (*not treated in this talk*).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies

- Automated synthesis also possible (not treated in this talk).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies

- Automated synthesis also possible *(not treated in this talk).*
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies

- Automated synthesis also possible (not treated in this talk).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.
- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness
- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)
- Combining DLs and SitCalc in a single theory is too fragile.
- Levesque’s functional approach is very robust.
- Compilation techniques at the base of OBDA carry over to processes with ontologies
- Automated synthesis also possible (*not treated in this talk*).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies.

- Automated synthesis also possible (*not treated in this talk*).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies

- Automated synthesis also possible \((not\ treated\ in\ this\ talk)\).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.

- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness

- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)

- Combining DLs and SitCalc in a single theory is too fragile.

- Levesque’s functional approach is very robust.

- Compilation techniques at the base of OBDA carry over to processes with ontologies

- Automated synthesis also possible (*not treated in this talk*).
Conclusion

What to bring home:

- Conceptual modeling of data and processes and support for reasoning is important, but largely neglected.
- Key progresses obtained in the very last years
  - Run-boundedness
  - State-boundedness
- They apply to SitCalc (the main AI formalism for Knowledge Representation of Actions!)
- Combining DLs and SitCalc in a single theory is too fragile.
- Levesque’s functional approach is very robust.
- Compilation techniques at the base of OBDA carry over to processes with ontologies
- Automated synthesis also possible (*not treated in this talk*).
Thanks!

Involved in this research

- Diego Calvanese
- Alin Deutsch
- Rick Hull
- Yves Lesperance
- Marco Montali
- Fabio Patrizi
- Mikhail Soutchanski
- Stavros Vassos
- Babak Bagheri Hariri
- Riccardo De Masellis
- Paolo Felli

Sapienza

Description Logics Group

- Domenico Lembo
- Maurizio Lenzerini
- Antonella Poggi
- Riccardo Rosati
- Marco Ruzzi
- Domenico Fabio Savo
- Cristina Civili (PhD)
- Marco Console (PhD)
- Lorenzo Lepore (PhD)
- Valerio Santarelli (PhD)

Service Oriented Computing Group

- Massimo Mecella
- Mario Caruso
- Claudio Di Ciccio
- Francesco Leotta
- Andrea Marrella
- Alessandro Russo

EU Project ACSI

- Marlon Dumas
- Lior Limonad
- Alessio Lomuscio
- Wil van der Aalst
References

References II

- [BoundedSitCalcAAMAS14] Giuseppe De Giacomo, Yves Lesperance, Fabio Patrizi, Stavros Vassos. Progression and Verification of Situation Calculus Agents with Bounded Beliefs. AAMAS 2014
References