Verification and Synthesis in Description Logic-Based Dynamic Systems

Marco Montali

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

Joint work with Diego Calvanese, Giuseppe De Giacomo, Fabio Patrizi

RR 2013
Motivation

The information assets of an organization are constituted by:

- **data**, and
- **processes**, that determine how data change and evolve over time.

Conceptual Modeling

Both aspects are modelled conceptually, but:

- Using different modeling tools
- By different teams with different competences
- Their connection is **NOT** modelled conceptually, but should!

▶ See a series of 2009 reports by Forrester.

Consequence

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

The information assets of an organization are constituted by:

- **data**, and
- **processes**, that determine how data change and evolve over time.

Conceptual Modeling

Both aspects are modelled conceptually, but:

- Using different modeling tools
- By different teams with different competences
- Their connection is **NOT** modelled conceptually, but **should**!
  
  See a series of 2009 reports by Forrester.
Motivation

The information assets of an organization are constituted by:

- data, and
- processes, that determine how data change and evolve over time.

Conceptual Modeling

Both aspects are modelled conceptually, but:

- Using different modeling tools
- By different teams with different competences
- Their connection is NOT modelled conceptually, but should!
  - See a series of 2009 reports by Forrester.

Consequence

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

Formalization


- **Static component**: full-fledged Description Logic KB.
- **Dynamic component**: parametric actions.
  - Progress the system by evolving the ABox - TBox is fixed.
  - Parameters: inject new values from the external world.

Verification

Check whether the system satisfies temporal/dynamic properties.

- Analyzing all possible executions.

Adversarial Synthesis

Action separation + alternation: system actions vs environment actions. System goal: force the execution to satisfy a desired property.
Concrete Example
Dynamic web applications.

- **Current data** \(\sim\) fields, text, highlights.
  - Include **status attributes**.
- **Actions** \(\sim\) buttons, links.
- **Input params** \(\sim\) writable fields.
Concrete Example

Dynamic web applications.

- **Current data** $\rightarrow$ fields, text, highlights.
  - Include *status attributes*.
- **Actions** $\rightarrow$ buttons, links.
- **Input params** $\rightarrow$ writable fields.
  - Their number *depends on the current data*!
Direct Combination of KBs and Actions

Idea: combine DL KB and actions into a single logical theory.

Problem

This gives raise to a many-dimensional modal logic $\sim$ undecidability.
Direct Combination of KBs and Actions

Idea: combine DL KB and actions into a single logical theory.

Problem

This gives raise to a many-dimensional modal logic \( \sim \) undecidability.

Conclusion

Unsuitable for combining data+processes.
Direct Combination of KBs and Actions

Idea: combine DL KB and actions into a single logical theory.

Problem

This gives raise to a many-dimensional modal logic \( \sim \) undecidability.

Decidability regained with restrictions on the combination.

- E.g., restricted to concepts only.

Conclusion

Unsuitable for combining data + processes.
Levesque’s Functional Approach

Radical solution: functional view of KBs [Levesque, AIJ1984].

Levesque’s Functional approach

Knowledge base supports:

- \text{ASK}(q, KB_{cur})\), which returns the certain answers to a query \(q\) that are logically implied by \(KB_{cur}\).
- \text{TELL}(a, KB_{cur})\), which produces a new ontology \(KB_{new}\) as a result of the application of an action \(a\) to \(KB_{cur}\).

Actions represented outside the KB.

(+) Decoupling between static knowledge and dynamics of computation.

(−) Knowledge and truth become indistinguishable [Sardina et al., KR06].

Goal

Formalize Description Logic-Based Dynamic Systems (DLDS) following Levesque’s functional approach.
Description Logic-Based Dynamic Systems

Parametric wrt DL and progression mechanism.

**DLDS**

Tuple \((T, A_0, \Gamma)\), where

- \((T, A_0)\): initial DL KB \(\leadsto\) during system progression: \(A_0\) evolves.
- \(\Gamma\): parametric actions \((\pi, \tau)\) that evolve the system. Given ABox A:
  - \(\pi\): determines action params depending on \(A\).
  - \(\tau\): given \(A+\)param values is undefined or gives new T-consistent ABox.
Description Logic-Based Dynamic Systems

Parametric wrt DL and progression mechanism.

**DLDS**

Tuple \((T, A_0, \Gamma)\), where

- \((T, A_0)\): initial DL KB \(\sim\) during system progression: \(A_0\) evolves.
- \(\Gamma\): parametric actions \((\pi, \tau)\) that evolve the system. Given ABox \(A\):
  - \(\pi\): determines action params depending on \(A\).
  - \(\tau\): given \(A\)+param values is undef. or gives new \(T\)-consistent ABox.

![Diagram of DLDS components: current state, ask/tell, param selection, and ABox.](image)
Description Logic-Based Dynamic Systems

Parametric wrt DL and progression mechanism.

**DLDS**

Tuple \((T, A_0, \Gamma)\), where

- \((T, A_0)\): initial DL KB \(\sim\) during system progression: \(A_0\) evolves.
- \(\Gamma\): parametric actions \((\pi, \tau)\) that evolve the system. Given ABox \(A\):
  - \(\pi\): determines action params depending on \(A\).
  - \(\tau\): given \(A\)+param values is undefined or gives new \(T\)-consistent ABox.

Note: param values taken from countably infinite set.
Description Logic-Based Dynamic Systems

Parametric wrt DL and progression mechanism.

**DLDS**

Tuple \((T, A_0, \Gamma)\), where

- \((T, A_0)\): initial DL KB \(\sim\) during system progression: \(A_0\) evolves.
- \(\Gamma\): parametric actions \((\pi, \tau)\) that evolve the system. Given ABox \(A\):
  - \(\pi\): determines action params depending on \(A\).
  - \(\tau\): given \(A +\) param values is undef. or gives new \(T\)-consistent ABox.

Note: \(\text{ADOM}(A_{\text{new}}) \subseteq \text{ADOM}(A) \cup \text{IM}(m)\).
Description Logic-Based Dynamic Systems

Parametric wrt DL and progression mechanism.

DLDS

Tuple $(T, A_0, \Gamma)$, where

- $(T, A_0)$: initial DL KB $\leadsto$ during system progression: $A_0$ evolves.
- $\Gamma$: parametric actions $(\pi, \tau)$ that evolve the system. Given ABox $A$:
  - $\pi$: determines action params depending on $A$.
  - $\tau$: given $A$+param values is undef. or gives new $T$-consistent ABox.

(labeled) transition

$((\pi, \tau), m)$

A

A_{new}
Example

Registration of credit card number(s)

Calvanese, De Giacomo, Montali, Patrizi  Verification and Synthesis in DLDS  RR 2013  8 / 24
Execution Semantics

Infinite-state transition system $\Upsilon$.

- States: KBs.
- (Labeled) transitions: actions + param assignments.

Construction:

1. Start from $A_0$
2. Apply each action + param assignments in all possible ways
3. Recur over newly generated states.

Sources of infinity:

- Infinite branching - infinitely many param assignments.
- Infinite runs - usage of values obtained from action steps.
- Unbounded Aboxes - accumulation of such values.
Verification

Given a DLDS $\mathcal{K}$ and a dynamic/temporal property $\Phi$, check if $\gamma_{\mathcal{K}} \models \Phi$.

Verification logic $\mu DL_p$

Variant of first-order $\mu$-calculus.

$$\Phi ::= Q \mid \neg \Phi \mid \Phi_1 \land \Phi_2 \mid \exists x. \text{LIVE}(x) \land \Phi \mid Z \mid \mu Z.\Phi \mid \text{LIVE}(\bar{x}) \land \langle - \rangle \Phi(\bar{x}) \mid \text{LIVE}(\bar{x}) \land [\neg] \Phi(\bar{x})$$

- **Query the KBs**: any language with decidable KB query entailment.
- **Dynamics**: next-state operators and fixpoint constructs.
  - $\mu$-calculus subsumes PDL, CTL, LTL, CTL*, ....
- **Evolution of objects**: first-order quantification across states.
  - Only over objects that persist in the active domain.

Example

- $\nu Z.(\forall x. \text{Traveler}(x) \rightarrow \mu W.(\exists y. \text{booked}(y, x) \lor (\text{LIVE}(x) \land \langle - \rangle W)) \land [\neg] Z)$
- $\nu Z.(\forall x. \text{Travel}(x) \rightarrow \nu W. (\text{Processed}(x) \lor (\text{LIVE}(x) \rightarrow \langle - \rangle W)) \land [\neg] Z)$
Towards Decidability

Problem

DLDSs encode Turing machines.

\[ \Rightarrow \text{verification undecidable (even for propositional CTL/LTL)}. \]

Need to pose limitations on the DLDS itself.
Towards Decidability

**Problem**

DLDSs encode Turing machines.

→ verification **undecidable** (even for propositional CTL/LTL).

Need to pose limitations on the DLDS itself.

**State Boundedness**

Each ABox contains at most a bounded number of named individuals.

- The transition system is still infinite-state!
Towards Decidability

**Problem**

DLDSs encode Turing machines.
\[\leadsto\] verification **undecidable** (even for propositional CTL/LTL).

Need to pose limitations on the DLDS itself.

**State Boundedness**

Each ABox contains at most a bounded number of named individuals.
- The transition system is still infinite-state!

**Genericity**

Actions can only distinguish a bounded number of individuals \( C \) \((\text{ADOM}(A_0) \subseteq C')\):
- those: could affect the behavior of the actions.
- for the others: invariance of behavior (modulo renaming).
Genericity: Intuition

Travel payment

For analyzing the system (considering all possible executions):
- The actual credit card number does not matter.
- What matters is the outcome of the payment.

The process behavior:
- Distinguishes the bank status.
- Does not “see” the actual cc number
  \[\sim\] only how it relates with the other objects!
Consider action \((\pi, \tau)\).

\(h\) is a bijection that is the identity over \(C\).
Genericity

Consider action $(\pi, \tau)$.

$h$ is a bijection that is the identity over $C$. 
Consider action \((\pi, \tau)\).

\(h\) is a bijection that is the identity over \(C\).
Example

Registration of credit card number(s)

... Traveler(john)

john ⇔ bob

... Traveler(bob)

Calvanese, De Giacomo, Montali, Patrizi
Verification and Synthesis in DLDS
RR 2013 14 / 24
Example

Registration of credit card number(s)

john ⇐⇒ bob 12345 ⇐⇒ 6574

Calvanese, De Giacomo, Montali, Patrizi
Verification and Synthesis in DLDS
RR 2013
Decidability of Verification

**Theorem**

Verification of $\mu DL_p$ over a state-bounded, generic DLDS $\mathcal{K}$ is decidable. Reduced to model checking of propositional $\mu$-calculus over finite-state TS.

• # states: at most exponential in the size of $\mathcal{K}$. 

Calvanese, De Giacomo, Montali, Patrizi
Decidability of Verification

Theorem

Verification of $\mu DL_p$ over a state-bounded, generic DLDS $\mathcal{K}$ is decidable. Reduced to model checking of propositional $\mu$-calculus over finite-state TS.

- # states: at most exponential in the size of $\mathcal{K}$.

Crux of the proof (inspired by [Bagheri-Hariri et al., PODS2013])

Construction of a finite-state faithful abstraction for $\Upsilon_{\mathcal{K}}$. 

Calvanese, De Giacomo, Montali, Patrizi
Decidability of Verification

Theorem

Verification of $\mu DL_p$ over a state-bounded, generic DLDS $\mathcal{K}$ is decidable. Reduced to model checking of propositional $\mu$-calculus over finite-state TS.

- # states: at most exponential in the size of $\mathcal{K}$.

Crux of the proof (inspired by [Bagheri-Hariri et al., PODS2013])

Construction of a finite-state faithful abstraction for $\Upsilon_\mathcal{K}$.

1. Genericity $\sim$ prune infinite branching.
   - Isomorphic types instead of actual param values.
Decidability of Verification

Theorem

Verification of $\mu DL_p$ over a state-bounded, generic DLDS $\mathcal{K}$ is decidable. Reduced to model checking of propositional $\mu$-calculus over finite-state TS.

• $\#$ states: at most exponential in the size of $\mathcal{K}$.

Crux of the proof (inspired by [Bagheri-Hariri et al., PODS2013])

Construction of a finite-state faithful abstraction for $\Upsilon_{\mathcal{K}}$.

1. Genericity $\leadsto$ prune infinite branching.
   - Isomorphic types instead of actual param values.
   - Keep only one successor for every type.
Decidability of Verification

**Theorem**

Verification of $\mu DL_p$ over a state-bounded, generic DLDS $\mathcal{K}$ is decidable. Reduced to model checking of propositional $\mu$-calculus over finite-state TS.

- # states: at most **exponential** in the size of $\mathcal{K}$.

**Crux of the proof (inspired by [Bagheri-Hariri et al., PODS2013])**

Construction of a finite-state faithful abstraction for $\Upsilon_{\mathcal{K}}$.

1. Genericity $\leadsto$ prune infinite branching.
   - Isomorphic types instead of actual param values.
   - Keep only one successor for every type.

2. State boundedness + $\mu DL_p$ $\leadsto$ make runs finite.
   - Recycle old (forgotten) objects as new param values.
   - $\mu DL_p$ does not distinguish the difference.
Decidability of Verification

Theorem

Verification of $\mu DL_p$ over a state-bounded, generic DLDS $\mathcal{K}$ is decidable. Reduced to model checking of propositional $\mu$-calculus over finite-state TS.

- \# states: at most exponential in the size of $\mathcal{K}$.

Crux of the proof (inspired by [Bagheri-Hariri et al., PODS2013])

Construction of a finite-state faithful abstraction for $\Upsilon_{\mathcal{K}}$.

1. Genericity $\rightsquigarrow$ prune infinite branching.
   - Isomorphic types instead of actual param values.
   - Keep only one successor for every type.

2. State boundedness + $\mu DL_p$ $\rightsquigarrow$ make runs finite.
   - Recycle old (forgotten) objects as new param values.
   - $\mu DL_p$ does not distinguish the difference.

3. Prove that the abstraction preserves all $\mu DL_p$ formulae.
Adversarial Synthesis

DL-based two-player game structure (DL2GS)

DLDS with actions partitioned into environment and system actions.

- The environment is autonomous, we control the system.

Execution by alternation of moves.

- Move = choice of action + param assignment.
- Starting from $A_0$, repeat forever: environment moves, system replies.

Resulting TS: two-player game structure [De Giacomo et al., AAAI2010].

Basis for many synthesis problems:

- conditional planning with full-observability;
- behavior composition;
- ...
System Properties

System properties: logics a là ATL.

- Goals/desired behaviors the system wants to guarantee/realize.
- “Next Φ” ~ “system can force Φ to hold in the next state”.

Synthesis logic $\mu ADL_p$

Fragment of $\mu DL_p$: properties satisfied by system in spite of environment.

$\Phi ::= Q | \neg Q | \Phi_1 \land \Phi_2 | \Phi_1 \lor \Phi_2 | Z | \mu Z.\Phi | \nu Z.\Phi | \exists x.\text{LIVE}(x) \land \Phi | \forall x.\text{LIVE}(x) \rightarrow \Phi |$

$\text{LIVE}(\vec{x}) \land [\langle - \rangle_s \Phi(\vec{x})] | \text{LIVE}(\vec{x}) \rightarrow [\langle - \rangle_s \Phi(\vec{x})] | \text{LIVE}(\vec{x}) \land [\langle - \rangle_w \Phi(\vec{x})] | \text{LIVE}(\vec{x}) \rightarrow [\langle - \rangle_w \Phi(\vec{x})]$

Idea: $[\langle - \rangle] \Phi = [-]$ for every environment move, $\langle - \rangle$ system can lead to $\Phi$.

But does the environment persist the quantified objects? Two possibilities:

- $[\langle - \rangle]_s \Phi = $ for every environment move: environment persists $\vec{x}$, system replies leading to $\Phi$.
- $[\langle - \rangle]_w \Phi = $ for every environment move that persist $\vec{x}$: system replies leading to $\Phi$.

In addition: system must persist $(\text{LIVE}(\vec{x}) \land \ldots)$ or can drop $(\text{LIVE}(\vec{x}) \rightarrow \ldots)$ $\vec{x}$. 
Existence vs Synthesis of Winning Strategies

**Theorem**
Verification of $\mu ADL_p$ properties over a state-bounded, generic DLDS $\mathcal{K}$ is decidable.

**Proof**
$\mu ADL_p$ is a fragment of $\mu DL_p$.

Strategy: given a history of moves, tells the system what to do next.

**Existence of strategies**
Given a $\mu ADL_p$ property $\Phi$, if $\Upsilon_{\mathcal{K}} \models \Phi$ then

there exists a winning strategy for the system to force $\Phi$.

But which???
Strategies in the Finite-State Setting

Synthesis via model checking.

- Remember: the property alternates $\neg$ and $\langle \neg \rangle$ moves over next states.

1. Model check the property against the system.
2. Property not satisfied $\not\models \Phi$ no strategy.
Strategies in the Finite-State Setting

Synthesis via model checking.

- Remember: the property alternates $\neg$ and $\langle \neg \rangle$ moves over next states.

1. Model check the property against the system.
2. Property not satisfied $\not\models$ no strategy.
   Property satisfied $\models$ model checker returns a witness.
     - Labeling of the states.
3. See the labeling as a strategy.
Strategies in the Infinite-State Setting

Theorem

Given a state-bounded, generic DL2GS $\mathcal{K}$ and a $\mu ADL_p$ formula $\Phi$, if $\Upsilon_{\mathcal{K}} \models \Phi$ then there exists an algorithm that realizes a concrete strategy for the system to force $\Phi$.

Proof idea

- Design-time:
  1. Construct the finite-state, faithful abstraction for $\Upsilon_{\mathcal{K}}$.
  2. Extract an abstract strategy.
     - Corresponds to an infinite family of concrete strategies.
     - Cannot be concretized at design-time.

- Run-time: system lazily lifts the abstract strategy into a concrete one.
Lazy Strategy Concretization

System: repeat forever
Lazy Strategy Concretization

System: repeat forever

1. Environment moves
   \[\sim\] update concrete history.

faithful abstraction

concrete TS

Calvanese, De Giacomo, Montali, Patrizi
Lazy Strategy Concretization

System: repeat forever

1. Environment moves
   $\leadsto$ update concrete history.

2. Match concrete with abstract history.

Calvanese, De Giacomo, Montali, Patrizi
Verification and Synthesis in DLDS
RR 2013 21 / 24
Lazy Strategy Concretization

System: repeat forever

1. Environment moves
   \( \leadsto \) update concrete history.

2. Match concrete with abstract history.

3. Apply abstract strategy
   \( \leadsto \) get action + abstract params.

faithful abstraction

concrete TS
Lazy Strategy Concretization

System: repeat forever

1. Environment moves
   \[\leadsto\text{update concrete history.}\]

2. Match concrete with abstract history.

3. Apply abstract strategy
   \[\leadsto\text{get action + abstract params.}\]

4. Choose concrete params respecting the isomorphic type of abstract params
   \[\leadsto\text{move.}\]
Instantiating the Framework

Decidability results carry over any system that can be expressed as a generic DLDS (under state-boundedness).

Example: Knowledge and Action Bases [Bagheri Hariri et al., JAIR2013].

- Specify actions by means of effect rules relating queries on the current state with facts asserted in the next state.
- Effects can call external services to incorporate new values.

KABs can be seen as generic DLDSs.

- With sufficient syntactic conditions to check state-boundedness (see [Bagheri Hariri et al., JAIR2013-PODS2013]).
# Instantiating the Framework on KABs

## Complexity of verification/synthesis for state-bounded KABs

<table>
<thead>
<tr>
<th></th>
<th>Instance Checking</th>
<th>Epistemic CQs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DL-Lite</strong></td>
<td><strong>ExpTime</strong></td>
<td><strong>ExpTime</strong></td>
</tr>
<tr>
<td><strong>ALCQI</strong></td>
<td><strong>ExpTime</strong></td>
<td><strong>2ExpTime</strong></td>
</tr>
</tbody>
</table>

**Note**

Complexity can be refined by isolating “changeable memory slots”.

- Process can only change those.
- See relational transducers, register automata, . . .

**Exponentiality is in the number of changeable slots!**
Conclusions

• Our work is grounded in real-world data-aware processes.
• We study robust conditions for decidability.
  ▶ no conditions on the structure of the ontology;
  ▶ the extensional knowledge changes over time;
  ▶ suitable restrictions are posed on the actions.
• Complexity wise, our techniques are exponential in the size of the initial ABox / changeable slots.
• Most often processes change only a small portion of the entire data.
  ▶ Logarithmic? 😊
• Next steps:
  ▶ formalize this intuition, moving towards practical implementation;
  ▶ investigate the connection with concrete instantiations;
  ▶ study other forms of abstraction.