Reasoning over Evolving Graph-structured Data Under Constraints

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

Based on joint work with: S. Ahmetaj, M. Ortiz, M. Šimkus

Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2016)
Panama City, Panama, June 6–10, 2016
Outline

1. Description Logics for Graph-structured Data
2. Reasoning in Dynamic Systems
3. Description Logics for Evolving Graph Structured Data
4. Planning
5. Conclusions
Motivations for this research

Comes from two important “trends” in data and information management:

1. Graph databases [Mendelzon and Wood 1995], aka graph-structured data (GSD).

2. Dealing with dynamic systems, while properly taking into account data.

What we are going to do here:

- We argue that research in knowledge representation has provided important contributions to both settings.

- We combine the two aspects in a novel setting relying on constraints expressed in Description Logics (DLs) for managing evolving GSD.
Outline

1. Description Logics for Graph-structured Data
2. Reasoning in Dynamic Systems
3. Description Logics for Evolving Graph Structured Data
4. Planning
5. Conclusions
Graph-structured data are everywhere

The data underlying many settings is inherently graph structured:

- Web data
- Social data
- RDF data
- Open linked data
- XML data
- Pointer structure in a program

We need formalisms, techniques, and tools to properly manage GSD:

- **modeling languages and constraints**
- query languages
- efficient query answering
- **dealing with evolving GSD**
Graph-structured data is not really new

A graph-structured database instance
An edge and node labeled graph
A finite relational structure with unary and binary relations only
A \textbf{finite} Description Logic interpretation

Example:

[Diagram of a graph-structured database instance]

- Dept
- KRDB-RC
- KR
- EMCL
- ODBS
- Program
- Module
- FacMember
- Paul
- Course
- offers
- teaches
- requires
- partOf
The problem of specifying and reasoning over integrity constraints for GSD has been addressed in the database community.

Path constraints for GSD

Path constraints [Abiteboul and Vianu 1999; Buneman, Fan, and Weinstein 2000; Grahne and Thomo 2003]

- Make use of regular expressions $P$, interpreted over GSD instances $\mathcal{I}$:
  
  $$\llbracket P \rrbracket^\mathcal{I} = \text{set of pairs of nodes connected in } \mathcal{I} \text{ by a path}
  \text{ whose sequence of labels is a word in the language of } P.$$  

- Path constraints $\varphi$ come in two forms:
  
  $$P_\ell \subseteq P_r$$
  $$[P_p](P_\ell \subseteq P_r)$$

- Semantics: set of nodes satisfying the constraint
  
  $$\llbracket P_\ell \subseteq P_r \rrbracket^\mathcal{I} = \{ n \mid \text{if } (n, n') \in \llbracket P_\ell \rrbracket^\mathcal{I} \text{ then } (n, n') \in \llbracket P_r \rrbracket^\mathcal{I}, \text{ for all } n' \}$$
  $$\llbracket [P_p](P_\ell \subseteq P_r) \rrbracket^\mathcal{I} = \{ n \mid \text{for all } n_1 \text{ s.t. } (n, n_1) \in \llbracket P_p \rrbracket^\mathcal{I}, \text{ if } (n_1, n') \in \llbracket P_\ell \rrbracket^\mathcal{I} \text{ then } (n_1, n') \in \llbracket P_r \rrbracket^\mathcal{I}, \text{ for all } n' \}$$
Reasoning with path constraints

- **Global** semantics: \( \mathcal{I} \models \varphi \), if every node is in \( [\varphi]^\mathcal{I} \).

- **Pointed** semantics: \( \mathcal{I}, a \models \varphi \), if \( a \in [\varphi]^\mathcal{I} \), where \( a \) is some given node.

**Central problem:** implication of path constraints

Given a set \( \Gamma \) of path constraints, and a path constraint \( \varphi \) decide:

- **Unrestricted implication:** Does \( \Gamma \models \varphi \)?
  
  I.e., for every \( \mathcal{I} \), whenever \( \mathcal{I} \models \Gamma \) then also \( \mathcal{I} \models \varphi \).

- **Finite implication:** Does \( \Gamma \models_{\text{fin}} \varphi \)?
  
  Same as above, but over finite instances.

Similarly for unrestricted and finite implication under pointed semantics.
Implication of path constraint is undecidable

Finite and unrestricted **implication** of path constraints was shown **undecidable**
[Buneman, Fan, and Weinstein 2000; Grahne and Thomo 2003]:

- for pointed semantics, and general constraints \([P_p](P_\ell \subseteq P_r)\)
- for global semantics, even for prefix-empty, word constraints \(w_\ell \subseteq w_r\)

\[\sim \text{Decidability requires both pointed semantics and empty prefixes.}\]

Recently, **undecidability has been tightened** to rather simple (word) constraints, of the forms [C., Ortiz, and Simkus 2016]:

\[ [r](r_1 \circ r_2 \subseteq r_3) \quad [r](r_1 \subseteq r_2 \circ r_3) \quad \text{(for both semantics)} \]

or:
\[ r_1 \circ r_2 \subseteq r_3 \quad r_1 \subseteq r_2 \circ r_3 \quad \text{(for global semantics)} \]

where all \(r\) are **simple labels** (i.e., no \(\varepsilon\), no inverse labels).
Impact on inference over TGDs

The previous result can easily be rephrased in terms of tuple-generating dependencies (TGDs):

1. \( r_1 \circ r_2 \subseteq r_3 \) is equivalent to \( r_1(x, y), r_2(y, z) \rightarrow r_3(x, y) \)
2. \( r_1 \subseteq r_2 \circ r_3 \) is equivalent to \( r_1(x, y) \rightarrow \exists z. r_2(x, z), r_3(z, y) \)

Undecidability of TGD entailment and of query answering under TGDs

(Finite) entailment of TGDs, and (finite) entailment of atomic queries under TGDs are undecidable already for TGDs of the forms:

\( r_1(x, y), r_2(y, z) \rightarrow r_3(x, y) \)
\( r_1(x, y) \rightarrow \exists z. r_2(x, z), r_3(z, y) \)
Expressive DLs for constraints on GSD

Expressive DLs are well suited to express constraints on GSD:

- powerful features for structuring the domain into classes (i.e., concepts)
- complex conditions for typing binary relations (i.e., roles)
- when resorting to expressive DLs with regular expressions over roles, we also have a mechanism to navigate the graph

Let us consider one such DL: $ALCOIb_{reg}$, also known as $ZOI$.

$ZOI$ is closely related to (positive) regular XPath with nominals [Cate and Segoufin 2008; C., De Giacomo, Lenzerini, et al. 2009] and Propositional Dynamic Logic [Fischer and Ladner 1979].
The DL $\mathcal{ZOI}$

- The vocabulary of $\mathcal{ZOI}$ has three alphabets:
  - $N_C$: concept names, or node symbols – denote unary predicates
  - $N_R$: role names, or edge symbols – denote binary predicates
  - $N_I$: individuals, or node names – denote constants

  **Note:** each node and edge can be labeled with a set of symbols.

- **Concepts**, i.e., node formulas
  
  \[
  C, C' \rightarrow A \mid \{a\} \mid \neg C \mid C \cap C' \mid C \cup C' \mid \forall P.C \mid \exists P.C
  \]

  $A \in N_C$, $a \in N_I$

- **Roles**, i.e., path formulas
  
  \[
  S, S' \rightarrow r \mid r^- \mid \{(a, b)\} \mid S \cap S' \mid S \cup S' \mid S \setminus S'
  \\
  P, P' \rightarrow \nabla \mid \varepsilon \mid id(C) \mid S \mid P \cup P' \mid P \circ P' \mid P^*
  \]

  $r \in N_R$, $a, b \in N_I$, $S$: simple role
Formulas in $\mathcal{ZOI}$

An **atomic formula** $\alpha$ of $\mathcal{ZOI}$ corresponds to a TBox or ABox assertion:

- **Inclusions** between concepts and between simple roles:
  
  \[ C_1 \sqsubseteq C_2 \quad \text{and} \quad S_1 \sqsubseteq S_2 \]

- **Assertions** on concepts and on simple roles
  
  \[ C(a) \quad \text{and} \quad S(a, b) \]

A **$\mathcal{ZOI}$ knowledge base** is a boolean combination of atomic formulas:

\[ \mathcal{K} \rightarrow \alpha \mid \mathcal{K} \land \mathcal{K}' \mid \mathcal{K} \lor \mathcal{K}' \mid \neg \mathcal{K} \]
Semantics of $\mathcal{ZOI}$

We have the standard DL semantics, based on (finite) FO interpretations.

- **Roles** (i.e., paths) are interpreted as **binary relations**:
  - Regular expressions are analogous to those in path constraints.
  - Inverse role $r^-$ denotes the inverse of the binary relation denoted by $r$.
  - Also: $\varepsilon^\mathcal{I} = \{(o, o) \mid o \in \Delta^\mathcal{I}\}$
    \[ \nabla^\mathcal{I} = \Delta^\mathcal{I} \times \Delta^\mathcal{I} \]
    \[ \{(a, b)^\mathcal{I} = \{(a^\mathcal{I}, b^\mathcal{I})\} \]

- **Concepts** are interpreted as **unary relations** (i.e., sets of objects):
  - The boolean operators $\sqcap$, $\sqcup$, $\neg$ are as usual.
  - Nominal $\{a\}$ denotes a singleton, i.e., $\{a\}^\mathcal{I} = \{a^\mathcal{I}\}$.
  - $\exists P.C$ denotes the starting points of a $P$-path ending in (an instance of) $C$.
  - $\forall P.C$ denotes the objects for which all $P$-paths starting there end in $C$.

- **Inclusions** are interpreted as **implications** (i.e., as set inclusion):
  - $\mathcal{I}$ satisfies $C_1 \subseteq C_2$ if $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$.
  - Analogously for roles.

- **Assertions** $C(a)$ and $S(a, b)$ are analogous to facts, but can make use of complex concept and role expressions.

- **Booleans in a KB** have the usual meaning.
Example of constraints expressible in $ZOI$

Complex domain and range restrictions

- $\exists$offers.Course $\sqsubseteq$ Dept
- $\exists$requires.$\top$ $\sqsubseteq$ Program
- Department $\sqsubseteq$ $\forall$offers.(Program $\sqcup$ Course)
- $\top$ $\sqsubseteq$ $\forall$requires.$(\exists$partOf$^\ast$.Course)$

Conditions requiring navigation on the graph

- Course $\sqsubseteq$ $\exists$taughtBy.FacMember
- $\exists$(partOf$^\ast$ $\circ$ requires).Program $\sqsubseteq$ $\exists$offers$^\ast$.Department
- Course $\sqcap$ $\exists$requires$^\ast$.UndergradProgram $\sqsubseteq$
- $\exists$teaches$^\ast$.($\exists$(memberOf $\circ$ partOf$^\ast$).Institute)$
Expressing path constraints in $\mathcal{ZOI}$

We can encode path constraints in $\mathcal{ZOI}$, for empty prefixes under pointed semantics (for individual $a$):

$$\varphi = P_\ell \subseteq P_r \quad \leadsto \quad \mathcal{T}^a_\varphi = \{a\} \sqsubseteq \forall P_\ell. \exists \text{inv}(P_r).\{a\}$$

(where $\text{inv}(P_r)$ denotes the role representing the inverse of path $P_r$)

**Lemma**

Let $\Gamma$ be a set of prefix-empty constraints, $\varphi$ a prefix-empty constraint, and $a$ an individual. Then:

$$\Gamma, a \models_{(\text{fin})} \varphi \quad \text{iff} \quad \left( \bigwedge_{\gamma \in \Gamma} \mathcal{T}^a_\gamma \right) \land \neg \mathcal{T}^a_\varphi \quad \text{is not (finitely) satisfiable}$$
Complexity of path-constraint implication

Satisfiability of \( \mathcal{ZOI} \) is \textbf{ExpTime-complete}. From this we get:

**Theorem ([C., Ortiz, and Simkus 2016])**

Implication of prefix-empty path constraints under pointed semantics is decidable in \textbf{ExpTime}

*Previous known bound:* \textbf{N2ExpTime}

What about finite implication?

- Finite model reasoning for \( \mathcal{ZOI} \) has not been considered so far.
- However, it turns out that \( \mathcal{ZOI} \) has the \textbf{finite model property}. Proof needs ideas from PDL and from 2-variable fragment.

**Theorem ([C., Ortiz, and Simkus 2016])**

Finite implication of prefix-empty path constraints under pointed semantics is decidable in \textbf{ExpTime}
Other classes of path constraints

For empty prefixes under pointed semantics, we have used a nominal to encode the inclusion of the left-tail in the right-tail. This does not work

- under global semantic, or
- in the presence of a prefix.

To express other path constraints, we need to extend the logic:

We can capture all forms of path constraints in \( \mathcal{ZOT} \) extended with role difference for arbitrary (non-simple) roles:

\[
\varphi = [P_p](P_\ell \subseteq P_r) \quad \leadsto \quad C_\varphi = \forall P_p.(\forall (P_\ell \setminus P_r).\bot)
\]

Lemma

Let \( \Gamma \) be a set of constraints, \( \varphi \) a constraint, and \( a \) an individual. Then:

\[
\begin{align*}
\Gamma, a \models (\text{fin}) \varphi & \iff \left( \bigcap_{\gamma \in \Gamma} C_\gamma \cap \neg C_\varphi \right)(a) \text{ is not (finitely) satisfiable,} \\
\Gamma \models (\text{fin}) \varphi & \iff \vdash \left( \bigcap_{\gamma \in \Gamma} C_\gamma \subseteq C_\varphi \right) \text{ is not (finitely) satisfiable}
\end{align*}
\]
Outline

1. Description Logics for Graph-structured Data
2. Reasoning in Dynamic Systems
3. Description Logics for Evolving Graph Structured Data
4. Planning
5. Conclusions
Dynamic systems taking into account data

Traditional approach to model dynamic systems: divide et impera of
- static, data-related aspects
- dynamic, process/interaction-related aspects

These two aspects traditionally treated separately by different communities:
- Data management community:
  data modeling, constraints, analysis deal mostly with static aspects
- (Business) process management and verification community:
  data is abstracted away
Reasoning about evolving data and knowledge

However, the knowledge representation community traditionally has paid attention to the combination of static and dynamic aspects:

- The combination in a single logical theory is well-known to be difficult [Wolter and Zakharyaschev 1999; Gabbay et al. 2003]
- Reasoning about actions in the Situation Calculus, cf. [Reiter 2001]
- Knowledge and Action Bases [Bagheri Hariri, C., Montali, et al. 2013]
Reasoning about evolving data

Actually, there is quite some work also coming from the database community:

- Dynamic relational model [Vianu 1983, 1984]
- Temporal deductive databases [Snodgrass 1984; Chomicki and Imielinski 1988]
- Relational and ASM transducers [Abiteboul, Vianu, et al. 1998; Spielmann 2000]
- Data-driven web systems [Deutsch, Sui, and Vianu 2004]
- Business Artifacts [Nigam and Caswell 2003; Bhattacharya et al. 2007]
- Active XML [Abiteboul, Benjelloun, and Milo 2004]
- Artifact systems with arithmetic [Damaggio, Deutsch, and Vianu 2012]
Relevant assumptions about the system behaviour

In the dynamic setting, there is a huge variety of different assumptions made, that deeply affect the inference services of interest and their computational properties:

1. System dynamics specified procedurally (e.g., through a finite state machine) vs. declaratively (e.g., through a set of condition-action rules).

2. Simple vs. complex actions.

3. Actions operate on the single instances (i.e., models), as opposed to adopting the functional approach [Levesque 1984].

4. Completely specified initial state vs. incomplete initial state.

5. Deterministic vs. non-deterministic effects of actions.

6. During system execution, new objects may enter the system or not.

7. The intensional knowledge about the system is fixed vs. changes.
The setting we adopt here

In our setting, we specialize the above options as follows:

1. We assume to have available a finite set of parametric actions.
2. Actions might be complex, and allow for checking conditions.
3. Actions operate on the single instances.
4. We assume incomplete information in the initial state, i.e., we are interested in reasoning over all possible initial states compliant with the incomplete specification.
5. Our actions are deterministic.
6. Our actions do not incorporate new objects in the system . . . but (when relevant) we allow for arbitrarily extending the domain in the initial state.
7. The intensional knowledge might change, since it is affected in complex ways by the extensional knowledge.
Reasoning services of interest

We consider several classical reasoning services that are of relevance in this setting:

- **Verification.**
- Variants of planning:
  - Existence of a plan.
  - Existence of a plan from a given precondition.
  - Conformant planning.
- Variants of bounded planning, i.e., we impose a priori finite bounds on the length or domain of the plan.
Applying an action to a finite DB instance

Let $\mathcal{K}$ be a KB, $\mathcal{I}$ a finite DB instance for $\mathcal{K}$, and $\alpha$ a (possibly complex) action. Then $\alpha(\mathcal{I})$ denotes the DB instance obtained by applying $\alpha$ to $\mathcal{I}$.

Verification (V) problem

Given KB $\mathcal{K}$ and action $\alpha$, is $\alpha$ $\mathcal{K}$-preserving?

I.e., is it the case that, for every finite DB instance $\mathcal{I}$, if $\mathcal{I} \models \mathcal{K}$ then $\alpha(\mathcal{I}) \models \mathcal{K}$?
Outline

1. Description Logics for Graph-structured Data
2. Reasoning in Dynamic Systems
3. Description Logics for Evolving Graph Structured Data
4. Planning
5. Conclusions
To make the framework concrete, we consider an update language for GSD that allows for various types of actions:

- **Adding** the result of a concept to an atomic concept.
- **Adding** the result of a role to an atomic role.
- **Removing** the result of a concept from an atomic concept.
- **Removing** the result of a role from an atomic role.
- **Conditional execution / composition / parameters.**
Example

A complex action with input parameters $x, y, z$ that transfers an employee $x$ from a project $y$ to the project $z$:

$$\alpha = (\text{Employee}(x) \land \text{Project}(y) \land \text{Project}(z) \land \text{worksFor}(x, y)) \ ?$$

$$\text{worksFor} \ominus \{(x, y)\} \cdot \text{worksFor} \oplus \{(x, z)\} : \varepsilon$$

- $\alpha$ checks if $x$ is an Employee, $y$ and $z$ are Projects, and $x$ worksFor $y$.
- If yes, it removes the worksFor link between $x$ and $y$, and creates a worksFor link between $x$ and $z$.
- If no (i.e., any of the checks in the conjunction fails), it does nothing.
Result of conditional action – Example

Before being executed, the action in grounded.

Example of execution of a grounded action:

Given:

\[ \alpha = (\text{Employee}(e) \land \text{Project}(p_1) \land \text{Project}(p_2) \land \text{worksFor}(e, p_1)) \?
\]

\[ \text{worksFor} \oplus \{(e, p_1)\} \cdot \text{worksFor} \oplus \{(e, p_2)\} : \varepsilon \]

\[ \mathcal{I} = \{ \text{Employee}(e), \text{worksFor}(e, p_1), \text{Project}(p_1), \text{Project}(p_2) \} \]

Result:

\[ \alpha(\mathcal{I}) = \{ \text{Employee}(e), \text{worksFor}(e, p_2), \text{Project}(p_1), \text{Project}(p_2) \} \]

**Recall:** We use \( \alpha(\mathcal{I}) \) to denote the result of applying \( \alpha \) to \( \mathcal{I} \).
Solving the verification problem

The verification problem can be reduced to finite (un)satisfiability of a \( \mathcal{ZO1} \) KB using a form of regression.

Let \( \mathcal{K}_{L \leftarrow L'} \) be the KB obtained from \( \mathcal{K} \) by replacing each occurrence of \( L \) by \( L' \).

Transformation \( \text{TR}(\mathcal{K}, \alpha) \) of a KB \( \mathcal{K} \) via an action \( \alpha \) is defined inductively:

\[
\begin{align*}
\text{TR}(\mathcal{K}, \epsilon) &= \mathcal{K} \\
\text{TR}(\mathcal{K}, (A \oplus C) \cdot \alpha) &= (\text{TR}(\mathcal{K}, \alpha))_{A \leftarrow A \triangle C} \\
\text{TR}(\mathcal{K}, (A \ominus C) \cdot \alpha) &= (\text{TR}(\mathcal{K}, \alpha))_{A \leftarrow A \cap \neg C} \\
\text{TR}(\mathcal{K}, (r \oplus P) \cdot \alpha) &= (\text{TR}(\mathcal{K}, \alpha))_{r \leftarrow r \cup P} \\
\text{TR}(\mathcal{K}, (r \ominus P) \cdot \alpha) &= (\text{TR}(\mathcal{K}, \alpha))_{r \leftarrow r \setminus P} \\
\text{TR}(\mathcal{K}, (\mathcal{K}_1 ? \alpha_1 : \alpha_2)) &= (\neg \mathcal{K}_1 \lor \text{TR}(\mathcal{K}, \alpha_1)) \land (\mathcal{K}_1 \lor \text{TR}(\mathcal{K}, \alpha_2))
\end{align*}
\]
Transforming a KB via an action – Example

Example

\[ K_1 = (\text{Project} \sqsubseteq \text{ActiveProject} \sqcup \text{ConcludedProject}) \land \\
(\text{Employee} \sqsubseteq \text{ProjectEmployee} \sqcup \text{PermanentEmployee}) \land \\
(\exists \text{worksFor}. \text{Project} \sqsubseteq \text{ProjectEmployee}) \]

\[ \alpha_1 = \text{ActiveProject} \ominus \{\text{optique}\} \cdot \\
\text{ConcludedProject} \oplus \{\text{optique}\} \cdot \\
\text{ProjectEmployee} \ominus \exists \text{worksFor}. \{\text{optique}\} \]

\[ \text{TR}(K_1, \alpha_1) = \\
(\text{Project} \sqsubseteq (\text{ActiveProject} \sqcap \neg \{\text{optique}\}) \\
\sqcup (\text{ConcludedProject} \sqcup \{\text{optique}\})) \land \\
(\text{Employee} \sqsubseteq (\text{ProjectEmployee} \sqcap \neg \exists \text{worksFor}. \{\text{optique}\}) \\
\sqcup \text{PermanentEmployee}) \land \\
(\exists \text{worksFor}. \text{Project} \sqsubseteq (\text{ProjectEmployee} \sqcap \neg \exists \text{worksFor}. \{\text{optique}\})) \]
Reducing verification to unsatisfiability

For a ground action $\alpha$ and a KB $\mathcal{K}$, the transformation $\text{TR}(\mathcal{K}, \alpha)$ correctly captures the meaning of $\alpha$.

Lemma

For every ground action $\alpha$ and DB instance $\mathcal{I}$:

$$\alpha(\mathcal{I}) \models \mathcal{K} \iff \mathcal{I} \models \text{TR}(\mathcal{K}, \alpha).$$

Theorem

For every action $\alpha$ and KB $\mathcal{K}$

$$\alpha \text{ is } \mathcal{K}\text{-preserving} \iff \mathcal{K} \land \not\exists \alpha_g \text{ is finitely unsatisfiable}$$

where $\alpha_g$ is obtained from $\alpha$ by replacing each variable with a fresh individual name not occurring in $\alpha$ and $\mathcal{K}$. 
Deciding verification

In order to obtain from the previous result decidability of verification, we need to ensure that $\text{TR}(\mathcal{K}, \alpha_g)$ is expressible in $\mathcal{ZOI}$.

Key issue: form of basic actions: $(A \oplus C)$, $(A \ominus C)$, $(r \oplus P)$, $(r \ominus P)$

- We can allow for arbitrary concepts $C$ to be added and removed via $(A \oplus C)$ and $(A \ominus C)$.
- Instead, in $(r \oplus P)$ and $(r \ominus P)$, the role $P$ must be simple: i.e., a role name, inverse role name, $\{(a, b)\}$, and their boolean combination, but no concatenation or transitive closure.

Complex actions containing these restricted basic actions are called role-simple.

Examples of role-simple actions:

friendOf \(\ominus\) ( hasAunt \(\cap\) sendsCandyCrushInvitation\(^-\) )

friendOf \(\ominus\) ( supports\(\downarrow\)\{Berlusconi\} )

preferredAICollaborators \(\oplus\) \(\exists\) ( collaboratesWith\(\downarrow\neg\exists\)projWith.\{Darpa\} \(\ast\).ExpertAI
Complexity of verification

**Theorem**

For $ZOI$ KBs and role-simple actions, verification is \text{ExpTime}-complete.

- The lower bound follows from the fact that a KB $\mathcal{K}$ is finitely satisfiable iff $(A' \oplus \{o\})$ is not $(\mathcal{K} \land (A \sqsubseteq \neg A') \land (A(o)))$-preserving, where $A$, $A'$, and $o$ are fresh.

- For the upper bound:
  - Observe that the KB $\text{TR}(\mathcal{K}, \alpha)$ might be exponential in $\alpha$, since conditional actions lead to duplication of $\mathcal{K}$.
  - However, the resulting KB can be put in disjunctive normal form, with exponentially many conjunctions of atoms, each of polynomial size.
  - Hence, once can run an exponential number of checks on polynomial-size KBs, each of which takes at most exponential time.
  - The resulting algorithm runs in single exponential time.
Complexity of verification

When actions are not role-simple, i.e., contain role concatenation, or transitive closure, verification becomes undecidable.

**Theorem**

Deciding whether $\alpha$ is $\mathcal{K}$-preserving is **undecidable**, even when

- $\mathcal{K}$ consists of a single fact $r(a,b)$, and
- $\alpha$ is just a sequence of basic actions of the form

\[
(r \oplus P) \quad (r \ominus P)
\]

with $P$ a sequence of one or two symbols.

The results relies on the undecidability of implication of path constraints of the simple form seen before.
Lightweight DLs

To simplify verification, we consider a restricted setting based on lightweight DLs of the *DL-Lite* family.

- *DL-Lite* is a family of DLs introduced for the purpose of accessing data through ontologies.
- This family provides a good foundation for ontology-based data access.

### Standard $DL-Lite_R$ KBs

- Roles $P$ are names $r$ or inverse roles $r^-$. 
- Concepts $B$ are: names $A$, or 
  - the projection $\exists P$ of role $P$ on the first component, or 
  - the projection $\exists P^-$ of role $P$ on the second component.
- In a KB, we can state: inclusions between concepts and roles, disjointness between concepts and roles ABox assertions $B(a)$ and $P(a, b)$. 
A restricted setting based on *DL-Lite*

We consider a generalization of *DL-Lite*$_{R}$ KBs:

A *DL-Lite*$_{R}^{+}$ KB is a KB satisfying the following conditions:

- Concept and role inclusions and disjointness are as in standard *DL-Lite*$_{R}$.
- In concept assertions $C(a)$, the concept $C$ might be a boolean combination of concept names $A$, projections $\exists P$, and nominals $\{a'\}$.
- $\neg$ may occur only in front of ABox assertions (while $\land$ and $\lor$ may be applied freely on KBs).

We need to restrict also the form of actions:

**Localized actions**

A localized action is one where in a conditional action $K?\alpha_1:\alpha_2$, the KB $K$ is a boolean combination of ABox assertions (hence, it may not contain concept or role inclusions or disjointness).
Verification for $DL$-$Lite_{R}^{+}$ KBs and localized actions can be reduced in linear time to finite unsatisfiability of $DL$-$Lite_{R}^{+}$ KBs.

**Theorem**

Verification for $DL$-$Lite_{R}^{+}$ KBs and localized actions can be reduced in linear time to finite unsatisfiability of $DL$-$Lite_{R}^{+}$ KBs.

**Intuition:**

1. Construct as before $\mathcal{K}' = \mathcal{K} \land \neg \text{TR}(\mathcal{K}, \alpha_g)$.
2. Push $\neg$ inside so that it occurs in front of inclusions and assertions only.
3. Replace each $\neg (B_1 \sqsubseteq B_2)$ by $(B_1 \sqcap \neg B_2)(o)$, where $o$ is fresh, and each $\neg (r_1 \sqsubseteq r_2)$ by $(r_1 \setminus r_2)(o, o')$, where $o, o'$ are fresh.

We obtain a $DL$-$Lite_{R}^{+}$ KB that we can check for unsatisfiability.
Complexity of verification in the *DL-Lite* setting

**Theorem**

Finite satisfiability of $DL-Lite^{+}_R$ KBs is NP-complete.

- NP-hardness is immediate.
- Membership in NP: we define a non-deterministic polynomial time rewriting procedure that transforms a $DL-Lite^{+}_R$ KB $\mathcal{K}$ into a $DL-Lite_R$ KB $\mathcal{K}'$, s.t., $\mathcal{K}$ is satisfiable iff there exists a $\mathcal{K}'$ that is satisfiable.

**Theorem**

Verification for $DL-Lite^{+}_R$ KBs and localized actions is coNP-complete.
Intractability in a very restricted setting

coNP-hardness does **not** depend on intractability of $DL$-$L$ite$_R^+$!

**Theorem**

Verification is coNP-hard already when:

- KBs consist of a conjunction of concept disjointness assertions:
  
  $$(A_0 \sqsubseteq \neg A'_0) \land \cdots \land (A_n \sqsubseteq \neg A'_n),$$

- actions are localized ground sequences of basic actions of the forms
  
  $$(A \oplus C)$$ and $$(A \ominus C).$$

The proof is by a reduction of non-3-colorability.
Outline

1. Description Logics for Graph-structured Data
2. Reasoning in Dynamic Systems
3. Description Logics for Evolving Graph Structured Data
4. Planning
5. Conclusions
Reasoning services – Planning

We are given:
- a KB $\mathcal{K}$,
- a finite DB instance $\mathcal{I}$ for $\mathcal{K}$, and
- a finite set $\text{Act}$ of actions.

Plan

A finite sequence $\alpha_1 \circ \cdots \circ \alpha_n$ of actions in $\text{Act}$ is a plan (of length $n$) for $\mathcal{K}$ from $\mathcal{I}$, if there exists a finite set $\Delta$ of objects such that

$$(\alpha_1 \circ \cdots \circ \alpha_n)(\mathcal{I}^\prime) \models \mathcal{K},$$

where $\mathcal{I}^\prime$ is identical to $\mathcal{I}$, except that the domain is extended by $\Delta$.

Note: extending the DB domain, account for new objects that might be needed in the plan to satisfy the goal constraints in $\mathcal{K}$.

Planning (P) problem and Domain Bounded Planning (PDb) problem

- Given $\mathcal{K}$, $\text{Act}$, and $\mathcal{I}$, does there exist a plan for $\mathcal{K}$ from $\mathcal{I}$.
- Given $\mathcal{K}$, $\text{Act}$, $\mathcal{I}$, and a bound $k$, does there exist a plan for $\mathcal{K}$ from $\mathcal{I}$ where $|\Delta|$ is at most $k$. 
In this variant of planning, we are not given the initial DB instance, but want to check plan existence from some DB instance satisfying a \textit{given precondition}. 

**Planning Under Incompleteness (PI) and Length Bounded Planning Under Incompleteness (PILb)**

- Given $\text{Act}$, $I$, $\mathcal{K}$, and $\mathcal{K}_{pre}$, does there exist a plan for $\mathcal{K}$ from $I$, for some finite DB instance $I$ such that $I \models \mathcal{K}_{pre}$.

- Given $\text{Act}$, $I$, $\mathcal{K}$, $\mathcal{K}_{pre}$, and a bound $\ell$, does there exist a plan for $\mathcal{K}$ from $I$ of length at most $\ell$, for some finite DB instance $I$ such that $I \models \mathcal{K}_{pre}$. 
Undecidability of unbounded planning

Without bounds, planning is undecidable already for the restricted $DL$-$Lite_{\mathcal{R}}^{+}$ setting.

**Theorem (Undecidability of Planning)**

Planning (P) and Planning Under Incompleteness (PI) are undecidable already for $DL$-$Lite_{\mathcal{R}}^{+}$ KBs and simple actions.

**Intuition:** we do not have a bound on the number of objects to be added to the domain to satisfy the goal KB.
Decidability of bounded planning

Planning under complete information becomes decidable if we bound the domain.

**Theorem (Decidability of Domain Bounded Planning)**

Domain Bounded Planning (PDb) is $\text{PSPACE}$-complete for $\text{ZOI}$ KBs

Interestingly Planning Under Incompleteness stays undecidable even if we bound the domain.

However, it becomes decidable by bounding the plan length.

**Theorem**

Length Bounded Planning Under Incompleteness (PILb) is

- $\text{EXPTIME}$-complete for $\text{ZOI}$ KBs, and
- $\text{NP}$-complete for $\text{DL-Lite}_R^+$ KBs and with simple actions.
Outline

1. Description Logics for Graph-structured Data
2. Reasoning in Dynamic Systems
3. Description Logics for Evolving Graph Structured Data
4. Planning
5. Conclusions
Summing up

- By exploiting techniques and tools coming from work in DLs, we obtain strong **decidability and complexity** results for reasoning about **evolving GSD under constraints**.

- This indicates that DLs are well suited not only to manage the structure of data, but also its dynamics.

- This calls for more interaction between the data management and knowledge representation communities.
Further work

- Investigate further useful fragments with lower complexity.

- Can we extend the update language while preserving decidability?
  - while loops
  - richer queries than concepts and roles

- Can we consider other forms of constraints?
  - keys
  - identification constraints
Thank you for your attention!
References


References VIII


