Planning and Change in Graph Structured Data under Description Logics Constraints

Mantas Šimkus
Joint work with Shqiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz

AMW 2014
June 5, 2014
Databases often store **important/sensitive** information

They are almost **never static**

They **evolve** due to updates

- by users
- by applications

Thus it is important to **protect** databases

**from evolving into undesired states!**
Database Updates

- Assume an application updates its database using procedures
  - `AddItem(x, y, z),`
  - `MoveCustomer(x, y, z),`
  - etc.

- Natural issues of safety arise
  - Do the procedures preserve the satisfaction of integrity constraints (ICs)?
  - Is it possible to issue a sequence of procedure calls that lead to an undesired state?
Undecidability

- No hope for **decidability** if
  - the procedures are in some general programming language
  - reasoning about the ICs is undecidable

- We need to carefully **design the language**
  - for specifying the ICs, and
  - for specifying database updates

- Maintain **balance** between expressivity and complexity

- We show that this **can be done for graph databases!**
Graph databases are important!

Graph structured data is everywhere:

- Web data
- RDF data
- XML data
- Program state (pointer structure, object references)
Graph Databases: a Definition

A graph database $\mathcal{I}$

\[ \mathcal{I} = \{ \text{Person}(Ann), \text{livesIn}(Ann, Bonn), \text{partOf}(Bonn, DE) \} \]
We advocate the use of a very expressive Description Logic called $ALCHOIQ_{br}$ (due to the naming convention:)

Most DLs can be viewed as syntactic fragments of $C^2$

- the 2-variable fragment of FOL with counting quantifiers

In this presentation, we assume ICs are expressed using $C^2$ formulae

All our DL-based results can be reformulated for $C^2$

Key property: finite satisfiability in $C^2$ is NEXPTIME-complete
What we previously called “procedures”

= 

**Actions** in our **update language**
Features of the Update Language

- **concept** = a $C^2$ formula with one free variable
- **role** = a $C^2$ formula with 2 free variables
- **Concepts/roles are unary/binary first-order queries**
- Our update language allows
  - adding the result of a concept to a unary relation
  - removing the result of a concept from a unary relation
  - adding the result of a role to a binary relation
  - removing the result of a role from a binary relation
  - conditional execution / composition / parameters / no loops!
Example (using DL notation)

A procedure with input variables $x, y, z$ that transfers an employee $x$ from a project $y$ to the project $z$:

$$
\alpha = (x : \text{Employee} \land y : \text{Project} \land z : \text{Project} \land (x, y) : \text{worksFor}) \ ? \\
\text{worksFor } \ominus \{(x, y)\} \cdot \text{worksFor } \oplus \{(x, z)\} : \varepsilon
$$

Intuition:

- Action $\alpha_2$ first checks whether $x$ 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 fails), it does nothing.
Some Notation

Given a database $\mathcal{I}$ and an action $\alpha$, 

$S(\mathcal{I}, \alpha)$ denotes the result of applying $\alpha$ on $\mathcal{I}$
### Preservation of Integrity Constraints

Assume a $C^2$ formula $\varphi$. An action $\alpha$ is **$\varphi$-preserving** if

$$ I \models \varphi \text{ implies } S(I, \alpha) \models \varphi, $$

for any input database $I$ and any values of parameters for $\alpha$.

We prove that

- deciding if $\alpha$ is $\varphi$-preserving is **coNEXPTIME-complete**
- but is **coNP-complete** for ICs expressed in **DL-Lite**
Beyond Preservation of Integrity Constraints

- Knowing that individual actions preserve ICs is sometimes not good enough!
- **Sequences of actions** may lead to database states that satisfy the ICs, yet are undesired
- E.g. it should not be possible to create a sequence of actions that makes a **concluded** project **active** again
- The above and related problems can be formalized by means of **automated planning**
Planning in Graph Databases

Definition

Let $\mathcal{I}$ be a graph database,

$\mathit{Act}$ a set of actions, and

$\varphi$ a goal, expressed in $\mathcal{C}^2$.

A sequence $\langle \alpha_1, \ldots, \alpha_n \rangle$ of actions from $\mathit{Act}$ is called a plan for $\varphi$ from $\mathcal{I}$, if $S(\mathcal{I}, \alpha_1 \cdots \alpha_n) \models \varphi$. 
### Planning in Graph Databases

#### Definition

- **(P1)** Given a database $\mathcal{I}$, a set of actions $Act$, and a goal $\phi$, does there exist a plan for $\phi$ from $\mathcal{I}$?

- **(P2)** Given a set of actions $Act$, a precondition formula $\psi$, and a goal $\phi$, does there exist a plan for $\phi$ from some database $\mathcal{I}$ with $\mathcal{I} \models \psi$?

- (P1) and (P2) are in $\text{NExpTime}$ when a bound on fresh values or plan length is given.

- Undecidable in general.

- Lower complexity for DL-Lite.
Certification and Synthesis

- Is a given action sequence a plan for the goal independently of the input database?
  - in \( \text{coNEXPTime} \) (lower for DL-Lite)

- Can we automatically assemble such plans with or without a bound on the length?
  - in \( \text{coNEXPTime} \) when a bound is given (lower for DL-Lite)
  - undecidable in general
Future Work

- Our technique is applicable to **guarded** FOL
- Extending the approach to logics with a rich support for **identification constraints**
  - Guarded FOL doesn’t support IdCs
  - $C^2$ has good but insufficient support
- Implementation
  - for DL-Lite, or
  - another well-behaved fragment of $C^2$