Monadic Datalog Containment
(and the Hidden Web)

(joint work with M. Benedikt, P. Bourhis, G. Gottlob)

Pierre Senellart
Datalog

Basic query language with recursion.

\[
\begin{align*}
\text{ReachGood}() & \leftarrow \text{Start}(x), \text{Reach}(x, y), \text{Good}(y) \\
\text{Reach}(x, y) & \leftarrow \text{Reach}(x, z), \text{Reach}(z, y) \\
\text{Reach}(x, y) & \leftarrow G(x, y)
\end{align*}
\]
Datalog

Basic query language with recursion.

\[
\text{ReachGood}(x) \leftarrow \text{Start}(x), \text{Reach}(x, y), \text{Good}(y)
\]

\[
\text{Reach}(x, y) \leftarrow \text{Reach}(x, z), \text{Reach}(z, y)
\]

\[
\text{Reach}(x, y) \leftarrow \text{G}(x, y)
\]

- Rules consisting of Horn clauses.
- Heads of rules are intensional predicates.
- Other predicates are extensional (input) predicates.
- Distinguished goal predicate.

Given an instance of the input predicates, computes the goal predicate using a least fixed point semantics.
Datalog

Basic query language with recursion.

\[
\begin{align*}
\text{ReachGood}() & \leftarrow \text{Start}(x), \text{Reach}(x, y), \text{Good}(y) \\
\text{Reach}(x, y) & \leftarrow \text{Reach}(x, z), \text{Reach}(z, y) \\
\text{Reach}(x, y) & \leftarrow G(x, y)
\end{align*}
\]

- Rules consisting of Horn clauses.
- Heads of rules are intensional predicates.
- Other predicates are extensional (input) predicates.
- Distinguished goal predicate.

Given an instance of the input predicates, computes the goal predicate using a least fixed point semantics.

**Monadic Datalog (MDL)** = all intensional predicates are unary.
ReachGood() ← Start(x), Reach(x, y), Good(y)
Reach(x, y) ← Reach(x, z), Reach(z, y)
Reach(x, y) ← G(x, y)

DL query, not MDL

ReachGood() ← Reachable(x), Good(x)
Reachable(y) ← G(x, y), Reachable(x)
Reachable(x) ← Start(x)

(Equivalent) MDL query
$Q \subseteq Q'$ iff for every input instance $D$, $Q(D) \subseteq Q'(D)$

One can use containment to decide equivalence, giving natural way to optimize recursive queries.
\( Q \subseteq Q' \text{ iff for every input instance } D, \ Q(D) \subseteq Q'(D) \)

One can use containment to decide equivalence, giving a natural way to optimize recursive queries.

**Bad news [Shmueli, 1987]**

Datalog containment and equivalence are **undecidable**

But important special cases known to be decidable, e.g., containment of Datalog in Monadic Datalog.
Outline

Datalog Containment

History of MDL Containment: 1990–2012
    Classical Results
    Limited Access Containment

Some Open Questions, and their Resolution
Outline

Datalog Containment

History of MDL Containment: 1990–2012
  Classical Results
  Limited Access Containment

Some Open Questions, and their Resolution
Decidability of DL Containment in MDL
[Courcelle, 1991]

Idea:

- $Q$ is not contained in $Q'$ iff there is a witness instance in which $Q$ holds and $Q'$ does not hold.

- The witness instance can be taken to be tree-like – of tree-width at most $|Q|$.
  - Thus can reduce non-containment reasoning to existence of a certain kind of tree.

- Exploit fact that checking satisfiability of certain kinds of sentences over trees is decidable.
Decidable Containment [Courcelle, 1991]

\[
\text{Goal()} \leftarrow U(x), R(x, y), V(y) \\
U(x) \leftarrow S(x, z), W(z, z), U(z) \\
\begin{align*}
Q : \quad U(x) & \leftarrow P(x, x) \\
\end{align*}
\]

\ldots
Decidable Containment [Courcelle, 1991]

\[
\text{Goal()} \leftarrow U(x), R(x, y), V(y) \\
U(x) \leftarrow S(x, z), W(z, z), U(z) \\
Q : \\
U(x) \leftarrow P(x, x)
\]

Chase models:

\[
\text{Goal()} \Rightarrow \text{Create } x_0, y_0 \quad \text{U}(x_0), R(x_0, y_0), V(y_0) \\
\text{U}(x_0) \Rightarrow \text{Create fresh } z_0 \neq x_0, y_0 \quad \text{S}(x_0, z_0), W(z_0, z_0), U(z_0) \\
\text{U}(z_0) \Rightarrow \text{Create ..}
\]
Relational Instance $I \Rightarrow$

\textit{code}(I), tree labeled with info about bags = collection of atoms over $U, R, S \ldots$ One bag for each chase step.

Label alphabet of codes: atoms and relationship of variables in one bag shared with children.

$\Rightarrow$ Code is a finite-labeled tree containing all the information of the instance.
Relational Instance $I \Rightarrow code(I)$, tree labeled with info about bags = collection of atoms over $U, R, S \ldots$ One bag for each chase step.

Label alphabet of codes: atoms and relationship of variables in one bag shared with children.

⇒ Code is a finite-labeled tree containing all the information of the instance.

Universality of tree-like models:

For Datalog $Q$ and $Q'$, non-containment of $Q$ in $Q' \iff \exists$ tree $T$ such that $decode(T)$ satisfies $Q \land \neg Q'$. 
(1) [Courcelle, 1991]: “Code contains all the information of the instance, and decoding an instance from the tree is simple.”

If $Q'$ is in Monadic Datalog (more generally, in Monadic Second Order Logic), one can rewrite $\neg Q'$ to formula $R'$ such that for any tree-like instance $I$, checking $R'$ on $\text{code}(I)$ is the same as checking $\neg Q'$ on $I$. 

(2) [Thatcher and Wright, 1968, Doner, 1970]: Monadic Second Order Logic is decidable on labeled trees. 

(1)+(2) gives decidability of Datalog in MDL. Closely-related to decidability of query answering for many classes of dependencies.
(1) [Courcelle, 1991]: “Code contains all the information of the instance, and decoding an instance from the tree is simple.” If $Q'$ is in Monadic Datalog (more generally, in Monadic Second Order Logic), one can rewrite $\neg Q'$ to formula $R'$ such that for any tree-like instance $I$, checking $R'$ on $\text{code}(I)$ is the same as checking $\neg Q'$ on $I$.

(2) [Thatcher and Wright, 1968, Doner, 1970]: Monadic Second Order Logic is decidable on labeled trees.
(1) [Courcelle, 1991]: “Code contains all the information of the instance, and decoding an instance from the tree is simple.” If $Q'$ is in Monadic Datalog (more generally, in Monadic Second Order Logic), one can rewrite $\neg Q'$ to formula $R'$ such that for any tree-like instance $I$, checking $R'$ on $\text{code}(I)$ is the same as checking $\neg Q'$ on $I$.

(2) [Thatcher and Wright, 1968, Doner, 1970]: Monadic Second Order Logic is decidable on labeled trees.

(1)+(2) gives decidability of Datalog in MDL.
(1) [Courcelle, 1991]: “Code contains all the information of the instance, and decoding an instance from the tree is simple.” If $Q'$ is in Monadic Datalog (more generally, in Monadic Second Order Logic), one can rewrite $\neg Q'$ to formula $R'$ such that for any tree-like instance $I$, checking $R'$ on $\text{code}(I)$ is the same as checking $\neg Q'$ on $I$.

(2) [Thatcher and Wright, 1968, Doner, 1970]: Monadic Second Order Logic is decidable on labeled trees.

(1)+(2) gives decidability of Datalog in MDL.

Closely-related to decidability of query answering for many classes of dependencies.
MDL Containment [Cosmadakis et al., 1988]

[Cosmadakis et al., 1988]

MDL containment is in 2EXPTIME; if $Q \land \neg Q'$ satisfiable it has a doubly-exponential model.
MDL Containment [Cosmadakis et al., 1988]

[Cosmadakis et al., 1988]
MDL containment is in 2EXPTIME; if $Q \land \neg Q'$ satisfiable it has a doubly-exponential model.

Idea, simplified for MDL/UCQ containment: Can collapse any two nodes with the same type.
In the case of MDL, this type can be captured by a tree automaton.
Outline

Datalog Containment

History of MDL Containment: 1990–2012
  Classical Results
  Limited Access Containment

Some Open Questions, and their Resolution
Restricted Access Scenario

We have a relational schema with relations $R_1 \ldots R_n$. Each $R_i$ has some arity $ar_i$ and is additionally restricted in that access is only via a set of access methods $m_1 \ldots m_{n_i}$. An access method has a set of “input positions” $S \subseteq \{1 \ldots ar_i\}$ that require known values. An access to method $m_i$ is a binding of the input positions of $m_i$, which returns an output.

Given an instance $I$ for the schema, a set of initial constants $C_0$ the access patterns define a collection of valid access paths: sequences of accesses $ac_1 \ldots ac_k$ and responses such that each value in the binding to $ac_i$ is either in $C_0$ or is an output of $ac_j$ with $j < i$. Facts that are returned by valid paths are the accessible data.
Access Methods

Method **ApartmentFind:**
Region, Area, NumBeds → Address, Price, Description, Link

Above the input fields have enum domains – but in general the domains can be infinite (e.g., textboxes). Querying over limited interfaces arises in many other data management settings: web services, legacy database managers.
Given two conjunctive queries $Q$, $Q'$ and a schema with access patterns, determine whether $Q$ and $Q'$ agree on the accessible data. Similarly $Q$ is contained in $Q'$ relative to the access patterns if whenever $Q$ is true on the accessible data, then so is $Q'$.

**Question**

What is the complexity of query equivalence, containment under access patterns?
Given two conjunctive queries \( Q, Q' \) and a schema with access patterns, determine whether \( Q \) and \( Q' \) agree on the accessible data. Similarly \( Q \) is contained in \( Q' \) relative to the access patterns if whenever \( Q \) is true on the accessible data, then so is \( Q' \).

**Question**

What is the complexity of query equivalence, containment under access patterns?

Containment can be used to solve a number of other static analysis questions about limited access schemas, such as whether an access is relevant to a query.
Axiomatizing accessibility

\[
\text{Accessible}(x_j) \leftarrow (R(\bar{x}) \land \bigwedge_{i \in \text{input}(m)} \text{Accessible}(x_i))
\]

\[
\text{Accessible}(c) \leftarrow
\]

\(c\) a constant or value in some enum datatype of the schema.

An MDL program that computes the accessible values: those obtainable via a valid access path.

⇒ For any UCQ query \(Q\) one can write an MDL query \(Q_{\text{acc}}\) that computes the value of \(Q\) restricting to accessible values.
Axiomatizing accessibility

\[
\text{Accessible}(x_j) \leftarrow (R(x) \land \bigwedge_{i \in \text{input}(m)} \text{Accessible}(x_i))
\]

\[
\text{Accessible}(c) \leftarrow
\]

\(c\) a constant or value in some enum datatype of the schema.

An MDL program that computes the accessible values: those obtainable via a valid access path.

⇒ For any UCQ query \(Q\) one can write an MDL query \(Q_{acc}\) that computes the value of \(Q\) restricting to accessible values.

\(Q\) contained in \(Q'\) under access patterns ⇔

\(Q_{acc}\) contained in \(Q'\) on all databases.

Containment of a Monadic Datalog Query in a UCQ!
Outline

Datalog Containment

History of MDL Containment: 1990–2012

Some Open Questions, and their Resolution
(Formerly) Open Questions

- Is the 2EXPTIME bound on MDL/UCQ containment tight? Only known lower-bound was PSPACE.
(Formerly) Open Questions

- Is the $2\text{EXPTIME}$ bound on MDL/UCQ containment tight? Only known lower-bound was $\text{PSPACE}$.

- What about containment under limited access patterns? Only obvious lower bound of $\text{NP}$. $\text{coNEXPTIME}$ bound proved for special cases [Calì and Martinenghi, 2008]
Our results

[Benedikt et al., 2011] + [Benedikt et al., 2012]

- Containment of UCQs relative to access patterns is coNEXPTIME-complete, provided that every relation has a single access method. Complexity reduces to EXPTIME with no constants or enum datatypes.
- Containment of MDL in UCQs is 2EXPTIME-complete.
- Containment of UCQs relative to access methods in general is 2EXPTIME-complete (if some attributes can be projected out).
- Closely related to containment of tree automata in UCQs over trees with different schemas:
  - Child relation: EXPTIME-complete
  - Child and label equality: coNEXPTIME-complete
  - Child and child-or-self: 2EXPTIME-complete
Merci.

Most of the slides’ content due to M. Benedikt!


