Monadic Datalog Containment (and the Hidden Web)

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

PIERRE SENELLART

18 October 2012, *MoDaS–Webdam Eilat Workshop*

Datalog

Basic query language with recursion.

$$
\begin{aligned}
\mathit{ReachGood}() &\leftarrow \mathit{Start}(x), \mathit{Reach}(x,y), \mathit{Good}(y)\\
\mathit{Reach}(x,y) &\leftarrow \mathit{Reach}(x,z), \mathit{Reach}(z,y)\\
\mathit{Reach}(x,y) &\leftarrow G(x,y)
\end{aligned}
$$

Datalog

Basic query language with recursion.

$$\begin{aligned} ReachGood() &\leftarrow Start(x), Reach(x,y), Good(y) \\ Reach(x,y) &\leftarrow Reach(x,z), Reach(z,y) \\ Reach(x,y) &\leftarrow G(x,y) \end{aligned}$$

- 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.

$$\begin{aligned} ReachGood() &\leftarrow Start(x), Reach(x,y), Good(y) \\ Reach(x,y) &\leftarrow Reach(x,z), Reach(z,y) \\ Reach(x,y) &\leftarrow G(x,y) \end{aligned}$$

- 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.

Datalog

$$ReachGood() \leftarrow Start(x), Reach(x, y), Good(y)$$
$$Reach(x, y) \leftarrow Reach(x, z), Reach(z, y)$$
$$Reach(x, y) \leftarrow G(x, y)$$

DL query, not MDL

$$ReachGood() \leftarrow Reachable(x), Good(x)$$
$$Reachable(y) \leftarrow G(x, y), Reachable(x)$$
$$Reachable(x) \leftarrow Start(x)$$

(Equivalent) MDL query

Containment of Datalog

$Q \subseteq Q'$ 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.

Containment of Datalog

$Q \subseteq Q'$ 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.

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]

$$
Q:\quad
\begin{aligned}
Goal() &\leftarrow U(x), R(x,y), V(y)\\
U(x) &\leftarrow S(x,z), W(z,z), U(z)\\
U(x) &\leftarrow P(x,x)\\
&\dots
\end{aligned}
$$

Decidable Containment [Courcelle, 1991]

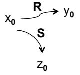
$$Q: \begin{array}{l} Goal() \leftarrow U(x), R(x,y), V(y) \\ U(x) \leftarrow S(x,z), W(z,z), U(z) \\ U(x) \leftarrow P(x,x) \\ \dots \end{array}$$

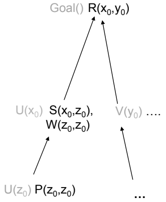
Chase models:

$Goal() \Rightarrow$ Create x_0, y_0 $\quad U(x_0), R(x_0,y_0), V(y_0)$

$U(x_0) \Rightarrow$ Create fresh $z_0 \neq x_0, y_0$ $\quad S(x_0,z_0), W(z_0,z_0), U(z_0)$

$U(z_0) \Rightarrow$ Create . .





Tree-like instances and decidability [Courcelle, 1991]

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.

$\Rightarrow$ Code is a finite-labeled tree containing all the information of the instance.

Tree-like instances and decidability [Courcelle, 1991]

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.

$\Rightarrow$ 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 $Q' \Leftrightarrow$
$\exists$ tree T such that $decode(T)$ satisfies $Q \wedge \neg Q'$.

Tree-like instances and decidability [Courcelle, 1991]

(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 $code(I)$ is the same as checking $\neg Q'$ on I.

Tree-like instances and decidability [Courcelle, 1991]

(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 $\mathit{code}(I)$ 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.

Tree-like instances and decidability [Courcelle, 1991]

(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 $\mathit{code}(I)$ 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.

Tree-like instances and decidability [Courcelle, 1991]

(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 $\mathit{code}(I)$ 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 in is in 2EXPTIME; if $Q \wedge \neg Q'$ satisfiable it has a doubly-exponential model.

MDL Containment [Cosmadakis et al., 1988]

[Cosmadakis et al., 1988]

MDL containment in is in 2EXPTIME; if $Q \wedge \neg Q'$ 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

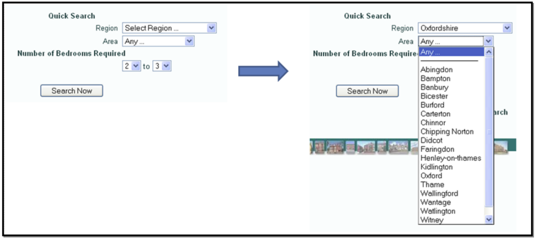
MDL containment and Restricted Interfaces

Restricted Access Scenario

We have a relational schema with relations $R_1 \dots R_n$.
Each R_i has some arity ar_i and is additionally restricted in that access is only via a set of access methods $m_1 \dots m_{n_i}$. An access method has a set of "input positions" $S \subseteq \{1 \dots ar_i\}$ 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 $ac_1 \dots ac_k$ 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 $\rightarrow$ 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.

Equivalence with 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?

Equivalence with 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.

Limited Access Containment and MDL [Li and Chang, 2001]

Axiomatizing accessibility

$$Accessible(x_j) \leftarrow (R(\vec{x}) \wedge \bigwedge_{i \in input(m)} Accessible(x_i))$$

$$Accessible(c) \leftarrow$$

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.

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

Limited Access Containment and MDL [Li and Chang, 2001]

Axiomatizing accessibility

$$Accessible(x_j) \leftarrow (R(\vec{x}) \wedge \bigwedge_{i \in input(m)} Accessible(x_i))$$

$$Accessible(c) \leftarrow$$

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.

$\Rightarrow$ 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 $\Leftrightarrow$
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 2EXPTIME bound on MDL/UCQ containment tight?
  Only known lower-bound was PSPACE.
- What about containment under limited access patterns?
  Only obvious lower bound of NP.
  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!

Michael Benedikt, Georg Gottlob, and Pierre Senellart. Determining relevance of accesses at runtime. In *PODS*, 2011.

Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic datalog containment. In *Proc. ICALP*, Warwick, United Kingdom, July 2012.

Andrea Calì and Davide Martinenghi. Conjunctive query containment under access limitations. In *ER*, 2008.

Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs. In *STOC*, 1988.

Bruno Courcelle. Recursive queries and context-free graph grammars. *Theoretical Computer Science*, 78(1), 1991.

John Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

Chen Li and Edward Chang. Answering queries with useful bindings. *TODS*, 26(3):313–343, 2001.

Oded Shmueli. Decidability and Expressiveness Aspects of Logic Queries. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'87)*, pages 237–249, 1987.

James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.