
# Monadic Datalog Containment (and the Hidden Web)

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

PIERRE SENELLART

31 May 2012, *Dahu Seminar*

# Datalog

Basic query language with recursion.

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

# Datalog

Basic query language with recursion.

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

- 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.

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

- 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

# Outline

# 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]

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

# Decidable Containment [Courcelle, 1991]

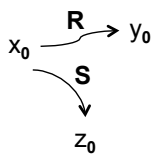
$$Q: \begin{array}{l} Goal() \leftarrow U(x), R(x,y), V(y) \\ U(x) \leftarrow S(x,z), W(z,z), U(z) \\ U(x) \leftarrow P(x,x) \\ \ldots \end{array}$$

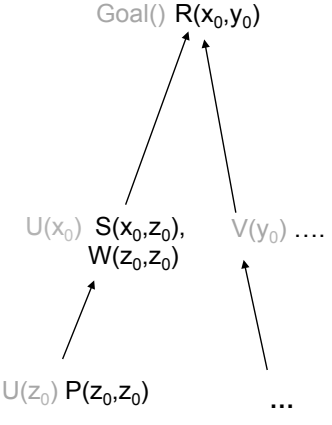
Chase models:

$Goal() \Rightarrow$ 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 $code(I)$ 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 $code(I)$ 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 $code(I)$ 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 MDL tree-like models, every bag $b$ is associated with one value of the corresponding instance, $val(b)$.

"Everything important" about the instance element $val(b)$ is captured by $b$ (collection of $Q$ atoms) and the collection of subqueries of $Q'$ that $val(b)$ satisfies.

# 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.

"Everything important" about the instance element $val(b)$ is captured by $b$ (collection of $Q$ atoms) and the collection of subqueries of $Q'$ that $val(b)$ satisfies.

Formally: can get a tree automaton that captures the counterexample models, where the states are of the above form.

# 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.

"Everything important" about the instance element $val(b)$ is captured by $b$ (collection of $Q$ atoms) and the collection of subqueries of $Q'$ that $val(b)$ satisfies.

Formally: can get a tree automaton that captures the counterexample models, where the states are of the above form.

Doubly-exponential many states $\Rightarrow$ reachability in a tree automaton decidable in time proportional with its number of states.

# 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.

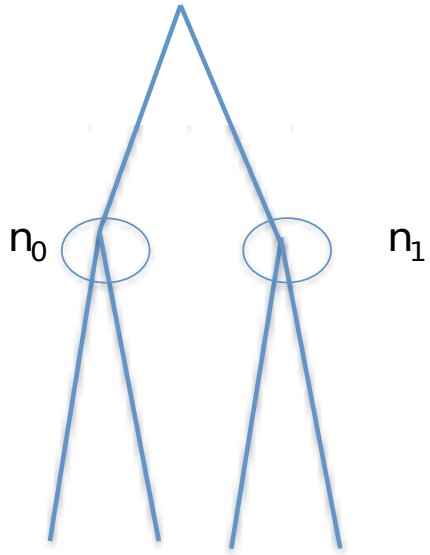
"Everything important" about the instance element $val(b)$ is captured by $b$ (collection of $Q$ atoms) and the collection of subqueries of $Q'$ that $val(b)$ satisfies.

Formally: can get a tree automaton that captures the counterexample models, where the states are of the above form.

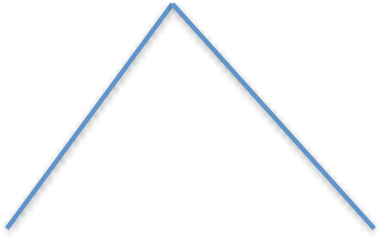
Doubly-exponential many states $\Rightarrow$ reachability in a tree automaton decidable in time proportional with its number of states.

Using a similar idea, [Chaudhuri and Vardi, 1992, 1997] showed that DL/UCQ containment is in 2EXPTIME (and is 2EXPTIME-complete).

# Abstracting nodes by collections of subqueries

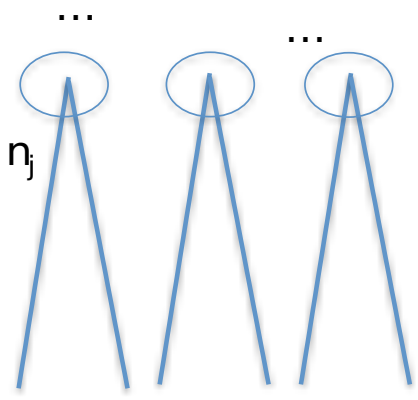



$Q' = S_0(x)$
$R_1(x,y), T_1(y),$
$R_2(x,y), T_2(y), \ldots$
$R_3(x,y), T_3(y), \ldots$



$S_0(x), R_1(x,y), ..$
$R_3(x,y);$
$S_0(x), R_2(x,y), ..$
$R_4(x,y) \ldots$

$S_0(x), R_3(x,y), ..$
$R_4(x,y);$
$\ldots$



# Outline

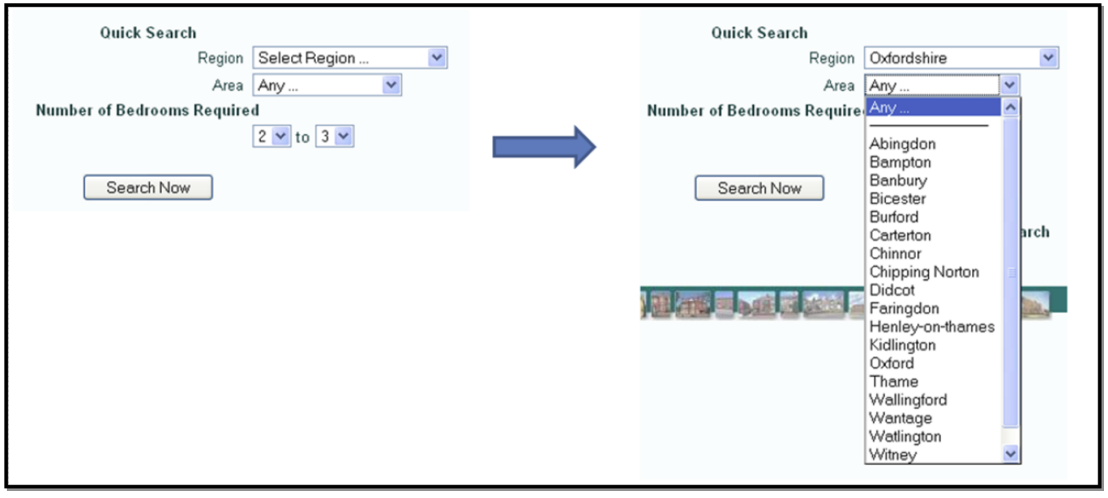
# MDL containment and Restricted Interfaces

## 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 $\rightarrow$ 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

# (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.
  NEXPTIME 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 NEXPTIME-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.

# Outline

# Single-access Limited Access Containment is in EXPTIME

First idea: there are few ways to satisfy a conjunctive query in a tree. Instead of characterizing the type of a node $n$ by all the subqueries of $Q'$ that are satisfied in the subtree of $n$, just look at the maximal subqueries of $Q'$ that are satisfied in the subtree of $n$ when mapping a given variable of $Q'$ to $n$.

In a tree, should be a single maximal way to extend a partial mapping of a variable to a node: $\Rightarrow$ only exponential number of types!

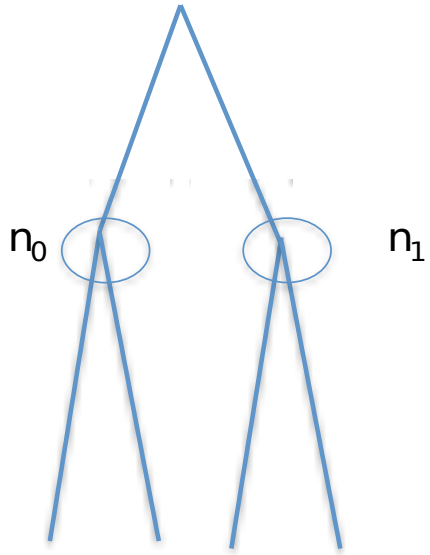
# Single-access Limited Access Containment is in EXPTIME

First idea: there are few ways to satisfy a conjunctive query in a tree. Instead of characterizing the type of a node $n$ by all the subqueries of $Q'$ that are satisfied in the subtree of $n$, just look at the maximal subqueries of $Q'$ that are satisfied in the subtree of $n$ when mapping a given variable of $Q'$ to $n$.

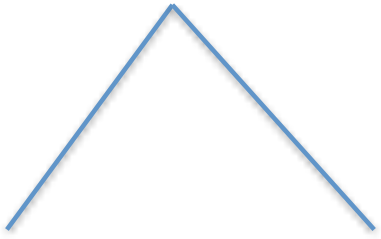
In a tree, should be a single maximal way to extend a partial mapping of a variable to a node: $\Rightarrow$ only exponential number of types!

Problem: this is true in a tree, but in a tree-like instance there might be many maximal subqueries that a node satisfies (e.g., because of several atoms that could map to some relation that the node satisfies, in different components).

# Abstracting nodes by maximal subqueries

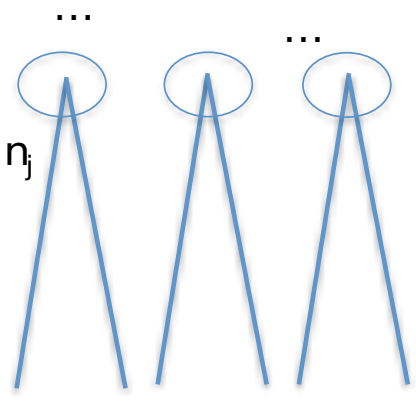


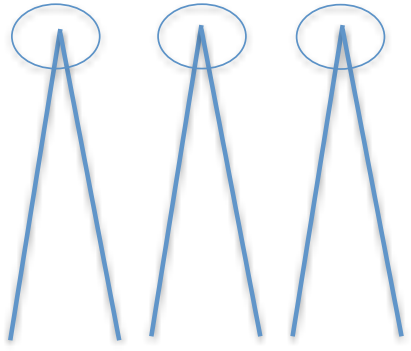

$Q' = S_0(x)$
$R_1(x,y), T_1(y),$
$R_2(x,y), T_2(y), \ldots$
$R_3(x,y), T_3(y), \ldots$



$S_0(x), R_1(x,y), ..$
$R_3(x,y)$

$S_0(x), R_3(x,y), ..$
$R_4(x,y)$



# Very tree-like instances

A class of tree-like instances satisfies the Unique Mapping Condition (UMC) if for any node $n$, any conjunctive query $Q'$, and any atom $A$ there exists at most one connected subquery $Q''$ of $Q'$ maximal w.r.t. mapping into the subtree of $n$, in such a way that $A$ maps into $\mathrm{bag}(n)$.

# Very tree-like instances

A class of tree-like instances satisfies the Unique Mapping Condition (UMC) if for any node $n$, any conjunctive query $Q'$, and any atom $A$ there exists at most one connected subquery $Q''$ of $Q'$ maximal w.r.t. mapping into the subtree of $n$, in such a way that $A$ maps into $\text{bag}(n)$.

**Theorem**

If a class of tree-like instances satisfies the UMC, then there is an *EXPTIME* function taking a (U)CQ $Q'$ to a tree automaton that accepts all codes of instances in the class satisfying $Q'$.

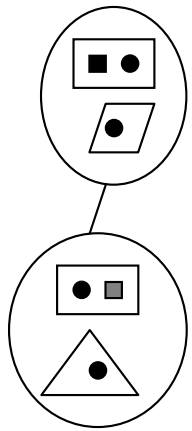
Proof: use functions $A \mapsto$ Maximal mappable subquery for $A$ as states.

# Very tree-like instances

A diversified tree-like instance is a tree-like instance in which:

(i) for each node $n$ which is not the root, for each relation $R$, there exists at most one fact in $\mathrm{bag}(n)$ having the relation name $R$;

(ii) we cannot have a value $v$ other than the output node of the root such that there are two distinct facts in the instance with the same relation name which have $v$ in the same position.



# Very tree-like instances and limited access patterns

If $Q$ is not contained in UCQ $Q'$ under single-access access patterns, then there is a diversified tree-like instance $I$ such that $I$ satisfies $Q \wedge \neg Q'$ and for each $n$ of $I$, the size of its bag is polynomial in $Q$.

# Very tree-like instances and few mappings

## Key Lemma

The class of diversified tree-like instances satisfies the UMC.

# Very tree-like instances and few mappings

## Key Lemma

The class of diversified tree-like instances satisfies the UMC.

Proof Idea:

- Fix tree node $n$ and atom $A$ from $Q'$ that maps to $n$
- We have no choice for how to extend a mapping down into the subtree of $n$: Suppose we have already mapped into bag $m$, including some atom with variables $x$. We now have an atom $B$ that includes $x$, and have to decide how to map it into $m$ or one of the children $m'$ and $m''$ of $m$. If we can extend the mapping to $m$, then we cannot extend it to a child, since then we would have two atoms in the same position of the same relation in parent and child. If we could extend it to $m$ then we would have two instances of the same relation in a bag.

  In both cases, a contradiction of diversified instance.

# Putting it all together

- Counterexamples to containment for limited-access schemas with a single access per relation can be taken to be diversified tree-like.
- Diversified instances satisfy the UMC, so they have unique maximal subqueries.
- We can construct a tree automaton that captures all diversified tree-like counterexamples, whose types are the vectors of maximal queries.
- Creating this automaton and checking non-emptiness can be done in EXPTIME.

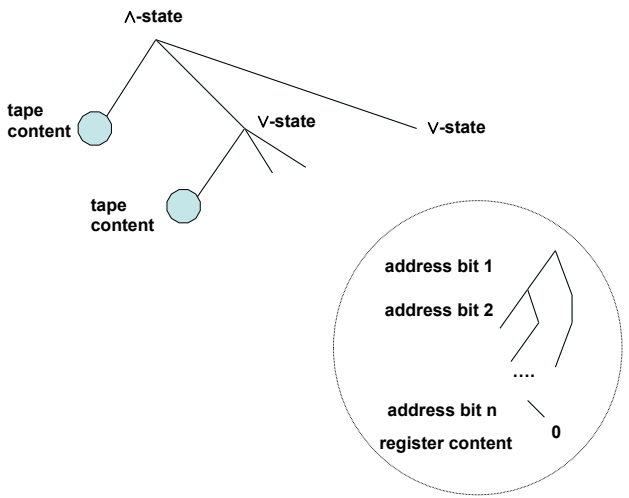
# Upper and Lower

- Upper bounds: use analysis of tree-like models. Also can be used to show:
  - Results in the presence of constants – still have exponential sized counterexample models, but complexity moves to NEXPTIME from EXPTIME
  - New results over trees – tree automata containment in a tree pattern with only child axis is EXPTIME
- Lower bounds: tricks for coding computation in MDL/limited access containment
  - Single-access Limited Access Containment is EXPTIME-hard
  - With constants, becomes NEXPTIME-hard
  - General Limited Access Containment is 2EXPTIME-hard: single access restriction makes a difference

# Outline

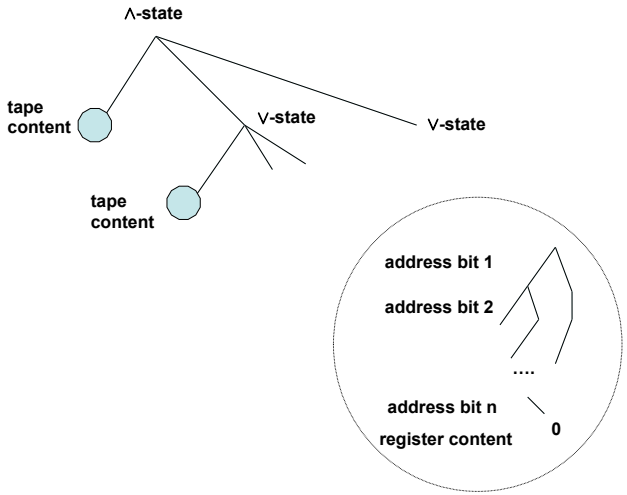
# MDL/UCQ Containment 2EXPTIME-hard

First idea: 2EXPTIME=AEXPSPACE, so reduce from the halting problem for an alternating EXPSPACE machine $M$. Computation of $M$ is a tree of configurations, with each configuration consisting of an exponential size subtree.



# MDL/UCQ Containment 2EXPTIME-hard

MDL $Q$: generate a tree of the right shape (full binary tree for each configuration, etc.)
UCQ $Q'$: return true if the tree does not satisfy the rules of a valid configuration, e.g., return true if there are two adjacent configurations *config* and *config'*, two cells $c$ in *config*, $c'$ in *config'* with the same address, different content, but far away from the address with the head.



# MDL/UCQ Containment is 2EXPTIME-hard

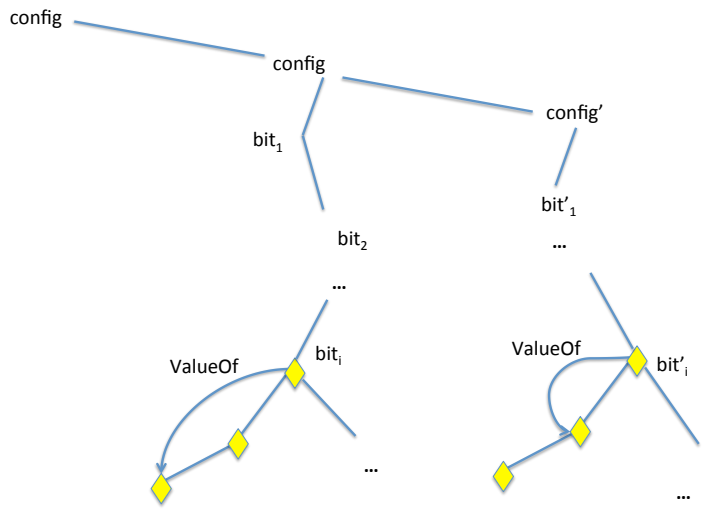
Problem: how to do the validation with a CQ?

Need to check for the existence of two adjacent configurations $C_1$ and $C_2$, whose subtrees are invalid: e.g., a path $p$ in $C_1$ away from the head has a certain letter, and a path $p'$ in $C_2$ with the same address has a different letter.

# MDL/UCQ is hard

## Key idea [Björklund et al., 2008]

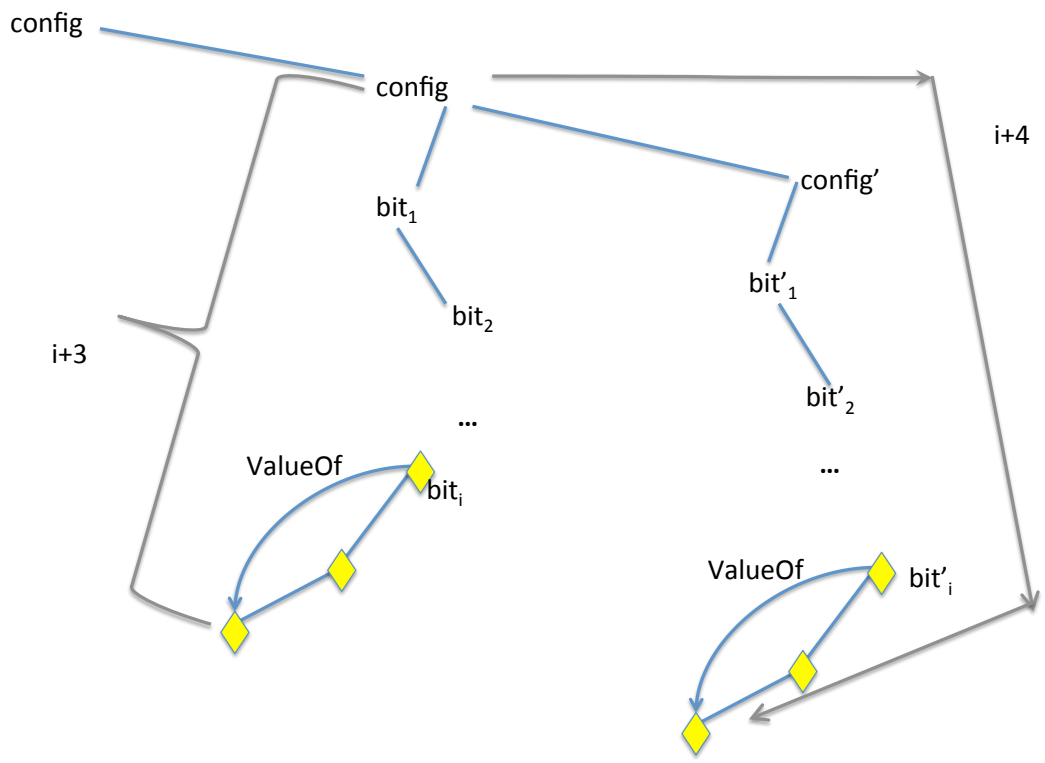
Use a slightly strange coding of the tree.



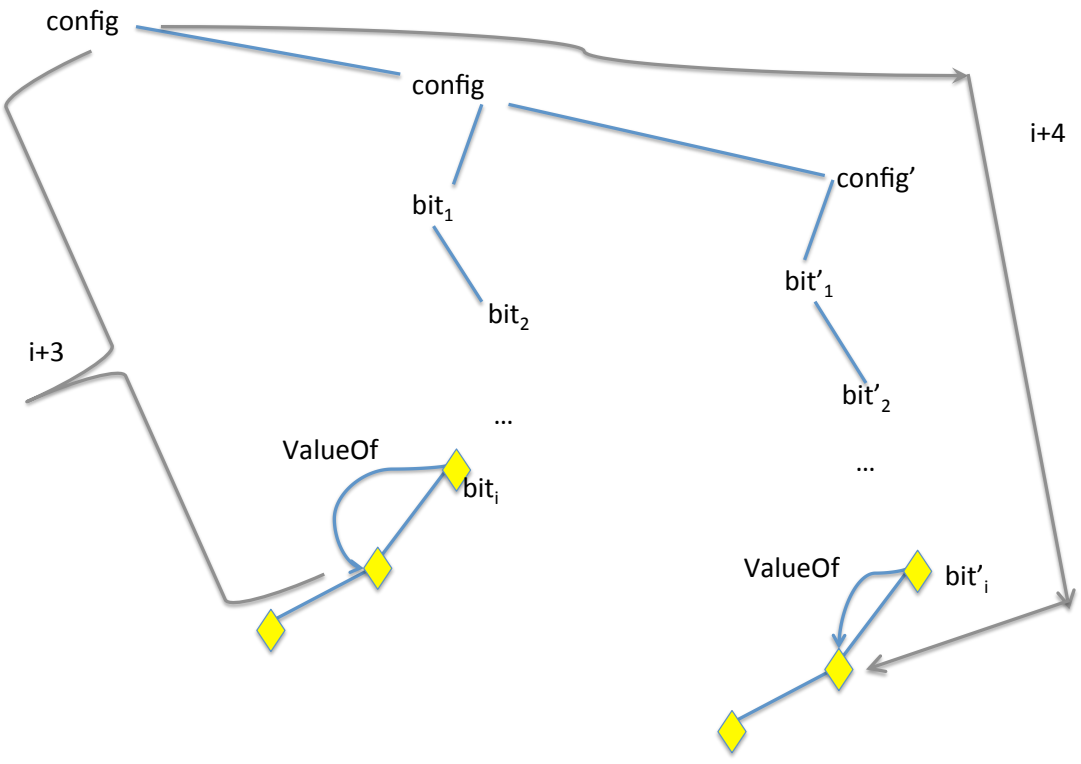
# Strange coding helps

Key construct: conjunctive query $=_i (s_1, b_x, s_2, b_y)$, that checks if a bit $b_x$ of depth $i$ in the configuration $s_1$ is equal to some bit $b_y$ of depth $i$ of the configuration $s_2$, where $s_2$ is a successor configuration to $s_1$.

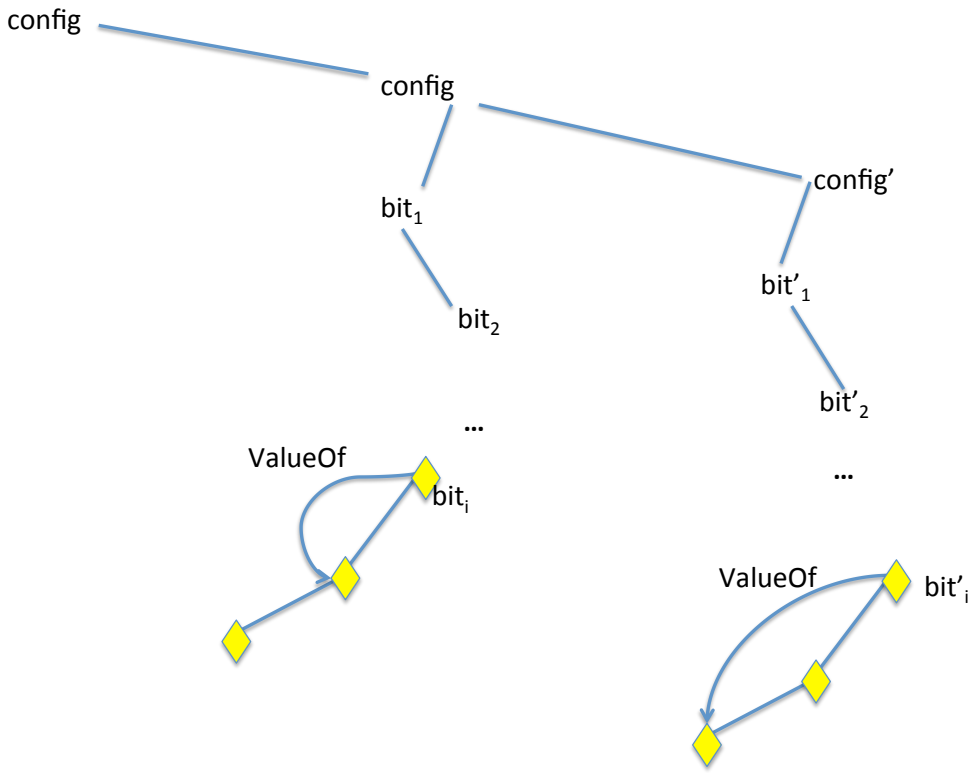
# MDL/UCQ is hard



# MDL/UCQ is hard

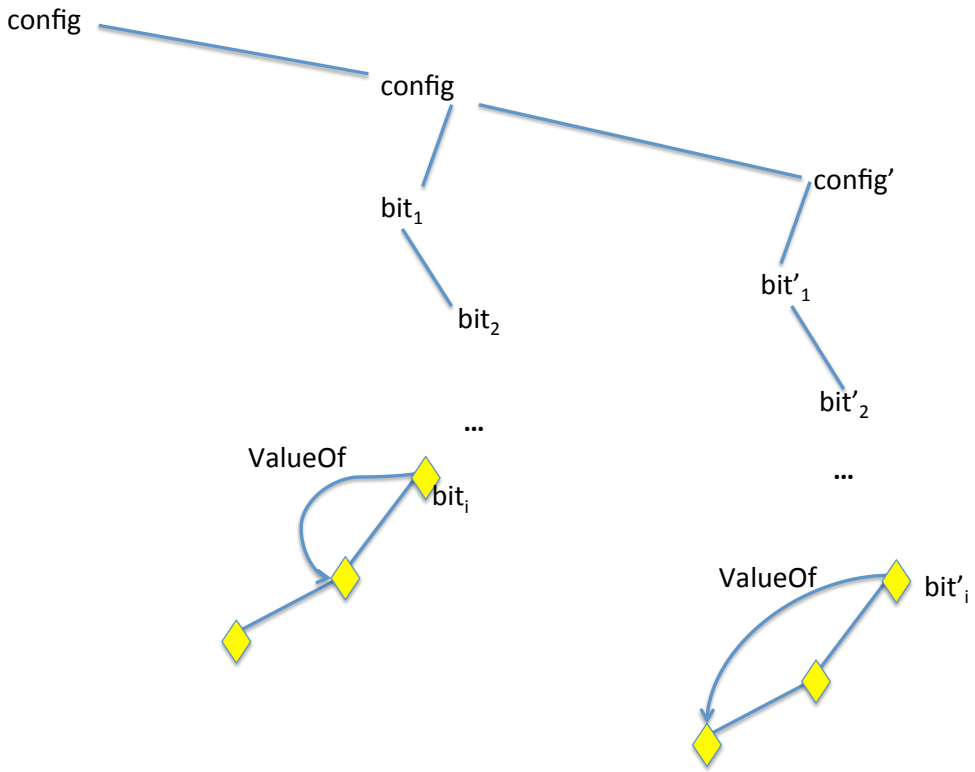


# MDL/UCQ is hard




No node that has distances $i + 3$ from one bit and $i + 4$ from the other.

# MDL/UCQ is hard




No node that has distances $i+3$ from one bit and $i+4$ from the other.

Thus: define $=_i (s_1, b_x, s_2, b_y)$ to check for a node that is $i+3$ from $b_x$ and $i+4$ from $b_y$.

# More on lower bounds

With more tricks

- 2EXPTIME-hardness for MDL in CQs.
- 2EXPTIME-hardness for limited access containment of UCQs with multiple accesses per relation.
- EXPTIME-hardness for limited access containment of UCQs with single access per relation.
- co-NEXPTIME-hardness for limited access containment of UCQs with constants over schemas with a single access per relation.

# Outline

# Summary

## MDL in UCQs

- Decidability of containment hinges on sufficiency of tree-like models.
- Complexity related to "fine structure" of tree-like models, which can be related to syntactic restrictions on the MDL query.

# Summary

## MDL in UCQs

- Decidability of containment hinges on sufficiency of tree-like models.
- Complexity related to "fine structure" of tree-like models, which can be related to syntactic restrictions on the MDL query.

## Current and Future Work

- Explore relationship between Hidden Web/MDL containment problems and query answering under guarded dependencies.
- Look at impact of constraints on Hidden Web/limited access problems.

# 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.

Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS*, 2008.

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

Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *PODS*, 1992.

Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. *JCSS*, 54(1):61–78, 1997.

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.