Schema-Guided Query Induction

Jérôme Champavère

Ph.D. Defense

September 10, 2010

Supervisors: Joachim Niehren and Rémi Gilleron
Advisor: Aurélien Lemay

Big Picture

- XML:
  - Standard language for storing and exchanging semi-structured data
  - Widespread in database, Web and document communities
- Schemas: meta-description of XML documents
- XML queries:
  - Select data in XML documents
  - Basis for XML transformations

How to Design Queries?

- Experts manually define queries by:
  - Writing Perl, Python or shell scripts
  - Using W3C standards (XPath)
  - Generating wrappers with tools like Lixto [Baumgartner et al., 2001]
- Average users need help:
  - Generic approach
  - No prior skills required
  - Use of graphical user interfaces

Wrapper Induction for Web Information Extraction

- Automatic construction of wrappers (queries)
- Various machine learning techniques:
  - String-based [Kushmerick, 2002]
  - Tree-based:
    - Inductive logic programming [Cohen et al., 2002]
    - Tree automata inference [Kosala, 2003]
    - Conditional random fields [Kristjansson et al., 2004]
    - Classification [Marty et al., 2006]

Wrapper Induction for Web Information Extraction

- Automatic construction of wrappers (queries)
- Various machine learning techniques:
  - String-based [Kushmerick, 2002]
  - Tree-based:
    - Inductive logic programming [Cohen et al., 2002]
    - **Tree automata inference**
    - Conditional random fields [Kristjansson et al., 2004]
    - Classification [Marty et al., 2006]

Motivations for Schema-Guided Query Induction

- Add schema information into existing learning algorithms
  - Learn queries consistent with the schema
  - Define more accurate pruning strategies
- Formal characterization of heuristics
  - Learnability results
  - Classification of queries: what makes it hard to learn?
- Beyond HTML:
  - XML data have a richer semantics thanks to schemas
  - Nodes in XML trees associated to graphical objects

Contributions

- How to use schema knowledge in algorithms
  - Infer schema-consistent queries
    - Efficient inclusion checking
    - New learnability result
  - Schema-guided pruning
- Class of queries that are learnable w.r.t. pruning
  - Formal characterization by stable queries
  - New learnability result
- Relevance of stable queries in practice
  - New experiments with XML datasets
  - Best pruning strategies: schema-guided and related to stability

Contributions

- How to use schema knowledge in algorithms
  - Infer schema-consistent queries
    - Efficient inclusion checking
    - New learnability result
  - **Schema-guided pruning**
- Class of queries that are learnable w.r.t. pruning
  - Formal characterization by **stable queries**
  - **New learnability result**
- Relevance of stable queries in practice
  - **New experiments** with XML datasets
  - Best pruning strategies: schema-guided and related to stability

Outline

1. XML, Schemas and Queries
2. Tree Automata
3. Schema-Guided Pruning
4. Stable Queries
5. Learnability Results
6. Experiments

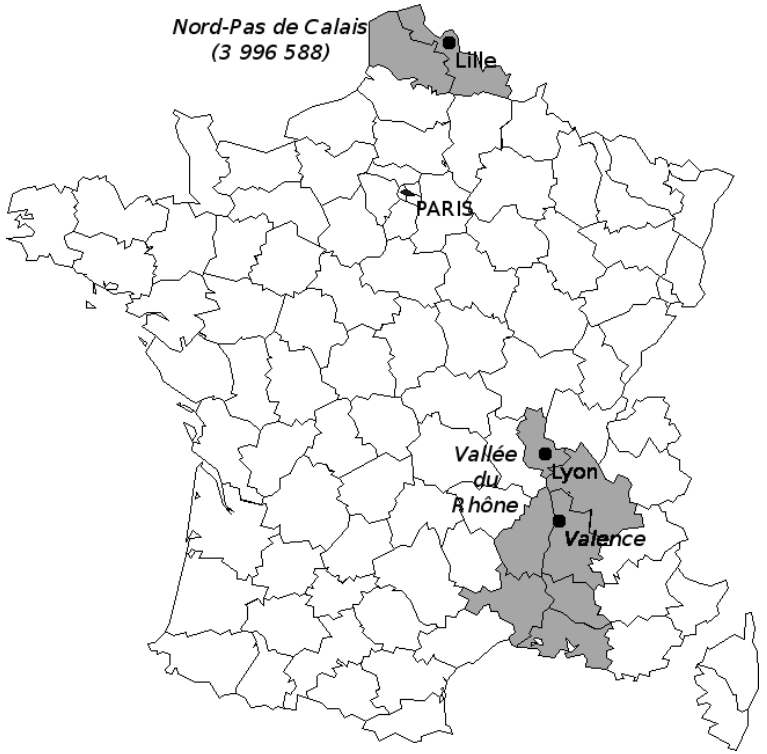
Outline

1 XML, Schemas and Queries
2 Tree Automata
3 Schema-Guided Pruning
4 Stable Queries
5 Learnability Results
6 Experiments

XML

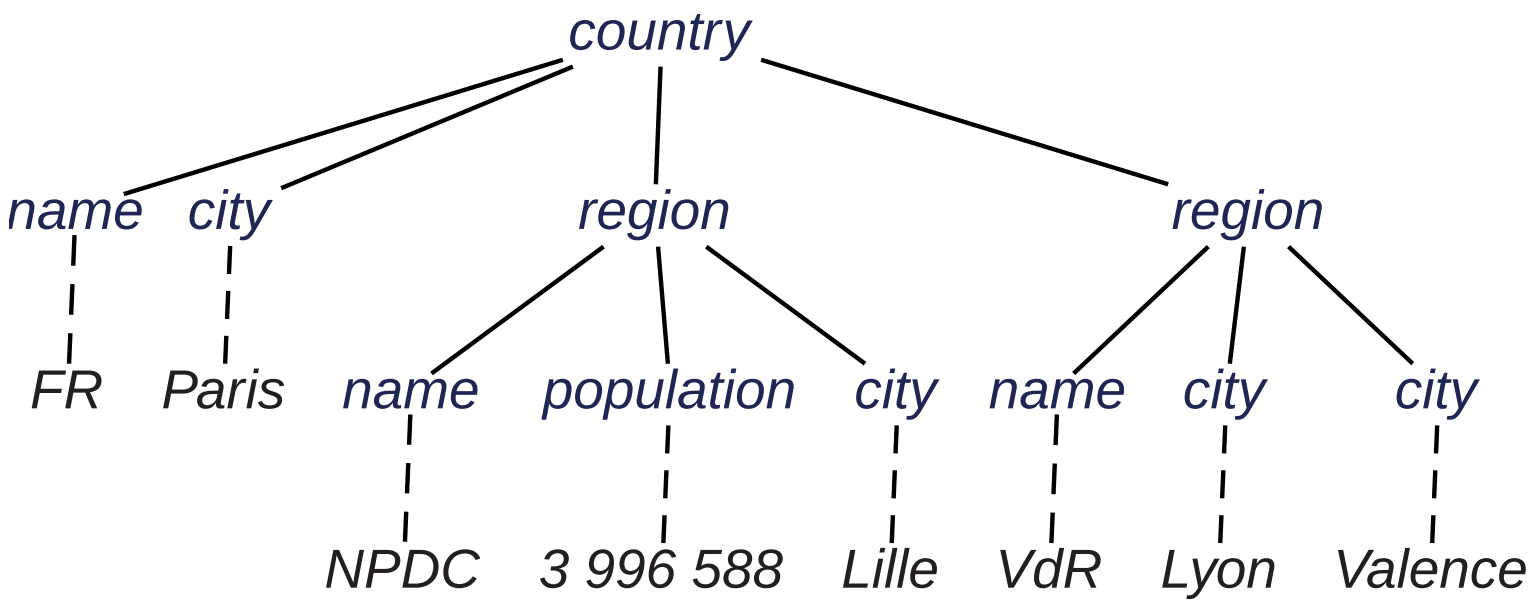
Example: representation of geographical data

```
<country>
  <name>France</name>
  <city>Paris</city>
  <region>
    <name>Nord–Pas de Calais</name>
    <population>3 996 588</population>
    <city>Lille</city>
  </region>
  <region>
    <name>Vallée du Rhône</name>
    <city>Lyon</city>
    <city>Valence</city>
  </region>
</country>
```



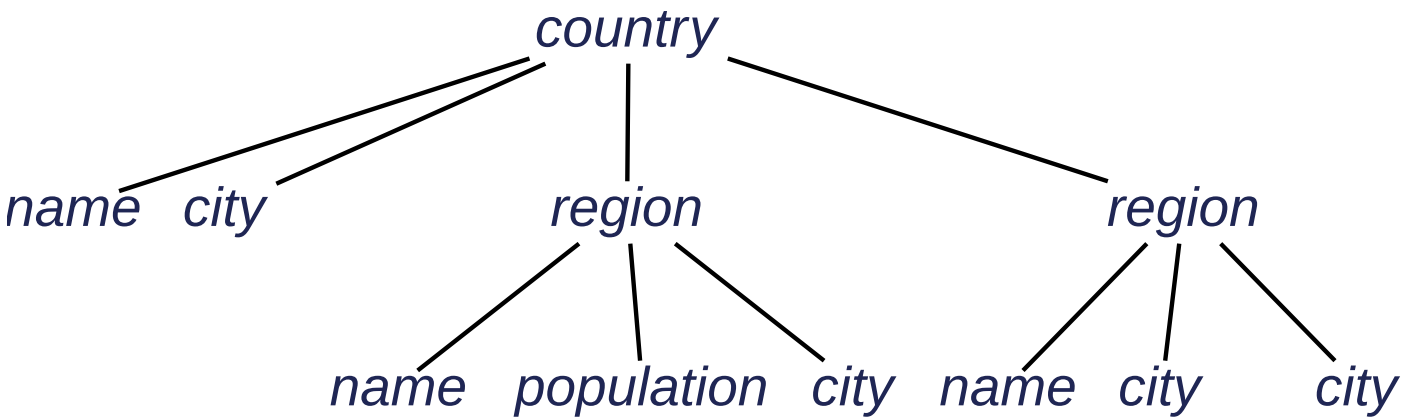
Tree Representation

- Standard abstraction
- Example:



Tree Representation

- Standard abstraction
- Example:



- We consider finite, ordered, unranked trees, over some alphabet Σ
- Text values (and attributes) are omitted

Schema Restrictions

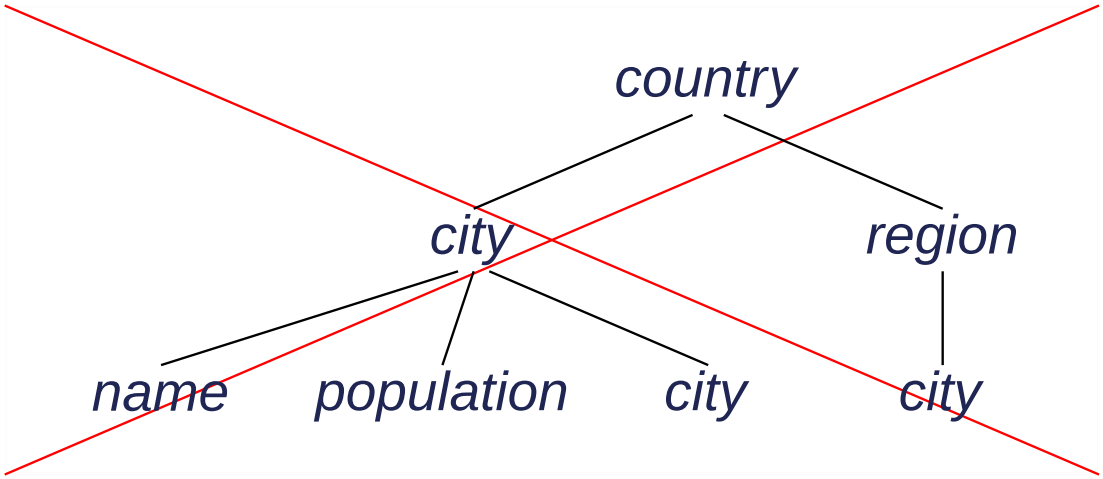
- Schemas are used to describe sets of valid trees
- Available in most XML applications
- Standard languages: DTD, W3C XML Schema, Relax NG, ...
- Example of DTD over Σ:

$$
\begin{array}{lcl}
\mathit{country} & \rightarrow & \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} & \rightarrow & \mathit{name} \cdot (\mathit{population} + \epsilon) \cdot \mathit{city}^* \\
\mathit{name} & \rightarrow & \epsilon \\
\mathit{city} & \rightarrow & \epsilon \\
\mathit{population} & \rightarrow & \epsilon
\end{array}
$$

Schema Restrictions

- Schemas are used to describe sets of valid trees
- Available in most XML applications
- Standard languages: DTD, W3C XML Schema, Relax NG, ...
- Example of DTD over Σ:

$$
\begin{aligned}
\mathit{country} &\rightarrow \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} &\rightarrow \mathit{name} \cdot (\mathit{population} + \epsilon) \cdot \mathit{city}^* \\
\mathit{name} &\rightarrow \epsilon \\
\mathit{city} &\rightarrow \epsilon \\
\mathit{population} &\rightarrow \epsilon
\end{aligned}
$$



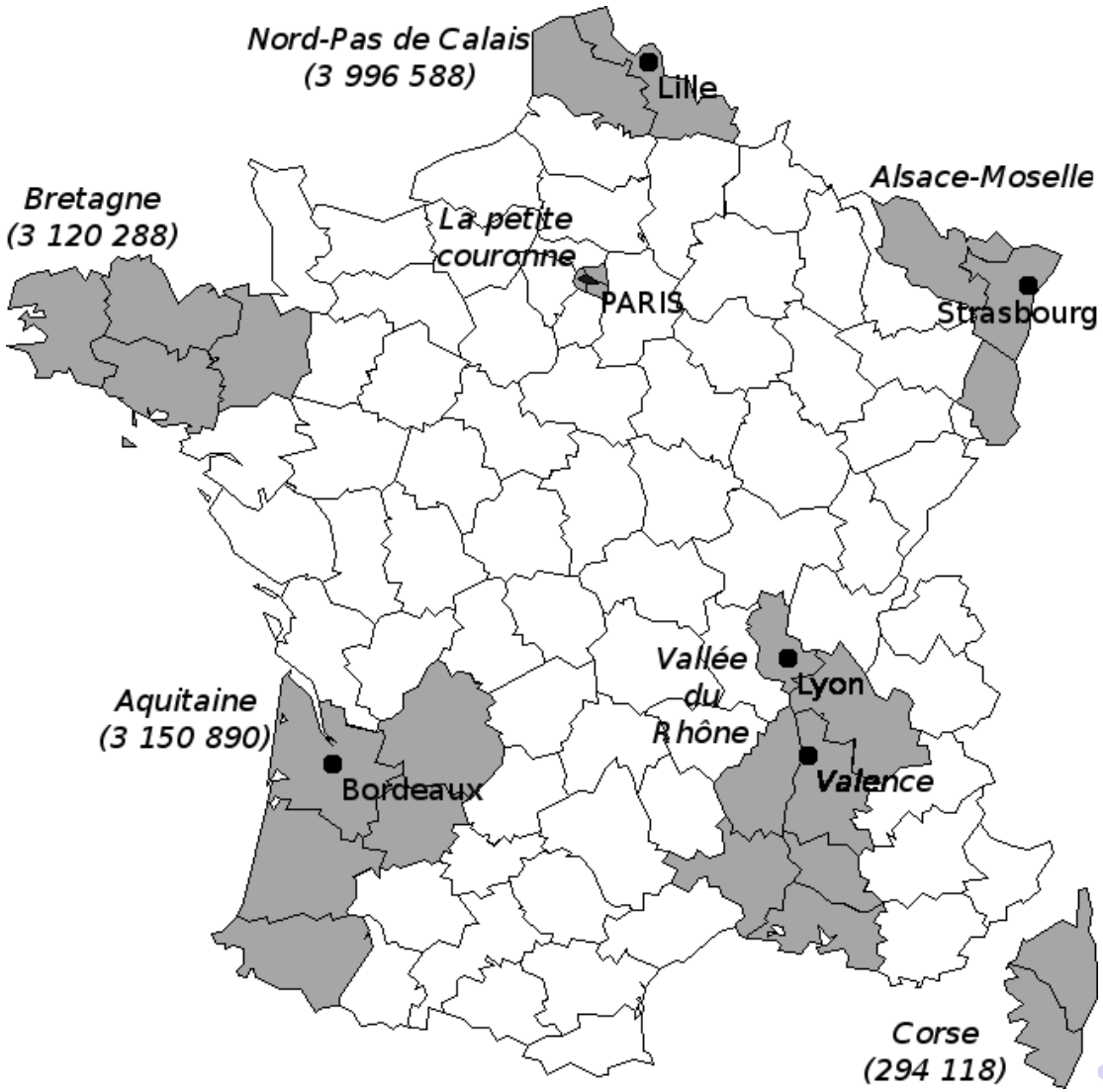
Invalid tree

Queries

- We restrict ourselves to *monadic* node-selecting queries, i.e., functions that selects sets of nodes in trees: $\mathcal{Q}(t) \subseteq nod(t)$.
- XPath:
  - Standard language for node-selecting queries over XML trees
  - Used by other XML standards
  - Expressiveness: first-order logic over unranked trees [Marx, 2005]
  - Example: $//region[population]/name$
- Richer formalisms:
  - Monadic Datalog [Gottlob & Koch, 2004]
  - Tree automata [Comon et al., 2007]
  - Expressiveness: monadic second-order logic

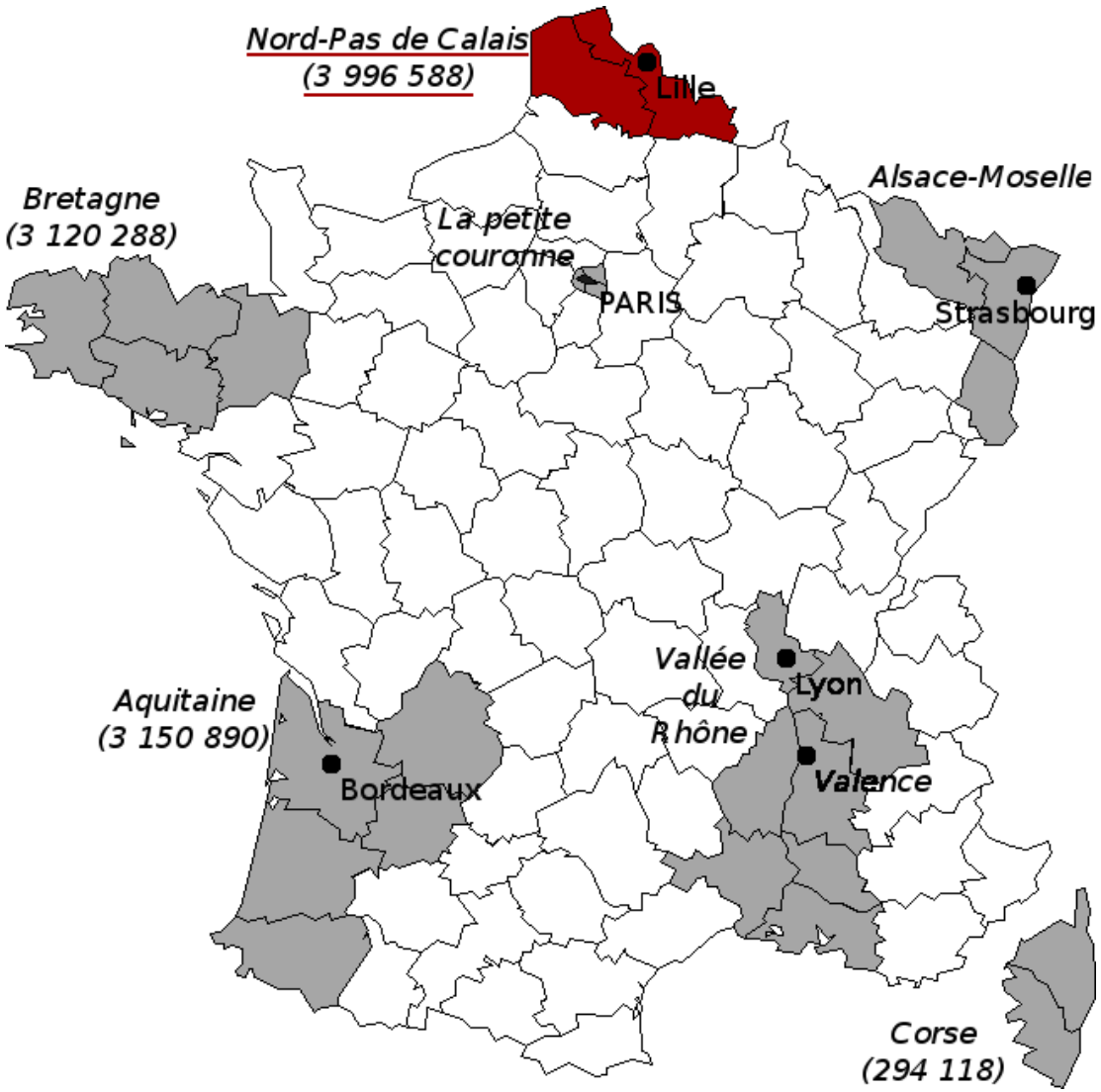
Supervised Query Induction

What are the regions whose population is known?



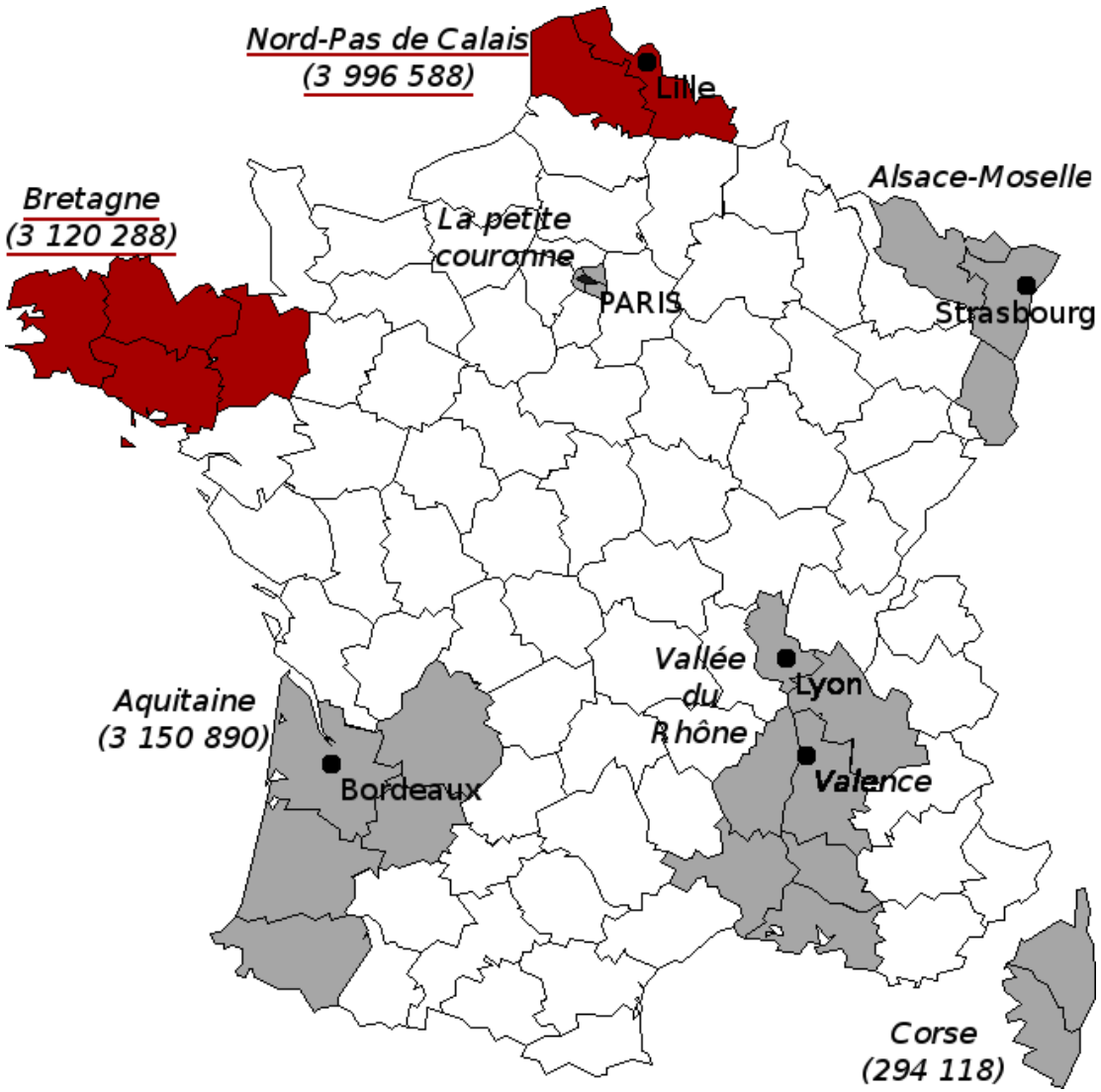
Supervised Query Induction

Select one example region



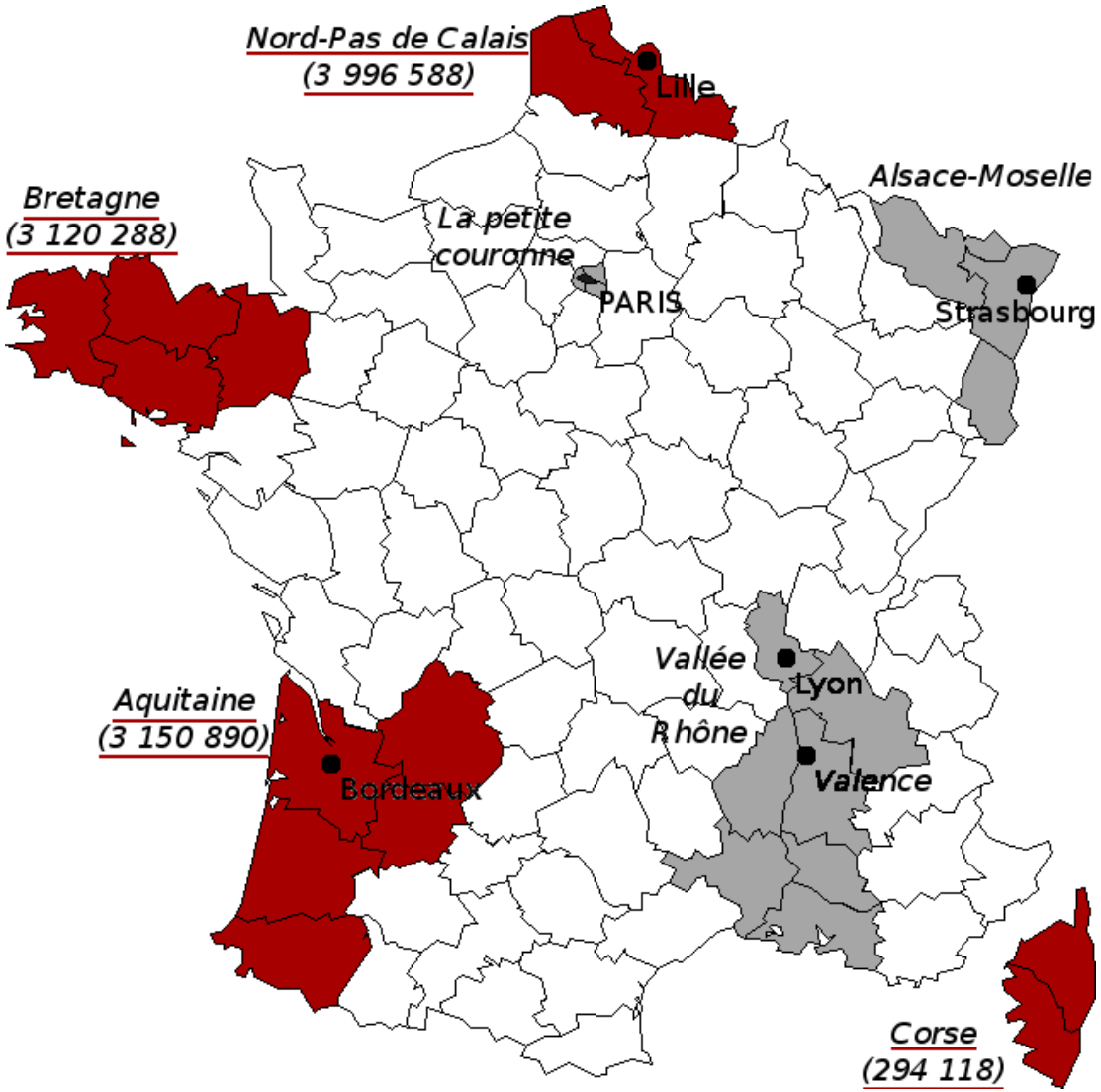
Supervised Query Induction

Select one other example region



Supervised Query Induction

The system will automatically infer the query



Outline

1 XML, Schemas and Queries

2 Tree Automata

3 Schema-Guided Pruning

4 Stable Queries

5 Learnability Results

6 Experiments

Deterministic Tree Automata for Unranked Trees

- Finite state machines for dealing with sets of trees
- Models:
  - Ranked tree automata over binary encodings of unranked trees
  - Hedge automata [Brüggemann-Klein et al., 2001]
- Deterministic stepwise tree automata [Carme et al., 2004]
  - Bottom-up deterministic tree automata over Curryfied encodings
  - Best-suited for schema-guided query induction
- Can express schemas: non-trivial conversion

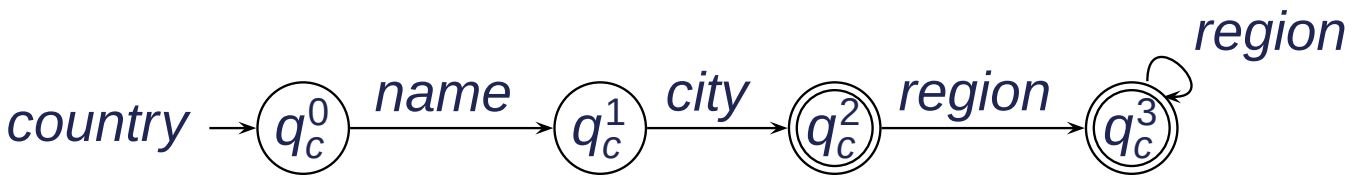
Deterministic DTDs

$$
\begin{array}{lcl}
\mathit{country} & \rightarrow & \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} & \rightarrow & \mathit{name} \cdot (\mathit{population} + \epsilon) \cdot \mathit{city}^* \\
\mathit{name} & \rightarrow & \epsilon \\
\mathit{city} & \rightarrow & \epsilon \\
\mathit{population} & \rightarrow & \epsilon
\end{array}
$$

DTD D

Deterministic DTDs

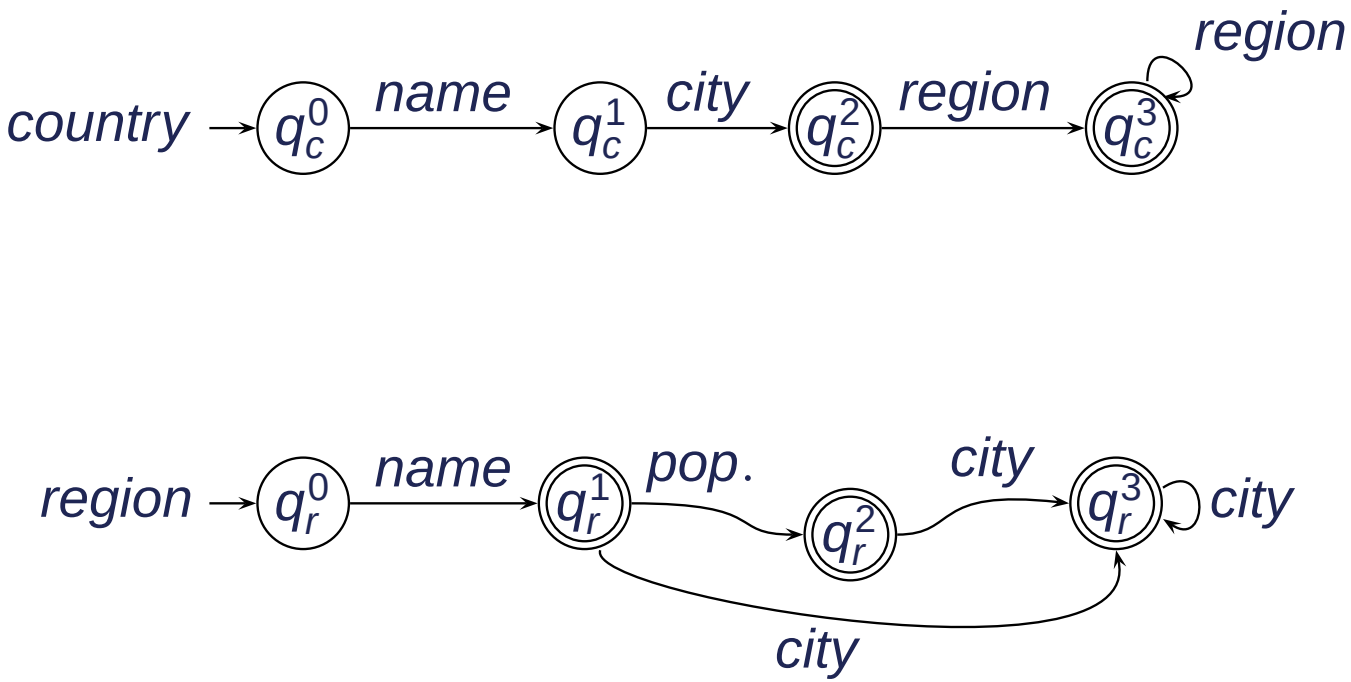
$$
\begin{aligned}
\textit{country} &\rightarrow \textit{name} \cdot \textit{city} \cdot \textit{region}^* \\
\textit{region} &\rightarrow \textit{name} \cdot (\textit{population} + \epsilon) \cdot \textit{city}^* \\
\textit{name} &\rightarrow \epsilon \\
\textit{city} &\rightarrow \epsilon \\
\textit{population} &\rightarrow \epsilon
\end{aligned}
$$



Glushkov automata: $\mathcal{O}(|\Sigma| \times |D|)$

Deterministic DTDs

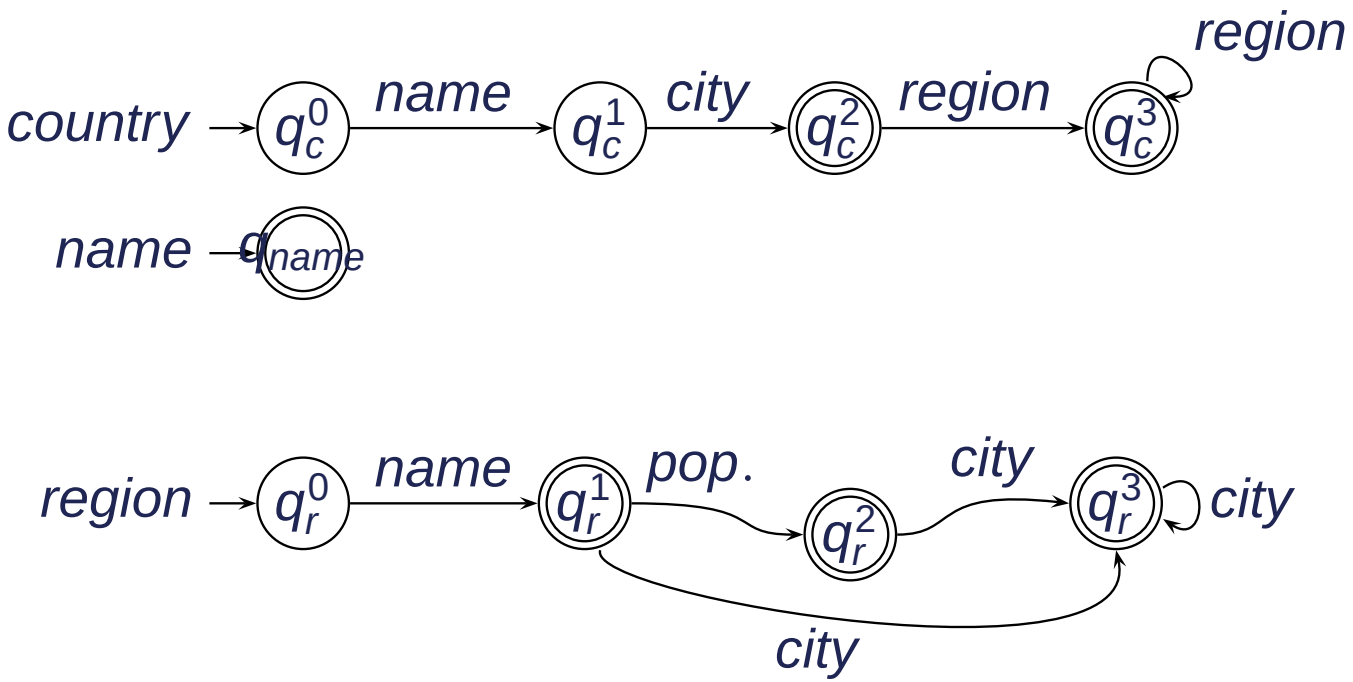
$$
\begin{aligned}
\textit{country} &\rightarrow \textit{name} \cdot \textit{city} \cdot \textit{region}^* \\
\textit{region} &\rightarrow \textit{name} \cdot (\textit{population} + \epsilon) \cdot \textit{city}^* \\
\textit{name} &\rightarrow \epsilon \\
\textit{city} &\rightarrow \epsilon \\
\textit{population} &\rightarrow \epsilon
\end{aligned}
$$



Glushkov automata: $\mathcal{O}(|\Sigma| \times |D|)$

Deterministic DTDs

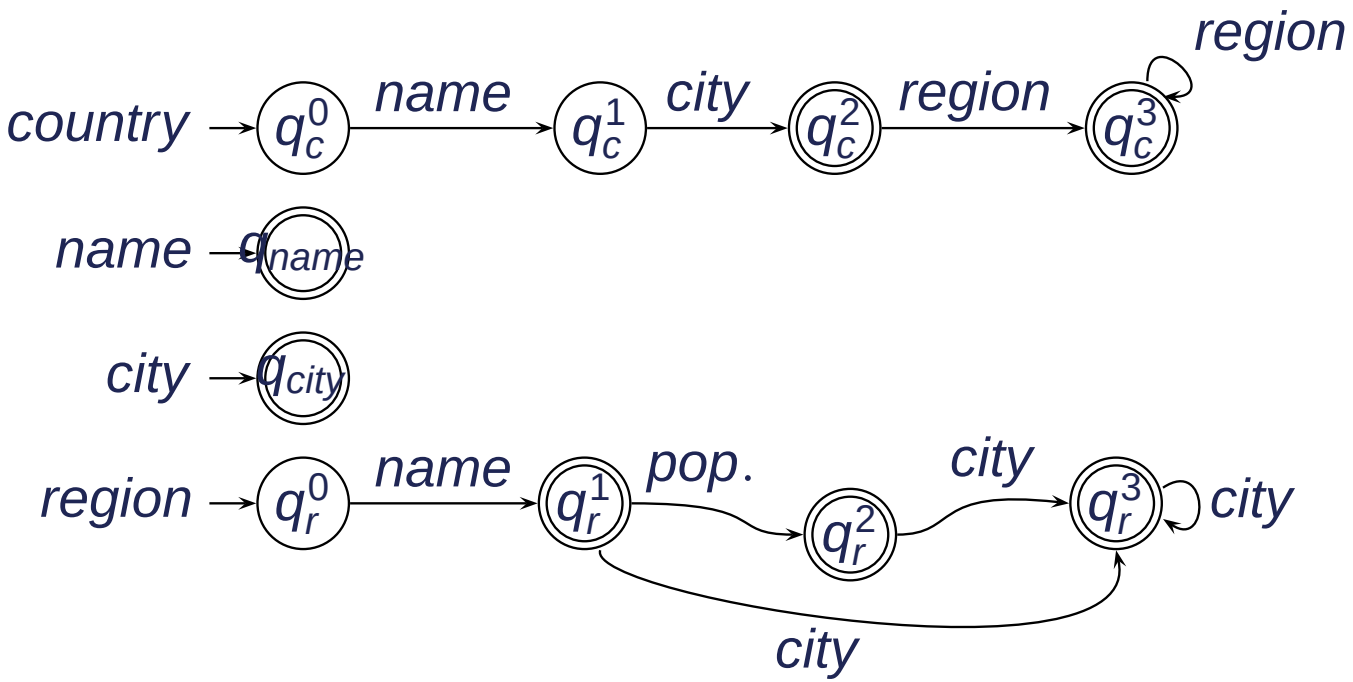
$$
\begin{aligned}
\textit{country} &\rightarrow \textit{name} \cdot \textit{city} \cdot \textit{region}^* \\
\textit{region} &\rightarrow \textit{name} \cdot (\textit{population} + \epsilon) \cdot \textit{city}^* \\
\textit{name} &\rightarrow \epsilon \\
\textit{city} &\rightarrow \epsilon \\
\textit{population} &\rightarrow \epsilon
\end{aligned}
$$



Glushkov automata: $\mathcal{O}(|\Sigma| \times |D|)$

Deterministic DTDs

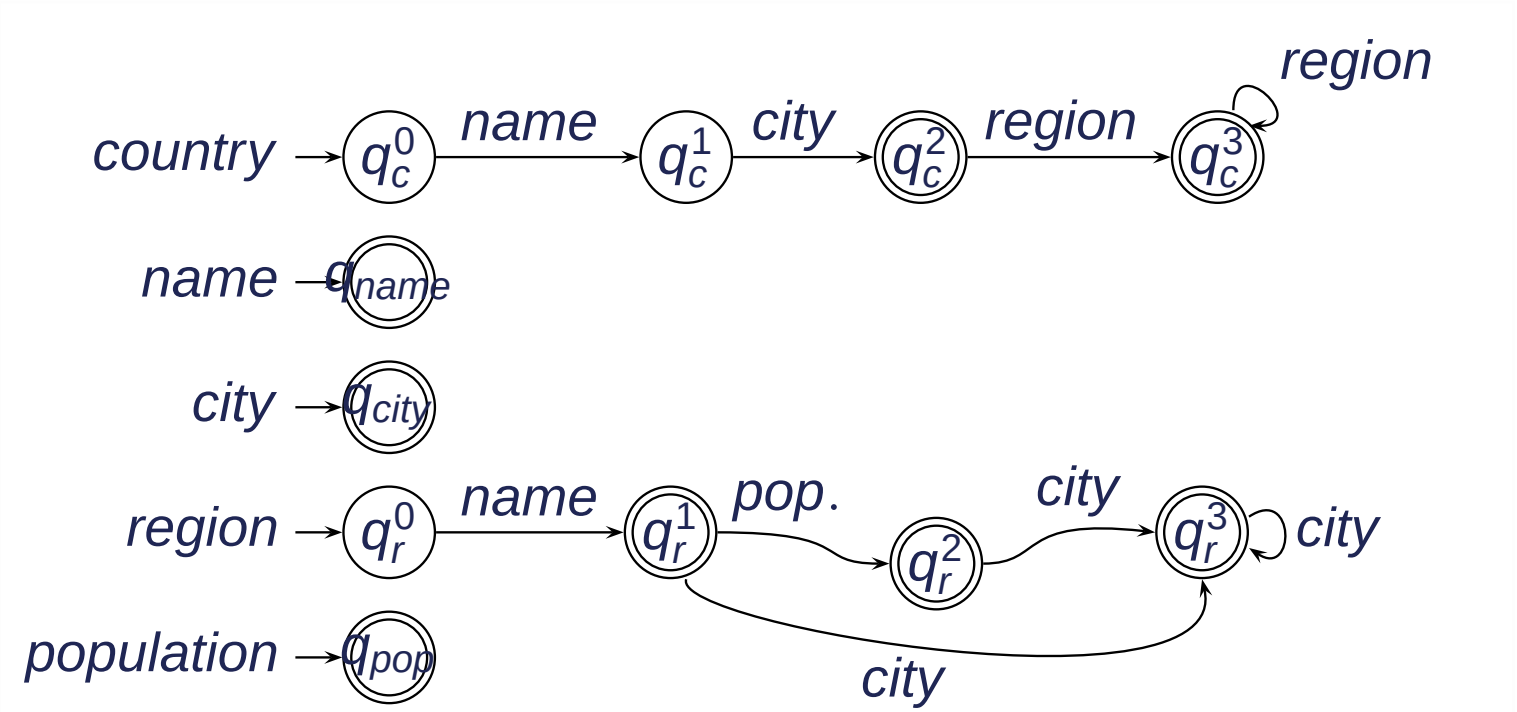
$$
\begin{aligned}
\textit{country} &\rightarrow \textit{name} \cdot \textit{city} \cdot \textit{region}^* \\
\textit{region} &\rightarrow \textit{name} \cdot (\textit{population} + \epsilon) \cdot \textit{city}^* \\
\textit{name} &\rightarrow \epsilon \\
\textit{city} &\rightarrow \epsilon \\
\textit{population} &\rightarrow \epsilon
\end{aligned}
$$



Glushkov automata: $\mathcal{O}(|\Sigma| \times |D|)$

Deterministic DTDs

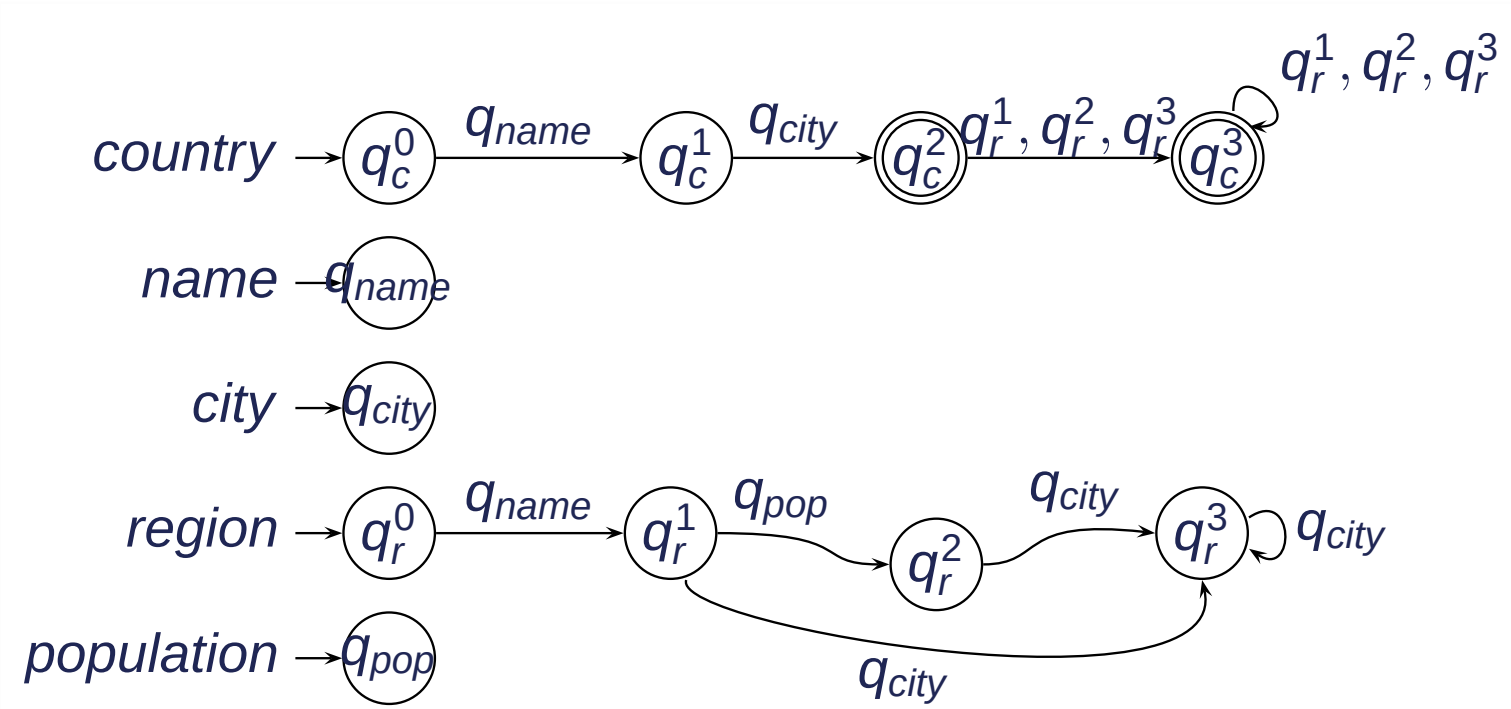
$$
\begin{aligned}
\mathit{country} &\rightarrow \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} &\rightarrow \mathit{name} \cdot (\mathit{population} + \epsilon) \cdot \mathit{city}^* \\
\mathit{name} &\rightarrow \epsilon \\
\mathit{city} &\rightarrow \epsilon \\
\mathit{population} &\rightarrow \epsilon
\end{aligned}
$$



Glushkov automata: $\mathcal{O}(|\Sigma| \times |D|)$

Deterministic DTDs

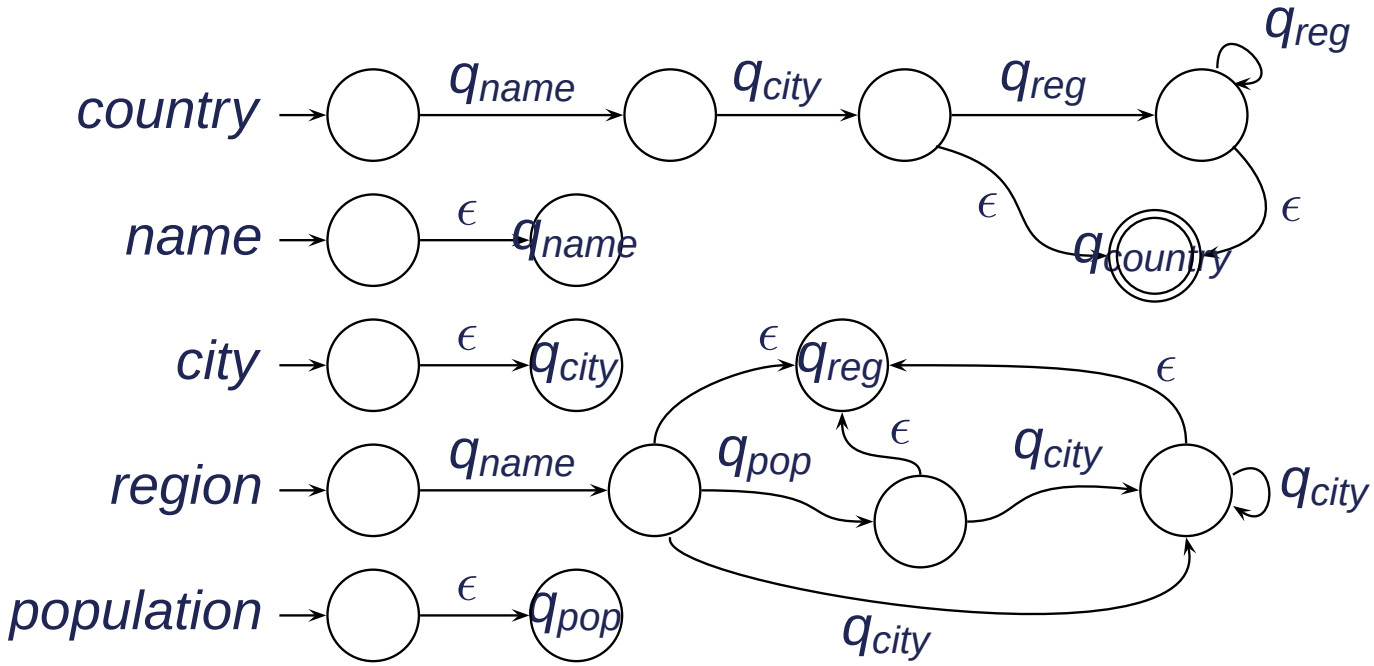
$$
\begin{aligned}
\mathit{country} &\rightarrow \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} &\rightarrow \mathit{name} \cdot (\mathit{population} + \epsilon) \cdot \mathit{city}^* \\
\mathit{name} &\rightarrow \epsilon \\
\mathit{city} &\rightarrow \epsilon \\
\mathit{population} &\rightarrow \epsilon
\end{aligned}
$$



Stepwise tree automaton: $\mathcal{O}(|\Sigma| \times |D|^2)$

Deterministic DTDs

$$
\begin{aligned}
\mathit{country} &\rightarrow \mathit{name} \cdot \mathit{city} \cdot \mathit{region}^* \\
\mathit{region} &\rightarrow \mathit{name} \cdot (\mathit{population} + \epsilon) \cdot \mathit{city}^* \\
\mathit{name} &\rightarrow \epsilon \\
\mathit{city} &\rightarrow \epsilon \\
\mathit{population} &\rightarrow \epsilon
\end{aligned}
$$



Factorized tree automaton: $\mathcal{O}(|\Sigma| \times |D|)$

Efficient Inclusion Checking

Theorem

Let A be a tree automaton for unranked trees (stepwise, first-child next-sibling, or hedge), and D a deterministic DTD, both over Σ. Inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(D)$ can be checked in time $O(|A| \times |\Sigma| \times |D|)$.

- Non-trivial algorithm:
  - Avoid complement automaton computation
  - Dectect inclusion failure on the pruduct automaton
  - Only accessible part is materialized
- Optimizations:
  - Early detection of inclusion failure
  - Incremental w.r.t. adding ϵ-rules to A (stepwise)

Monadic Queries

- Regular languages of annotated trees, i.e., over $\Sigma \times \{0, 1\}$
  - Functional: no contradictory annotations
  - Non-null: at least one positive annotation
- Node-selecting tree transducers (NSTTs) [Carme et al., 2007]:
  - Stepwise tree automata for monadic node-selecting queries
  - Functionality and non-nullity checked in polynomial time
  - Query answering done in combined linear time

Regular Query Induction

- Supervised learning problem
- Based on tree automata inference ($\mathcal{RPNI}$-like algorithm)
- Variant of Gold's identification in the limit model
  [Gold, 1978; de la Higuera, 1997]
- Regular monadic queries represented by NSTTs are indentifiable from completely annotated examples [Carme et al., 2007]

Outline

1 XML, Schemas and Queries

2 Tree Automata

3 Schema-Guided Pruning

4 Stable Queries

5 Learnability Results

6 Experiments

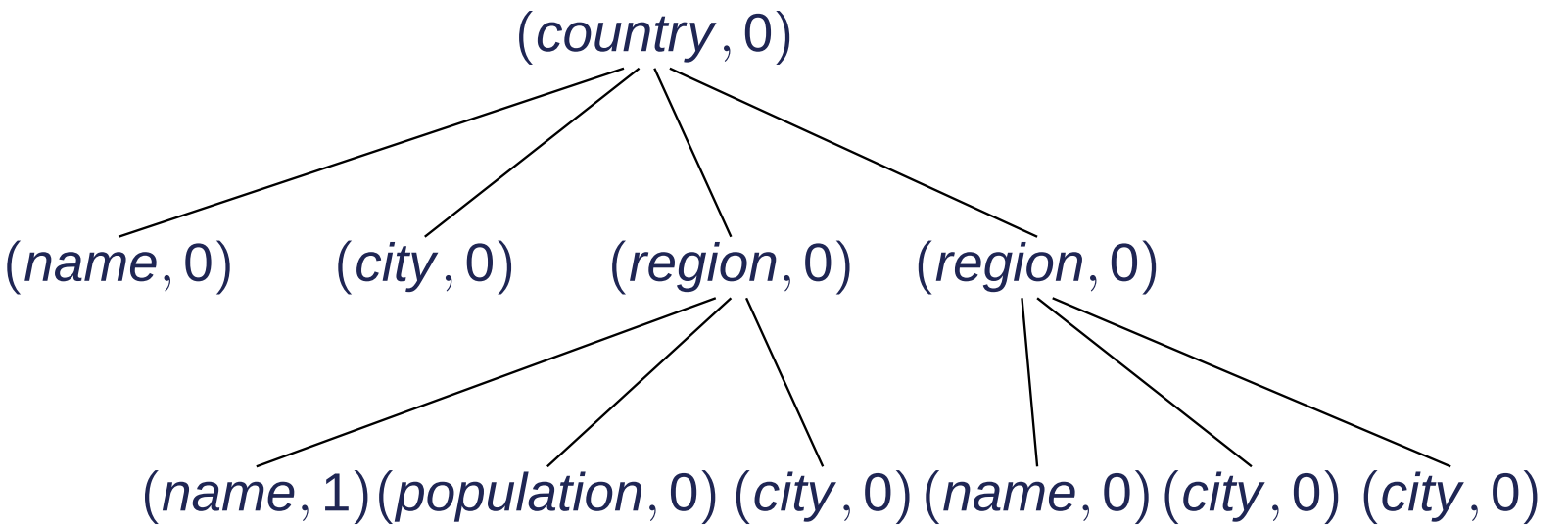
Why Pruning?

- Practical constraint: learn from partially annotated examples
- State-of-the-art systems use pruning in order to success [Carme et al., 2007; Raeymaekers, 2008]
- Basic idea: try to keep useful parts of tree examples that justify node selection, while removing irrelevant parts
- What can be learned depends on the fixed pruning strategy

Pruning Strategy

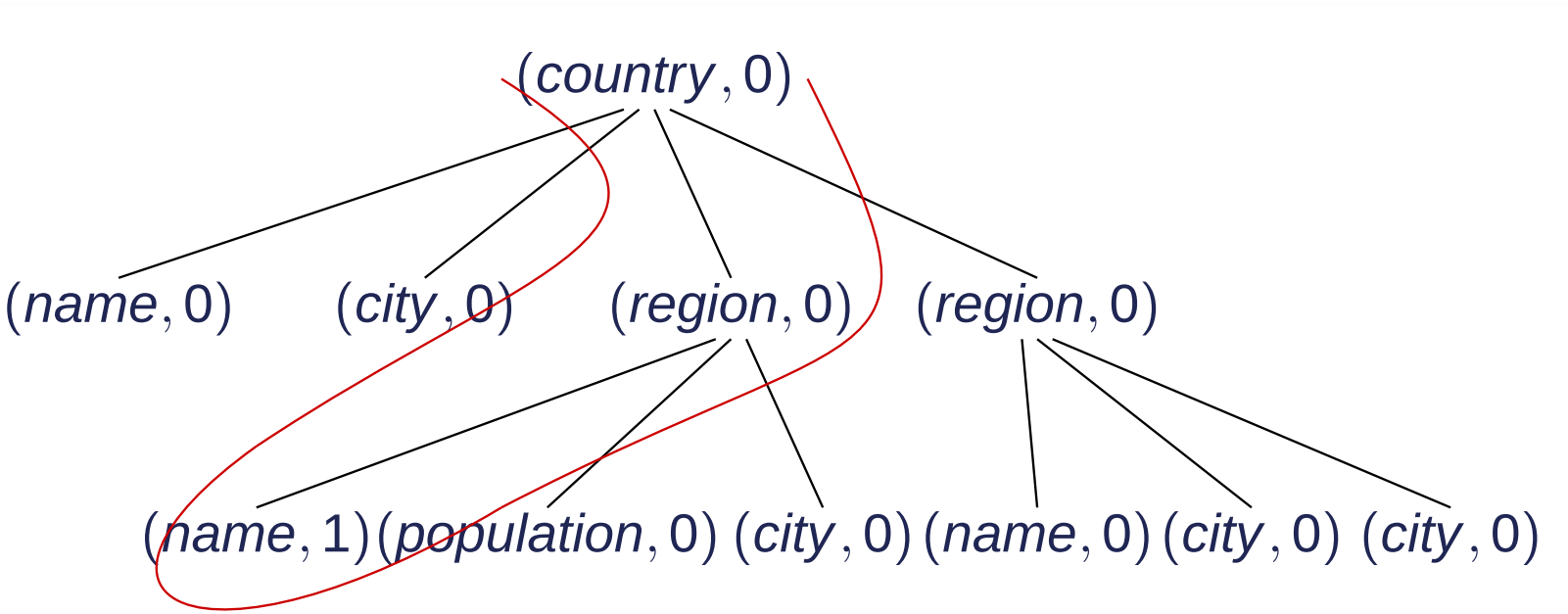
- Replace whole subtrees by symbols in some alphabet Γ
- Function $\mathcal{P}$ from trees over $\Sigma \times \{0,1\}$ to trees over $(\Sigma \times \{0,1\}) \uplus \Gamma$
  - Preserve at least selected nodes and their path from the root
  - Do not change label and annotation of preserved nodes
  - Expressible by some monadic query
- Schema-guided: let $\Gamma = \mathit{sta}(D)$

Path-Only Pruning



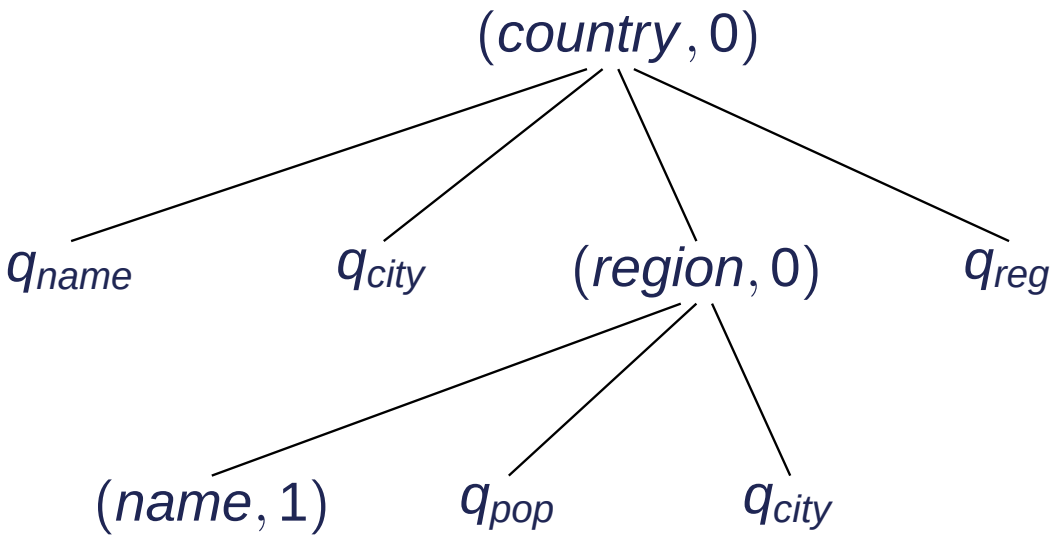
Annotated tree $t * \beta$

Path-Only Pruning

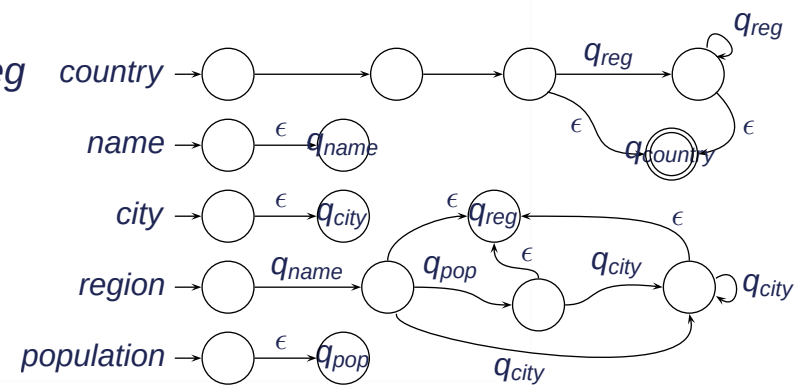


Annotated tree $t * \beta$

Path-Only Pruning

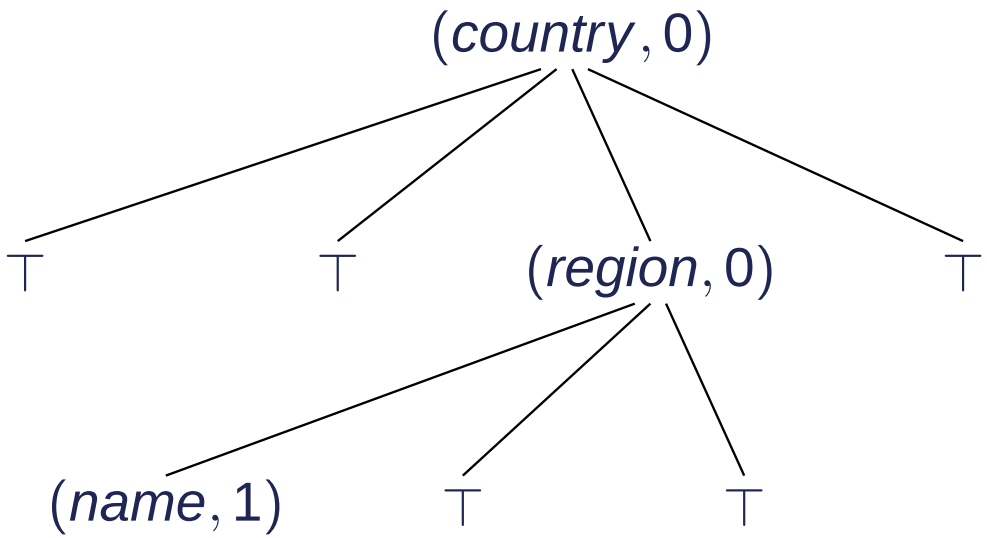


D-pruned tree $\mathcal{P}^D_{\text{only}}(t * \beta)$

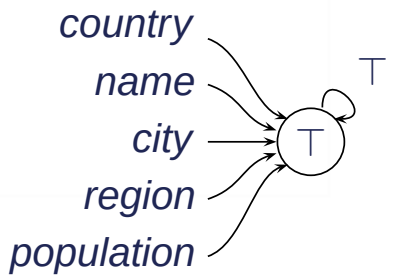


Schema D

Path-Only Pruning



$\mathcal{U}$-pruned tree $\mathcal{P}^{\mathcal{U}}_{\text{only}}(t * \beta)$



Schema $\mathcal{U}$

Outline

1 XML, Schemas and Queries

2 Tree Automata

3 Schema-Guided Pruning

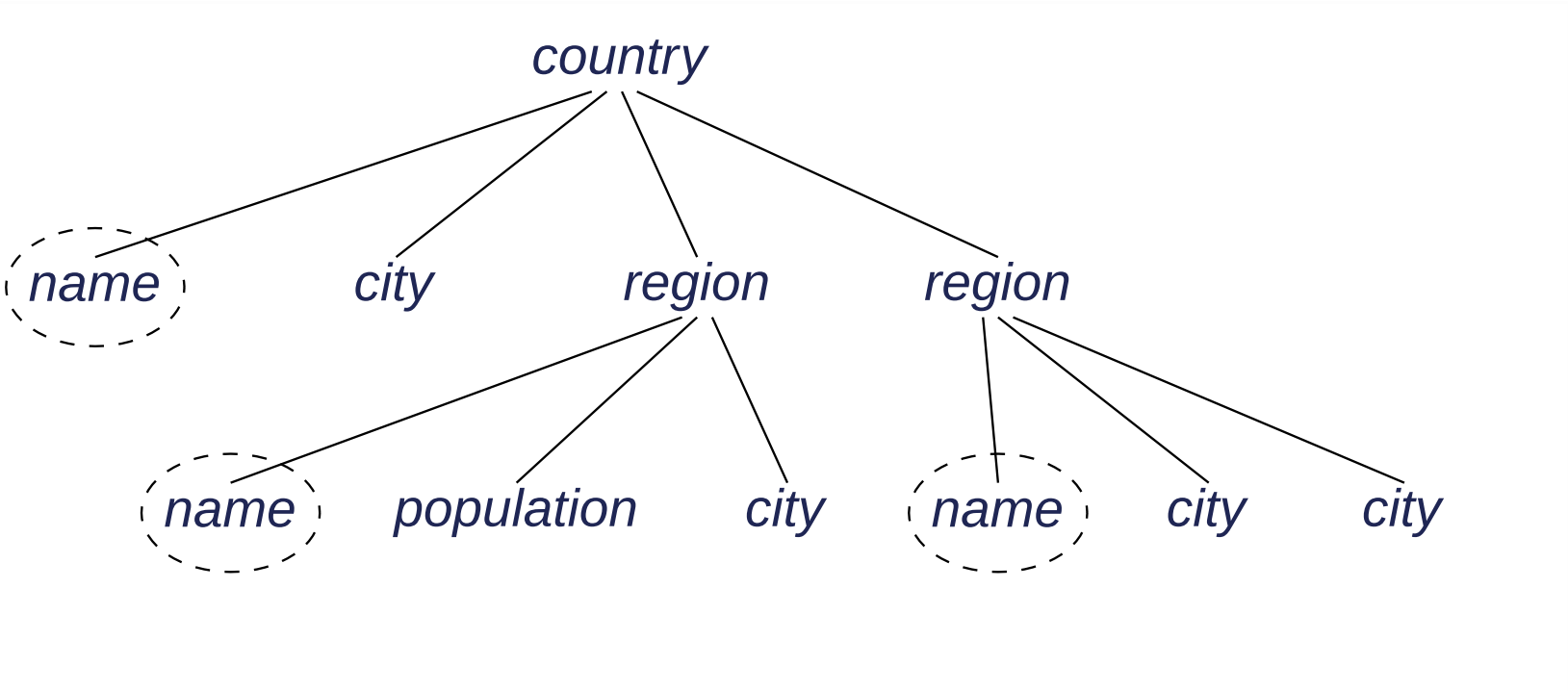
4 Stable Queries

5 Learnability Results

6 Experiments

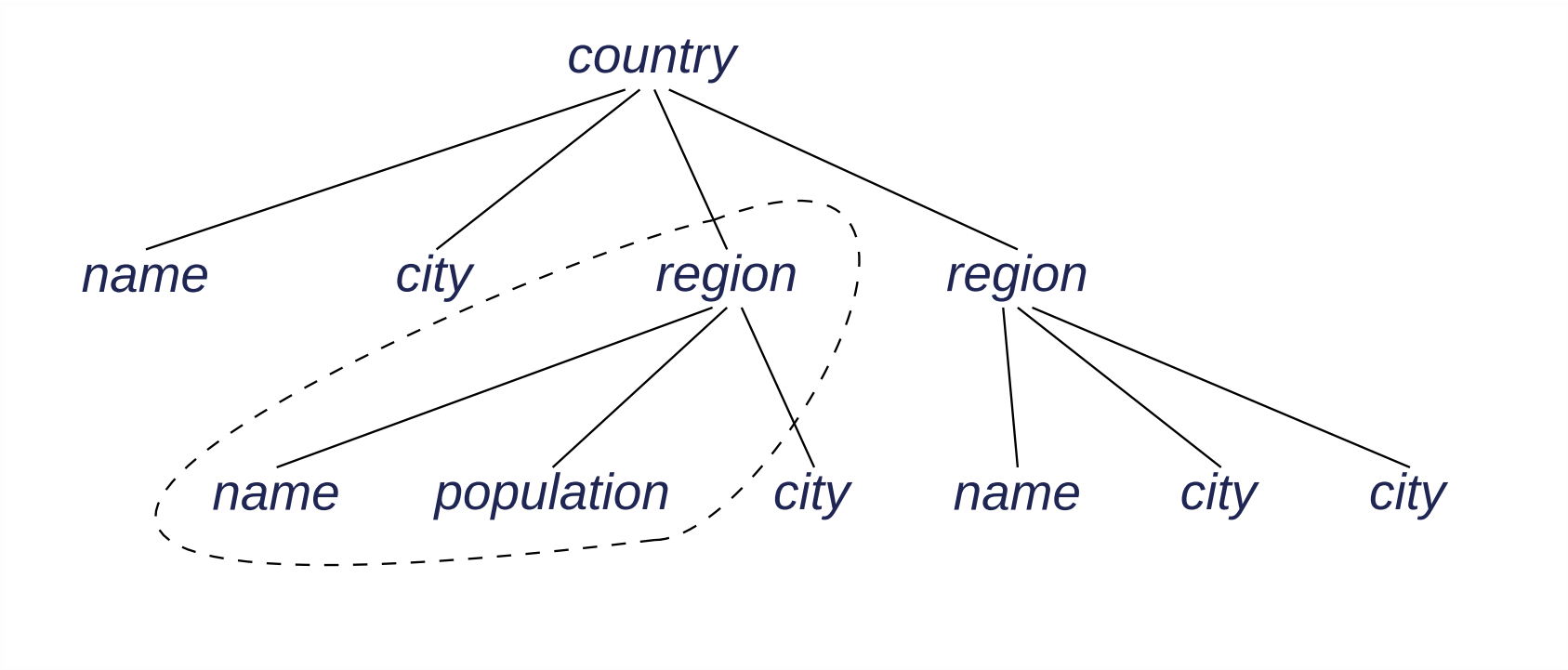
Simple Queries?

Select all names:



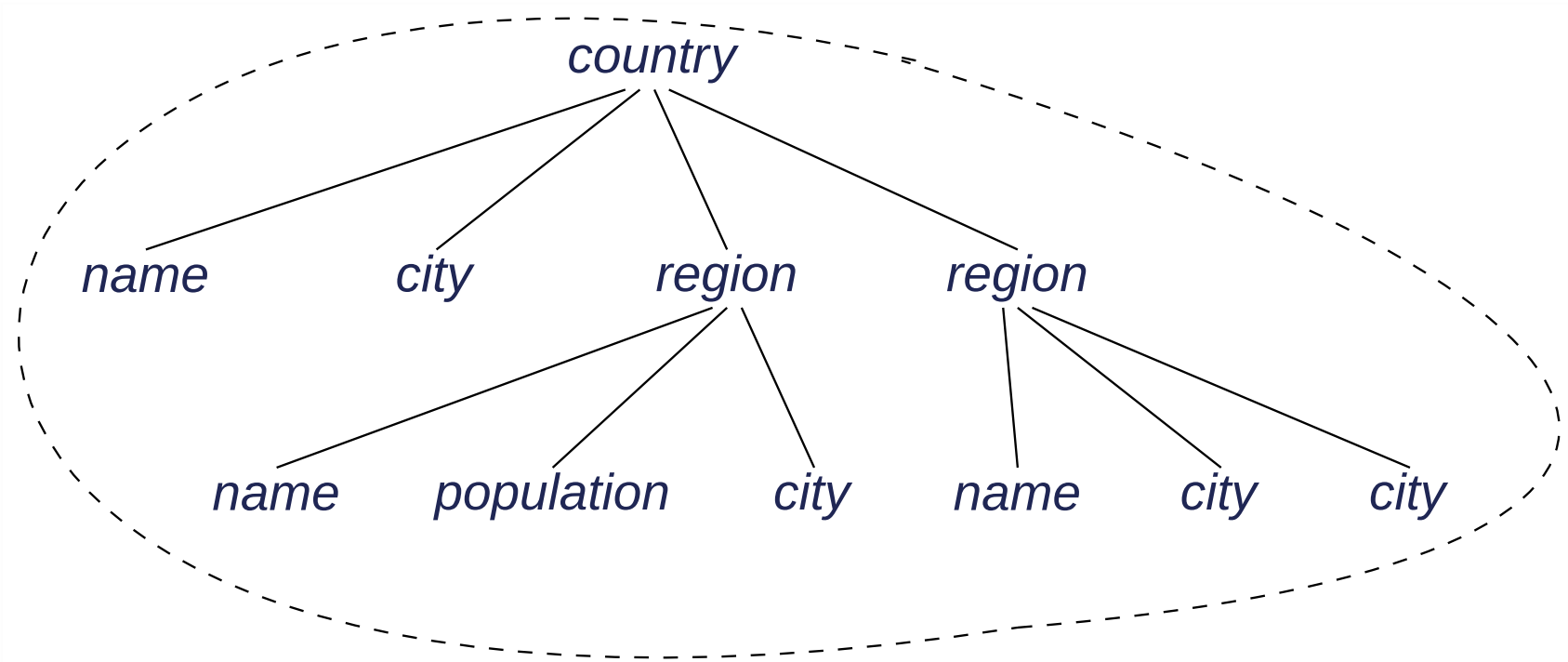
Simple Queries?

Select all regions' names whose population is known:



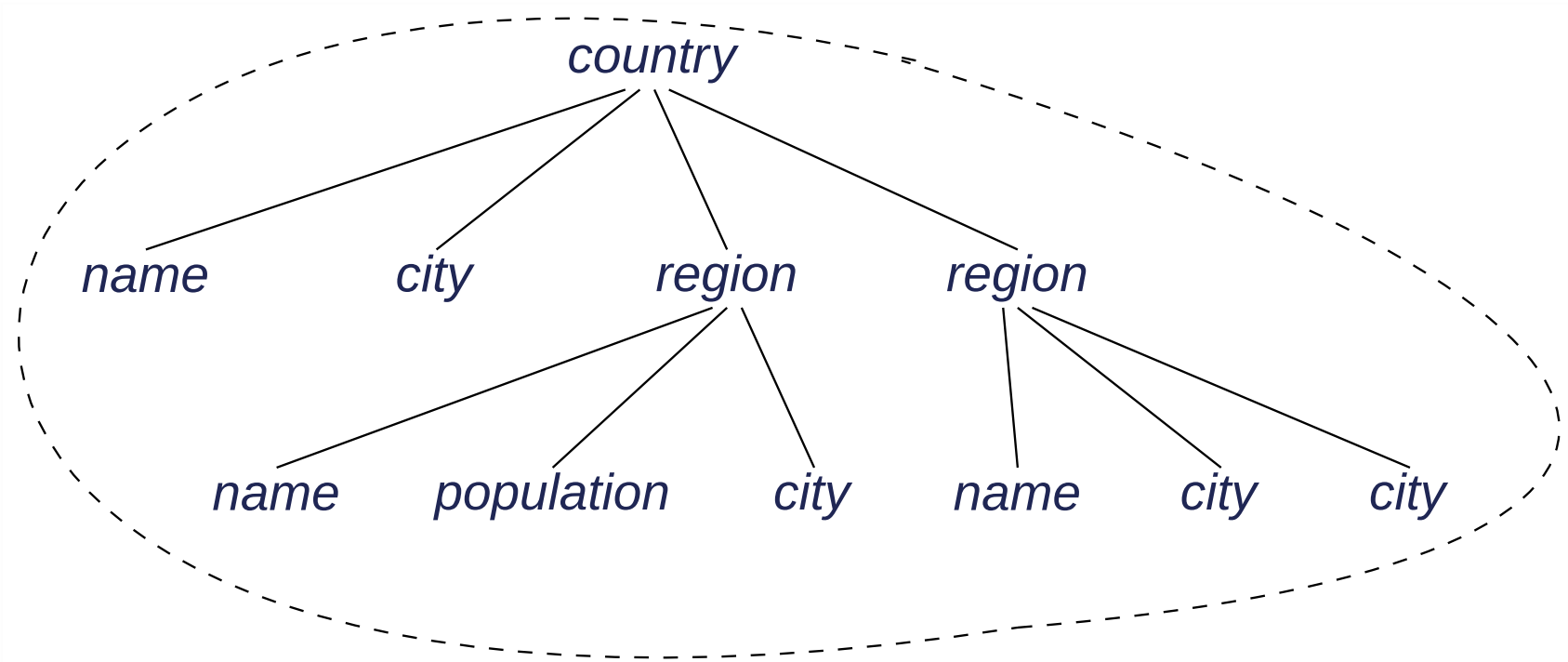
Simple Queries?

Select country's name if all regions have no population:



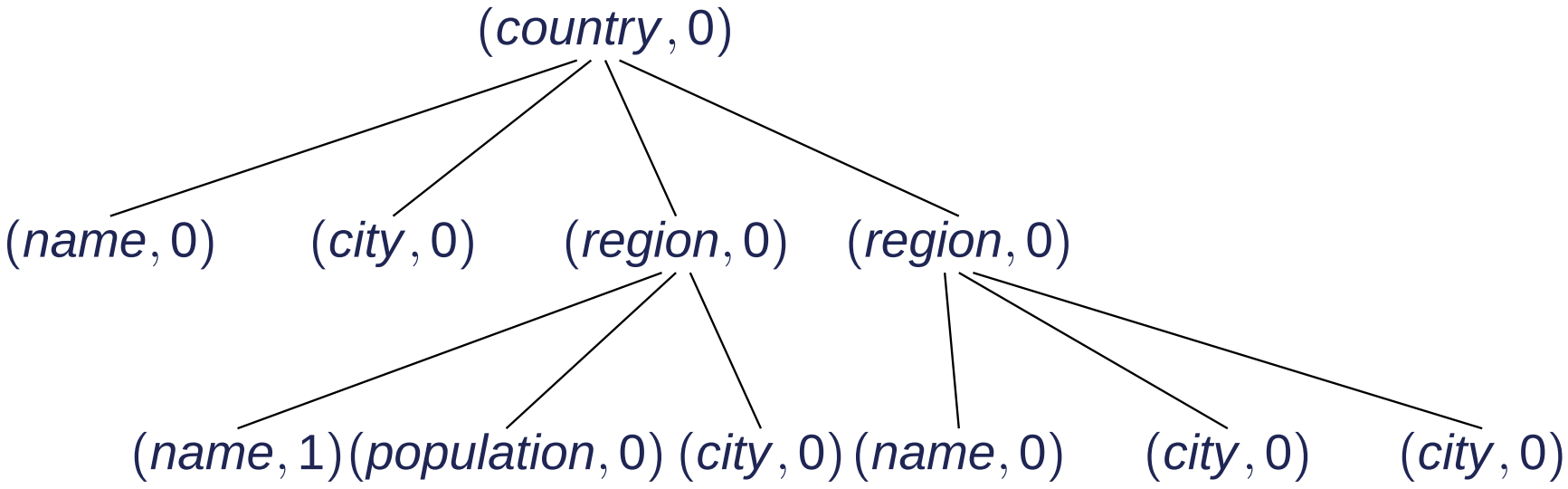
Simple Queries?

Select country's name if all regions have no population:



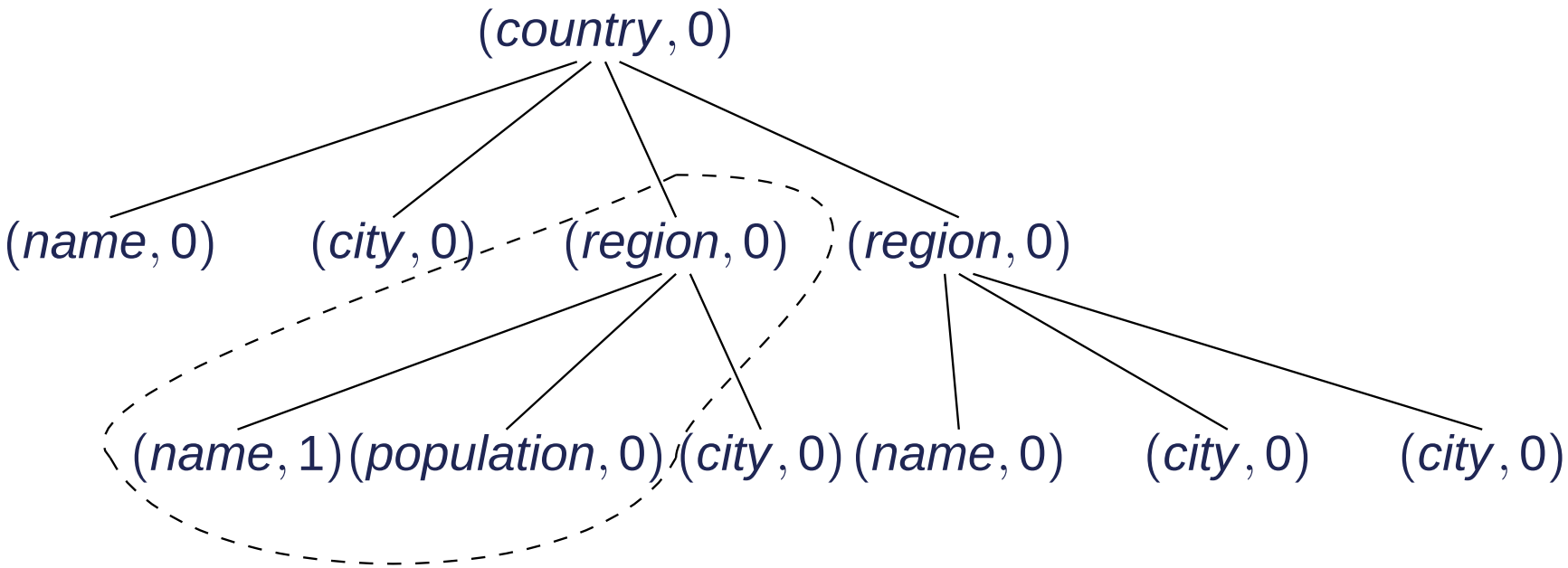
Conclusion: the larger is the dependency, the more difficult it is.

Insights



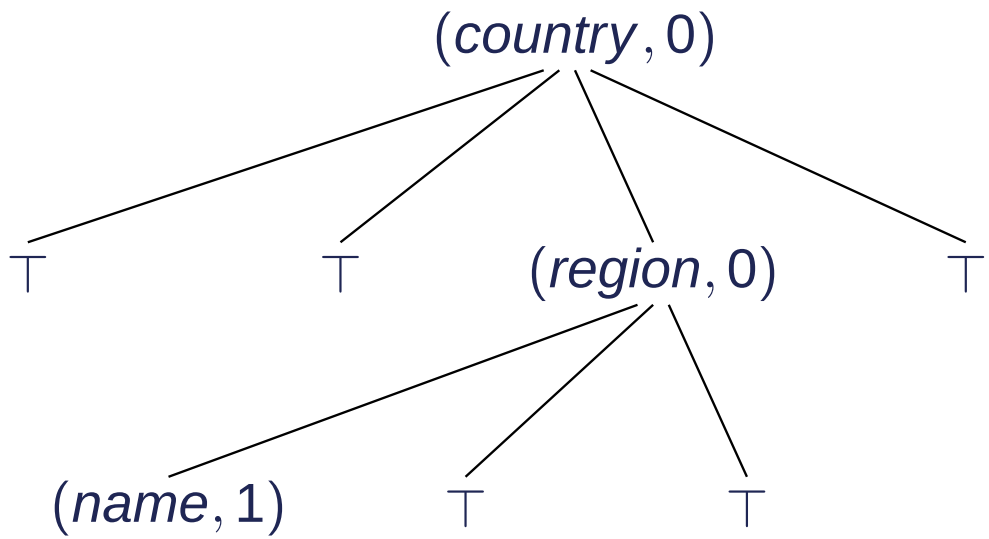
Annotated tree $t * \mathcal{Q}(t)$

Insights



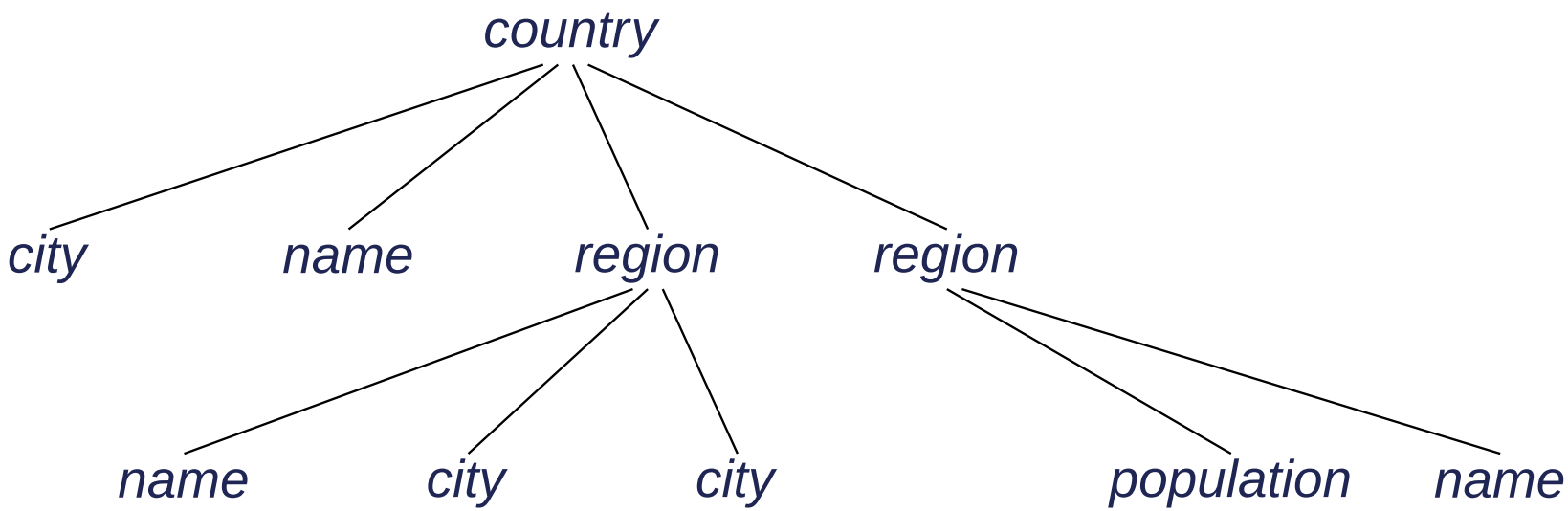
Relevant information for $\mathcal{Q}$

Insights



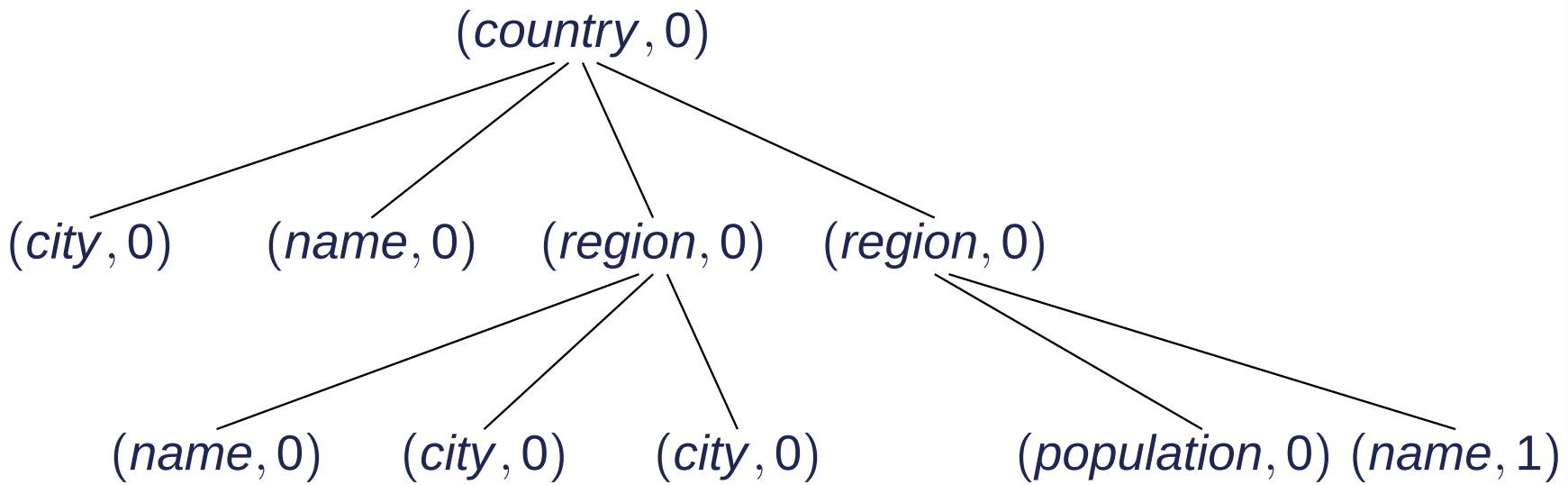
$\mathcal{U}$-path-only pruning $\mathcal{P}^{\mathcal{U}}_{\text{only}}(t * \mathcal{Q}(t))$

Insights



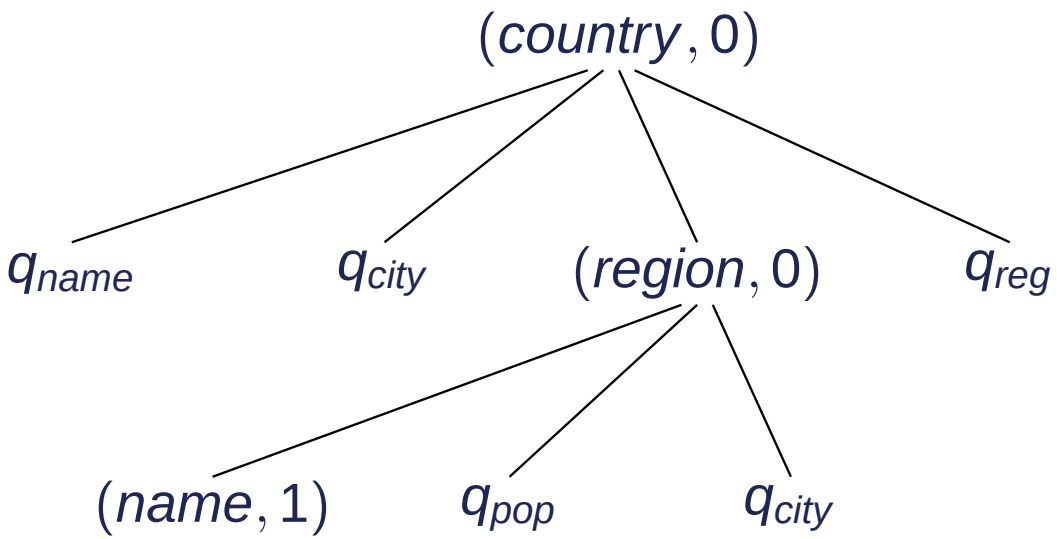
$\mathcal{U}$-completion t'

Insights



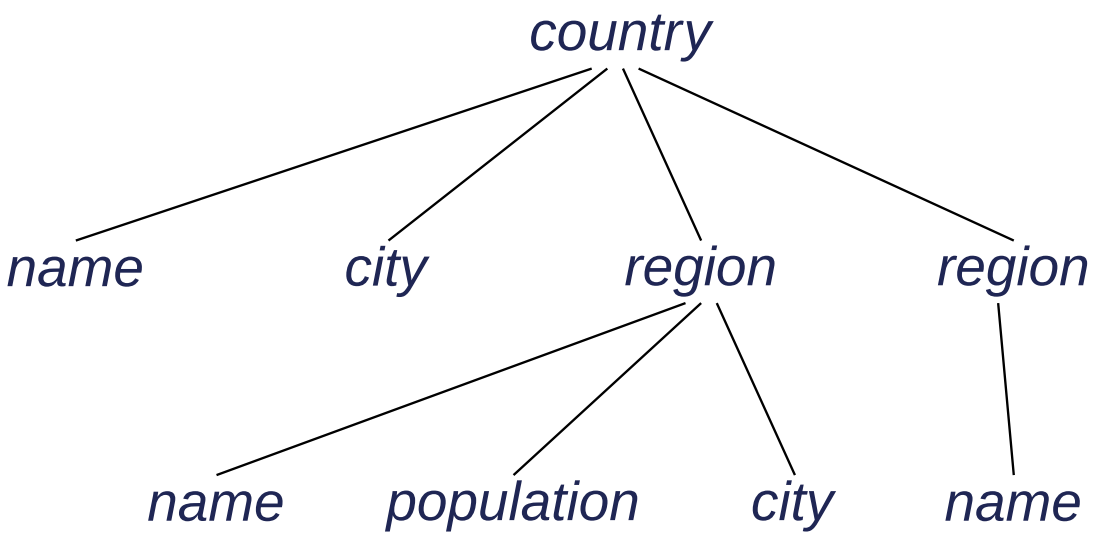
Annotated tree $t' * \mathcal{Q}(t')$

Insights



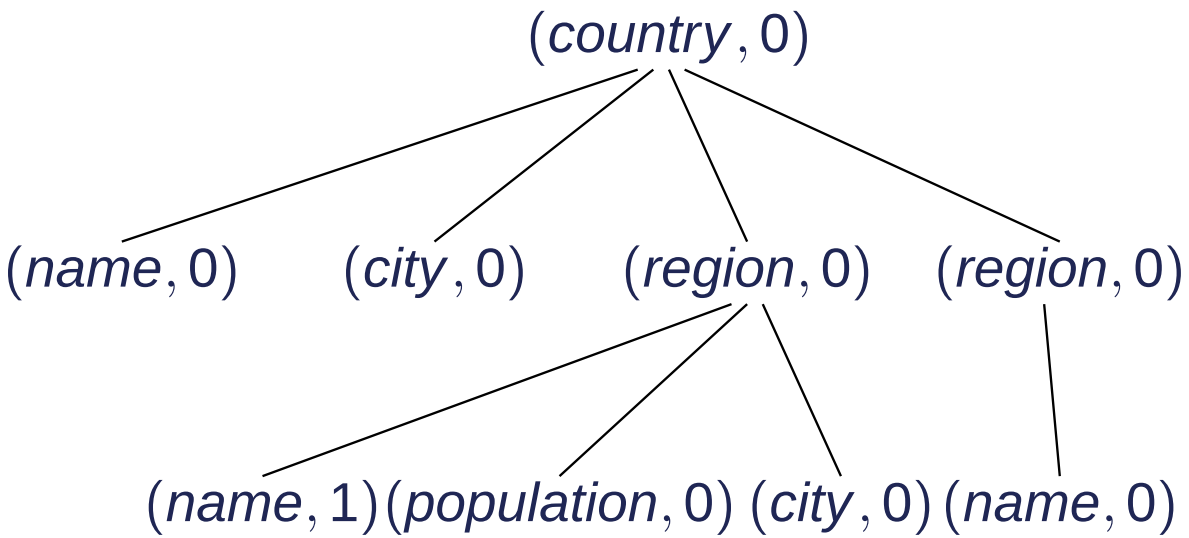
D-path-only pruning $\mathcal{P}^{D}_{\text{only}}(t * \mathcal{Q}(t))$

Insights



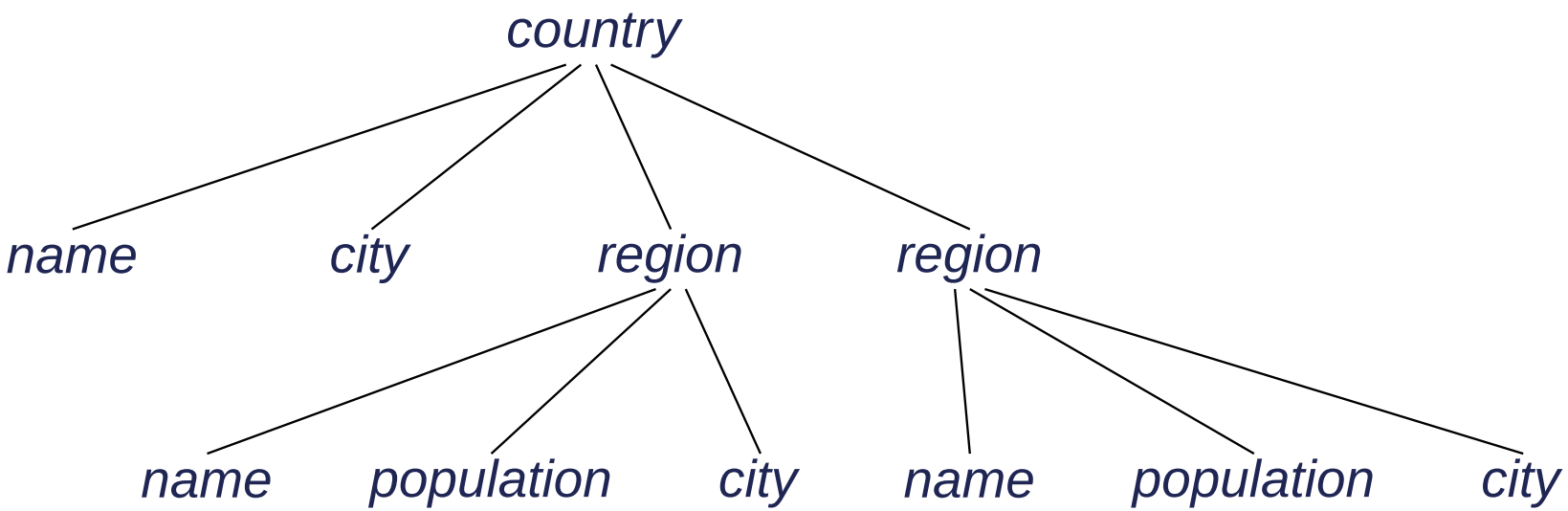
D-completion t_1

Insights



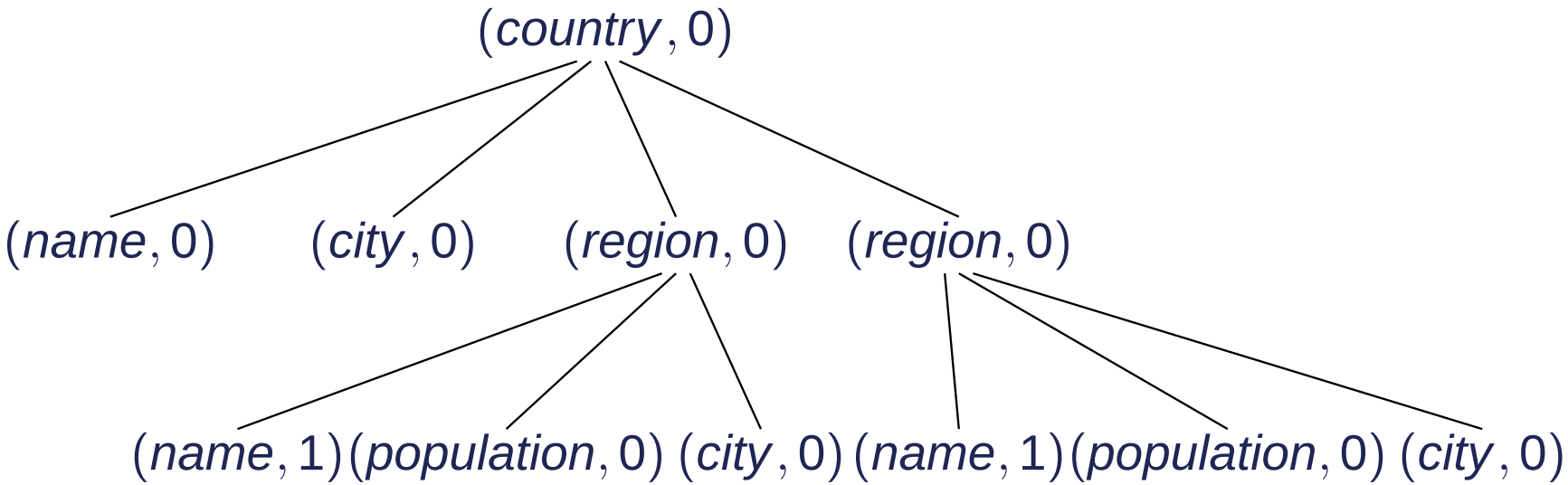
Annotated tree $t_1 * \mathcal{Q}(t_1)$

Insights



D-completion t_3

Insights



Annotated tree $t_3 * \mathcal{Q}(t_3)$

Stability

Definition

A query $\mathcal{Q}$ is stable by a D-pruning $\mathcal{P}$ (or $\mathcal{P}$-stable) if and only if for all $t_0 \in \mathcal{L}(D)$, let $t' = \mathcal{P}(t_0 * \mathcal{Q}(t_0))$:

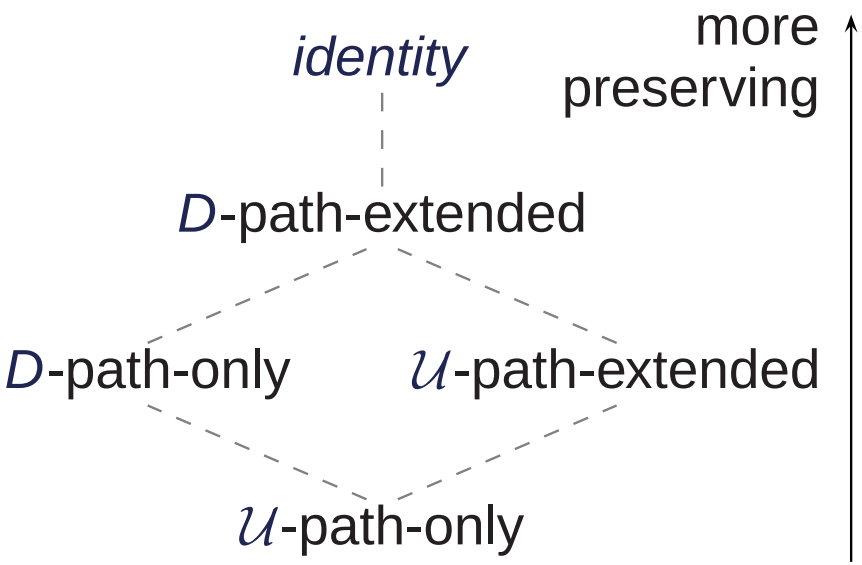
$$\forall t_1 \in \mathit{compl}_D(t'), \forall \nu \big(t'(\nu) \in \Sigma \times \{0,1\}\big), \big(\nu \in \mathcal{Q}(t_1) \Leftrightarrow \nu \in \mathcal{Q}(t_0)\big)$$

Informally stated: for every D-completion of a D-pruned tree, a $\mathcal{P}$-stable query $\mathcal{Q}$ selects exactly the same nodes on the non-pruned part as on the original tree.

Classification of Pruning Strategies

Definition

Let $\mathcal{P}_1$ be a D_1-pruning and $\mathcal{P}_2$ be a D_2-pruning, $\mathcal{P}_1$ is *more preserving* than $\mathcal{P}_2$ if $compl_{D_1}(\mathcal{P}_1(s)) \subseteq compl_{D_2}(\mathcal{P}_2(s))$ for all annotated trees s.



Proposition

Let $\mathcal{Q}$ be a query and $\mathcal{P}_1, \mathcal{P}_2$ be two pruning strategies. If $\mathcal{P}_1$ is more preserving than $\mathcal{P}_2$ and $\mathcal{Q}$ is $\mathcal{P}_2$-stable then $\mathcal{Q}$ is $\mathcal{P}_1$-stable.

Outline

1 XML, Schemas and Queries

2 Tree Automata

3 Schema-Guided Pruning

4 Stable Queries

5 Learnability Results

6 Experiments

Tree Automata for Pruned Trees

- D-functionality: no contradictory annotations for D-pruned trees that share a D-completion
- Monadic queries: regular languages of D-pruned trees
  - D-functional
  - Non-null
- D-NSTTs
  - D-functionality and non-nullity checked in polynomial time
  - Query answering done in combined linear time

Stability versus D-functionality

Lemma

Let $\mathcal{Q}$ be a monadic query, $\mathcal{P}$ be a D-pruning, and L the language of D-pruned trees such that

$$L = \{\mathcal{P}(t * \mathcal{Q}(t)) \mid t \in \mathcal{L}(D)\}.$$

If $\mathcal{Q}$ is $\mathcal{P}$-stable then L is D-functional.

Learning Algorithm

$tRPNI^{\mathcal{P}}_{D}(S)$

```
// parameters: D, a schema; P, a D–pruning strategy
// input:      S, a sample of D, P–pruned trees
// output:     A, a D–NSTT
A ← init(S) // maximal deterministic automaton recognizing S
while (p1, p2) ← pick_two_states(A) do // states properly ordered
  A' ← deterministically_merge(A, p1, p2)
  if A' is a D–NSTT then // check D–functional, non–null
    A ← A'
return A
```

Main Result

Theorem

Let D be a deterministic schema over Σ and $\mathcal{P}$ a D-pruning strategy. The class of $\mathcal{P}$-stable queries represented by deterministic D-NSTTs over Σ is learnable from D-pruned examples by $t\mathcal{RPNI}_D^{\mathcal{P}}$ in the following sense:

1. *given a sample S of $D, \mathcal{P}$-pruned trees, $t\mathcal{RPNI}_D^{\mathcal{P}}(S)$ returns a D-NSTTs in time polynomial in $|S|$;*
2. *for all D-NSTTs A representing a $\mathcal{P}$-stable query, there exists a characteristic set CS of $D, \mathcal{P}$-pruned trees of cardinality polynomial in $|A|$ for which, given $S \supseteq CS$ consistent with A, $t\mathcal{RPNI}_D^{\mathcal{P}}(S)$ returns a D-NSTTs equivalent to A.*

Proof Sketch

- Reduction onto learnability of regular monadic queries ($t\mathcal{RPNI}$)
- Based on the relation between $\mathcal{P}$-stable queries and D-functionality
- Characteristic set obtained by D-completion
- Bisimulation of $t\mathcal{RPNI}$ and $t\mathcal{RPNI}^{\mathcal{P}}_{D}$

Learnability of Schema-Consistent Queries

Corollary

Let D be a deterministic schema. The class of D-consistent queries represented by NSTTs can be learned from completely annotated examples.

Proof: $t\mathcal{RPNI}_D^{\mathcal{P}}$ with $\mathcal{P} = \mathcal{P}_{id}$ and D-inclusion checking.

Outline

1 XML, Schemas and Queries

2 Tree Automata

3 Schema-Guided Pruning

4 Stable Queries

5 Learnability Results

6 Experiments

Some Practical Questions

- XML data rather than HTML data
- Does schema's semantics help?
- Are stable queries relevant?
- Given a query that is stable by several pruning strategies, which one is the most approriate?

Settings

- New learning system in OCaml
- Learning algorithm implemented parameters:
  - Prunings: $\mathcal{P}^{\mathcal{U}}_{\text{only}}$, $\mathcal{P}^{\mathcal{U}}_{\text{ext.}}$, $\mathcal{P}^{D}_{\text{only}}$, $\mathcal{P}^{D}_{\text{ext.}}$
  - Dynamic/static inclusion checking, “horizontal” typing heuristic
- Several combinations are not theoretically complete

Protocol

- Simulation of the user behaviour when defining a new query
- Interactions between the user and the system:
  - The user provides annotated examples to the system
  - The system infers a new query, which in turn the user evaluates
  - More examples while not satisfied
- Weak interactions:
  - Annotations “per document” instead of “per node”
  - Sufficient for evaluating the system
- Cross-validation by repeated random sub-sampling
- Compute F-measure w.r.t. number of examples

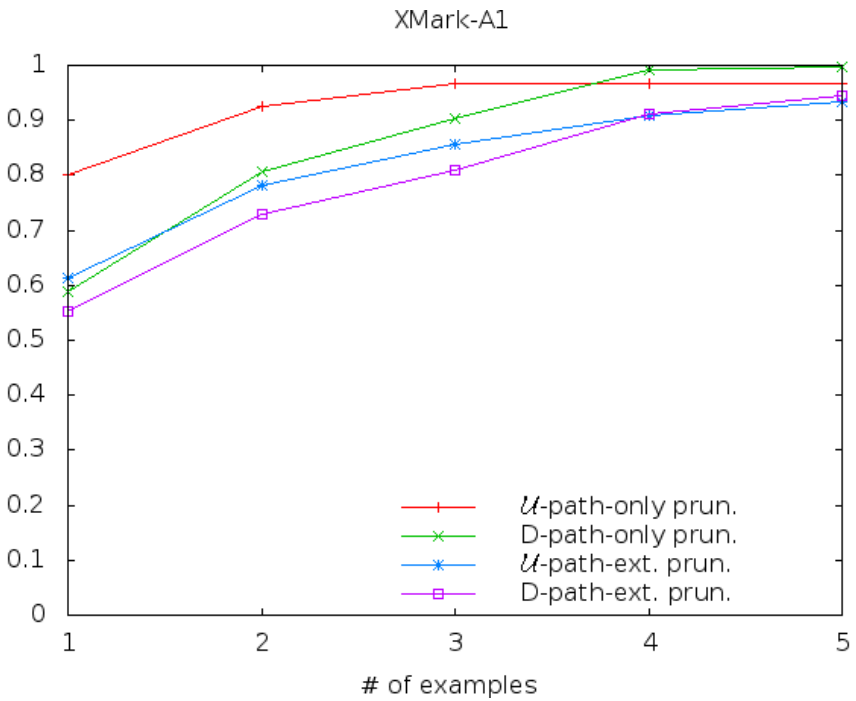
XPath Queries

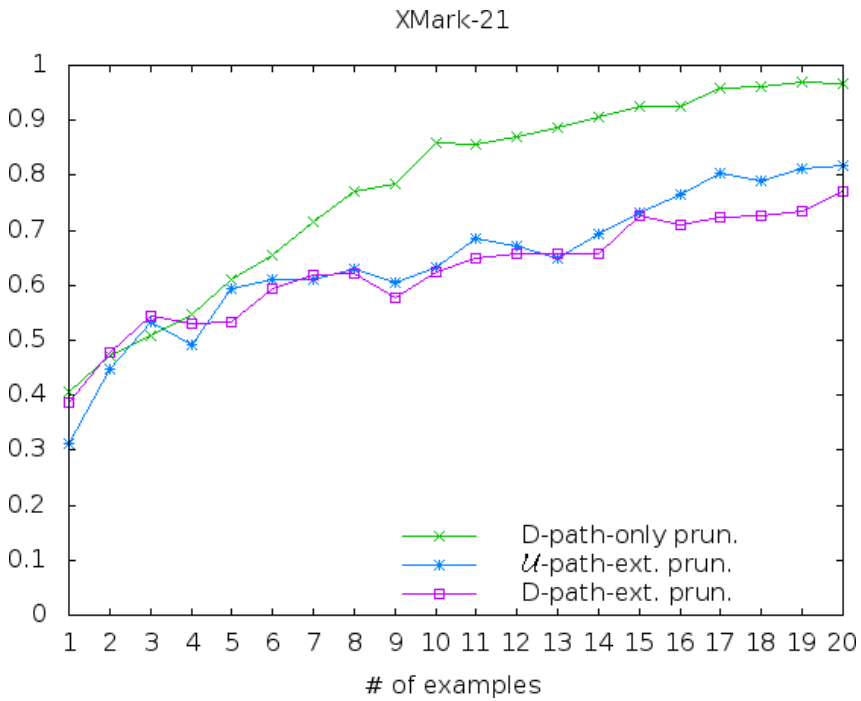
- XMark-A1:
  $/site/closed\_auctions/closed\_auction$
  $/annotation/description/text/keyword$
- XMark-02:
  $/site/open\_auctions/open\_auction/bidder[1]/increase$
- XMark-21:
  $/site/open\_auctions/open\_auction[count(bidder) \geq 3]/itemref$
- XMark-A6:
  $/site/people/person[profile/gender \texttt{ and } profile/age]/name$

Datasets

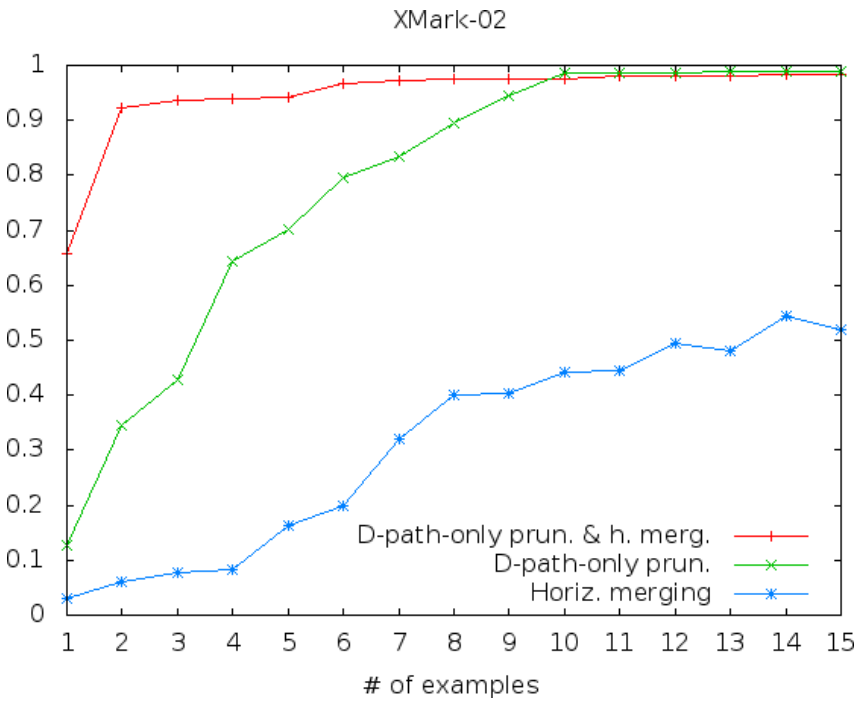
Query id.	# doc.	Stable w.r.t.			
		$\mathcal{P}^{\mathcal{U}}_{\text{only}}$	$\mathcal{P}^{\mathcal{U}}_{\text{ext.}}$	$\mathcal{P}^{D}_{\text{only}}$	$\mathcal{P}^{D}_{\text{ext.}}$
XMark-A1	50	yes	yes	yes	yes
XMark-02	100	no	yes	yes	yes
XMark-21	50	no	yes	yes	yes
XMark-A6	250	no	no	no	yes

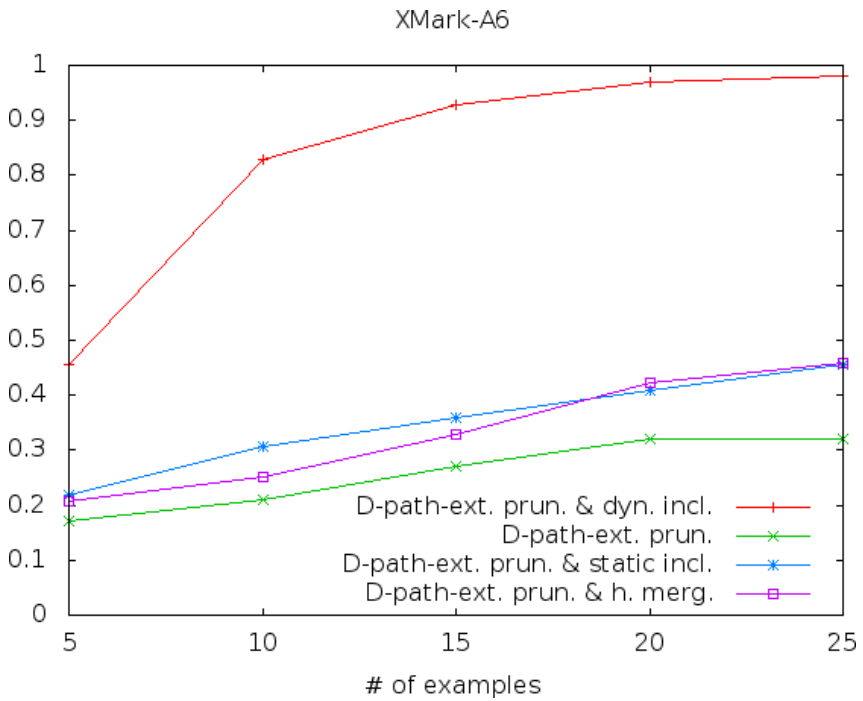
Pruning Alone





Combination of Heuristics





Facts

- Confirmation: pruning is essential for XML query induction
- Contextual information brought by the schema are very useful
- The most appropriate pruning is one for which the query is stable, and which makes use of the schema
- Combining dynamic inclusion checking with schema-guided pruning enables to learn “difficult” queries

Summary

Main contributions

- Complete learning algorithms
  - Stable queries
  - Schema-consistent queries
- Classification of pruning strategies w.r.t. stability
- Experiments on XML datasets
- Efficient inclusion checking (Information and Computation, 2009)
  - Both practical and theoretical interest
  - Factorized tree automata for DTDs

Summary

Main contributions

- Complete learning algorithms
  - Stable queries
  - Schema-consistent queries
- Classification of pruning strategies w.r.t. stability
- Experiments on XML datasets
- Efficient inclusion checking (Information and Computation, 2009)
  - Both practical and theoretical interest
  - Factorized tree automata for DTDs

Some lessons we can drawn. . .

- Using schema information improves XML query induction
  - Theoretical fundations
  - Practical justification
- Difficulty scale of target queries can be addressed through stability

Perspectives

- How perform other learning approches on stable queries?
- Interactive learning:
  - Start with the less preserving pruning (i.e., $\mathcal{P}^{D}_{\text{only}}$)
  - Evolve towards more preserving prunings
- n-ary queries:
  - How should we prune?
  - Same definition of stable queries?
- Text values: schemas may contain type information
- Extension to XML transformations: notion of stability?