When Can We Answer Queries Using Result-Bounded Data Interfaces?

Antoine Amarilli[1], Michael Benedikt[2]

June 12th, 2018

[1]Télécom ParisTech

[2]Oxford University

Problem: Answering Queries Using Web Services

Directory service

Find researchers from a department

DBLP service

Find papers from a researcher

- We have several Web services that expose data

Problem: Answering Queries Using Web Services

Directory service

*Find **researchers** from a **department***

DBLP service

*Find **papers** from a **researcher***

Query

*Find all **papers** written by **researchers** from my **department**?*

- We have several **Web services** that expose data
- We want to answer a **query** using the Web services

Problem: Answering Queries Using Web Services

Directory service	DBLP service	Query
		
*Find **researchers** from a **department***	*Find **papers** from a **researcher***	*Find all **papers** written by **researchers** from my **department**?*

- We have several **Web services** that expose data
- We want to answer a **query** using the Web services

→ How can we **rephrase** the query against the Web services?

Formalizing the Problem

Service schema:

DBLP(author, title, year)

- Model each service as a relation

Formalizing the Problem

Service schema:

DBLP(author, title, year)

- Model each service as a relation
- Some attributes are inputs

Formalizing the Problem

Service schema:

DBLP(author, title, year)

Input author? [ ] OK

- Model each service as a **relation**
- Some attributes are **inputs**
- Access the service by giving a **binding** for the input attributes

Formalizing the Problem

Service schema:

DBLP(<u>author</u>, title, year)

Input author? | Michael Benedikt | OK

- Model each service as a **relation**
- Some attributes are <u>inputs</u>
- Access the service by giving a **binding** for the input attributes

Formalizing the Problem

Service schema:

DBLP(<u>author</u>, title, year)

Input author? Michael Benedikt | OK

author	title	year
Michael Benedikt	Goal-Driven Query Answering ...	2018
Michael Benedikt	Form Filling Based on ...	2018
Michael Benedikt	How Can Reasoners Simplify ...	2018
Michael Benedikt	When Can We Answer Queries ...	2018
...	...	...

- Model each service as a **relation**
- Some attributes are <u>inputs</u>
- Access the service by giving a **binding** for the input attributes
- The service returns **all tuples** that match the **binding**

Formalizing the Problem

Service schema:

DBLP(author, title, year)

Input author? Michael Benedikt OK

author	title	year
Michael Benedikt	Goal-Driven Query Answering ...	2018
Michael Benedikt	Form Filling Based on ...	2018
Michael Benedikt	How Can Reasoners Simplify ...	2018
Michael Benedikt	When Can We Answer Queries ...	2018
...	...	...

- Model each service as a **relation**
- Some attributes are **inputs**
- Access the service by giving a **binding** for the input attributes
- The service returns **all tuples** that match the **binding**

Query: conjunctive query over the relations

Find all papers written by researchers from my department?

$Q(t) : \exists a\, y\ \text{Directory}(\text{MyDept}, a) \wedge \text{DBLP}(a, t, y)$

Formalizing the Problem

Service schema:

DBLP(author, title, year)

Input author? Michael Benedikt | OK

author	title	year
Michael Benedikt	Goal-Driven Query Answering ...	2018
Michael Benedikt	Form Filling Based on ...	2018
Michael Benedikt	How Can Reasoners Simplify ...	2018
Michael Benedikt	When Can We Answer Queries ...	2018
...	...	...

- Model each service as a **relation**
- Some attributes are inputs
- Access the service by giving a **binding** for the input attributes
- The service returns **all tuples** that match the **binding**

Query: conjunctive query over the relations

Find all papers written by researchers from my department?

$Q(t) : \exists a\, y\ \mathrm{Directory}(\mathrm{MyDept}, a) \land \mathrm{DBLP}(a, t, y)$

Constraints: express logical relationships between the services

Every researcher from the directory is in DBLP

$\Sigma : \forall d\, a\ \mathrm{Directory}(d, a) \rightarrow \exists t\, y\ \mathrm{DBLP}(a, t, y)$

Existing Solutions

Given the service schema S, the query Q and the constraints Σ we want to find a monotone plan for Q using the services of S

Existing Solutions

Given the service schema S, the query Q and the constraints Σ we want to find a monotone plan for Q using the services of S

Example: *Find all papers written by researchers from my department?* with Directory($\underline{\text{department}}$, person) and DBLP($\underline{\text{author}}$, title, year)

Existing Solutions

Given the service schema S, the query Q and the constraints Σ we want to find a monotone plan for Q using the services of S

Example: *Find all papers written by researchers from my department?* with Directory(<u>department</u>, person) and DBLP(<u>author</u>, title, year)

What is a monotone plan?

- Access the services by giving bindings
- Evaluate monotone relational algebra
- The plan is correct if it returns $Q(D)$ on any database D that satisfies Σ

Example:

$T_1 \Leftarrow$ Directory $\Leftarrow$ MyDept;

$T_2 \Leftarrow$ DBLP $\Leftarrow \pi_{\text{person}}(T_1)$;

$T_3 \Leftarrow \pi_{\text{title}}(T_2)$;

Return T_3

Existing Solutions

Given the service schema S, the query Q and the constraints Σ we want to find a monotone plan for Q using the services of S

Example: *Find all papers written by researchers from my department?* with Directory(department, person) and DBLP(author, title, year)

What is a monotone plan?

- Access the services by giving bindings
- Evaluate monotone relational algebra
- The plan is correct if it returns $Q(D)$ on any database D that satisfies Σ

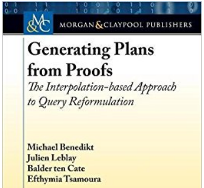
Example:

$T_1 \Leftarrow$ Directory $\Leftarrow$ MyDept;

$T_2 \Leftarrow$ DBLP $\Leftarrow \pi_{\text{person}}(T_1)$;

$T_3 \Leftarrow \pi_{\text{title}}(T_2)$;

Return T_3



Extensive literature about how to reformulate queries to monotone plans and about the complexity depending on the constraint language

New Challenge: Result Bounds

- Real Web services do not return all matching tuples for accesses!

New Challenge: Result Bounds

- Real Web services do not return **all matching tuples** for accesses!

Currently the following URL query parameters are recognized:

Parameter	Description
q	The query string to search for.
h	Maximum number of search results (hits) to return. For bandwidth reasons, this number is capped at 1000.

New Challenge: Result Bounds

- Real Web services do not return **all matching tuples** for accesses!

Currently the following URL query parameters are recognized:

Parameter	Description
q	The query string to search for.
h	Maximum number of search results (hits) to return. For bandwidth reasons, this number is capped at 1000.

→ Plan results are **nondeterministic** and may not be **correct**!

New Challenge: Result Bounds

- Real Web services do not return **all matching tuples** for accesses!

Currently the following URL query parameters are recognized:

Parameter	Description
q	The query string to search for.
h	Maximum number of search results (hits) to return. For bandwidth reasons, this number is capped at 1000.

→ Plan results are **nondeterministic** and may not be **correct**!

Formalization: DBLP(author, title, year) has a **result bound** of 1000

- If an access matches ≤ 1000 tuples then they are **all** returned
- If it matches > 1000 tuples then we get 1000 of them (random)

→ **How to reformulate queries using result-bounded services?**

Formal Problem Statement

Input:

- Service schema S of relation names and attributes with <u>input attributes</u> and optionally a result bound
  - $\rightarrow$ Directory(<u>department</u>, person)
  - $\rightarrow$ DBLP(<u>author</u>, title, year) with bound 1000
- Conjunctive query Q
  - $\rightarrow$ $Q(t) : \exists a\, y\ \text{Directory}(\text{MyDept}, a) \wedge \text{DBLP}(a, t, y)$
- Constraints Σ
  - $\rightarrow$ $\forall d\, a\ \text{Directory}(d, a) \rightarrow \exists t\, y\ \text{DBLP}(a, t, y)$

Formal Problem Statement

Input:

- Service schema S of relation names and attributes with <u>input attributes</u> and optionally a result bound
  - → Directory(<u>department</u>, person)
  - → DBLP(<u>author</u>, title, year) with bound 1000
- Conjunctive query Q
  - → $Q(t) : \exists a\, y\ \text{Directory}(\text{MyDept}, a) \wedge \text{DBLP}(a, t, y)$
- Constraints Σ
  - → $\forall d\, a\ \text{Directory}(d, a) \rightarrow \exists t\, y\ \text{DBLP}(a, t, y)$

Output:

- Is there a monotone plan for Q on S under Σ?

Formal Problem Statement

Input:

- Service schema S of relation names and attributes with input attributes and optionally a result bound
  - → Directory(department, person)
  - → DBLP(author, title, year) with bound 1000
- Conjunctive query Q
  - → $Q(t) : \exists a\, y\ \text{Directory}(\text{MyDept}, a) \wedge \text{DBLP}(a, t, y)$
- Constraints Σ
  - → $\forall d\, a\ \text{Directory}(d, a) \rightarrow \exists t\, y\ \text{DBLP}(a, t, y)$

Output:

- Is there a monotone plan for Q on S under Σ?

We study:

→ What is the complexity of deciding plan existence, depending on the constraint language?

→ In which ways are result-bounded services useful?

Summary of Results

→ We show **schema simplification** results that describe when result bounds on services can be **removed**

Summary of Results

- → We show schema simplification results that describe when result bounds on services can be removed
- → We show complexity results for many constraint languages on deciding the existence of monotone plans

Summary of Results

→ We show **schema simplification** results that describe when result bounds on services can be **removed**

→ We show **complexity results** for many constraint languages on deciding the existence of monotone plans

Fragment	Simplification	Complexity
Inclusion dependencies (IDs)	Existence-check	EXPTIME-complete
Bounded-width IDs	Existence-check	NP-complete
Functional dependencies (FDs)	FD	NP-complete
FDs and UIDs	Choice	NP-hard, in EXPTIME
Equality-free FO	Choice	Undecidable
Frontier-guarded TGDs	Choice	2EXPTIME-complete

Summary of Results

- → We show **schema simplification** results that describe when result bounds on services can be **removed**
- → We show **complexity results** for many constraint languages on deciding the existence of monotone plans

Fragment	**Simplification**	**Complexity**
Inclusion dependencies (IDs)	Existence-check	**EXPTIME**-complete
Bounded-width IDs	Existence-check	**NP**-complete
Functional dependencies (FDs)	FD	**NP**-complete
FDs and UIDs	Choice	**NP**-hard, in **EXPTIME**
Equality-free FO	Choice	Undecidable
Frontier-guarded TGDs	Choice	**2EXPTIME**-complete

→ Let's see the **schema simplification results** and proof techniques

Existence-Check Simplification

Idea: use result-bounded services to check the existence of tuples

- Schema: DBLP(author, title, year) with bound 1000
- Query Q: *Has Michael Benedikt published something?*
- Plan: access DBLP with "Michael Benedikt" and check if empty

Existence-Check Simplification

Idea: use result-bounded services to **check the existence** of tuples

- **Schema:** **DBLP**(<u>author</u>, **title**, **year**) with bound 1000
- **Query** Q: *Has Michael Benedikt published something?*
- **Plan:** access **DBLP** with "Michael Benedikt" and check if empty

A schema S with constraints Σ is **existence-check simplifiable** if any query that has a plan on S under Σ still has a plan on the **existence-check approximation**:

Existence-Check Simplification

Idea: use result-bounded services to **check the existence** of tuples

- **Schema:** DBLP(<u>author</u>, title, year) with bound 1000
- **Query** Q: *Has Michael Benedikt published something?*
- **Plan:** access DBLP with "Michael Benedikt" and check if empty

A schema S with constraints Σ is **existence-check simplifiable** if any query that has a plan on S under Σ still has a plan on the **existence-check approximation**:

- For each relation DBLP(<u>author</u>, title, year) with a result bound, create a new relation $\text{DBLP}_{\text{check}}$(<u>author</u>)

Existence-Check Simplification

Idea: use result-bounded services to **check the existence** of tuples

- **Schema:** DBLP(<u>author</u>, title, year) with bound 1000
- **Query** Q: *Has Michael Benedikt published something?*
- **Plan:** access DBLP with "Michael Benedikt" and check if empty

A schema S with constraints Σ is **existence-check simplifiable** if any query that has a plan on S under Σ still has a plan on the **existence-check approximation**:

- For each relation DBLP(<u>author</u>, title, year) with a result bound, create a new relation $\text{DBLP}_{\text{check}}$(<u>author</u>)
- Add two IDs in Σ to relate $\text{DBLP}_{\text{check}}$ and DBLP:

$$\forall a\ \text{DBLP}_{\text{check}}(a) \leftrightarrow \exists t\, y\ \text{DBLP}(a, t, y)$$

Existence-Check Simplification

Idea: use result-bounded services to **check the existence** of tuples

- **Schema:** DBLP(author, title, year) with bound 1000
- **Query** Q: *Has Michael Benedikt published something?*
- **Plan:** access DBLP with "Michael Benedikt" and check if empty

A schema S with constraints Σ is **existence-check simplifiable** if any query that has a plan on S under Σ still has a plan on the **existence-check approximation**:

- For each relation DBLP(author, title, year) with a result bound, create a new relation $\text{DBLP}_{\text{check}}$(author)
- Add two IDs in Σ to relate $\text{DBLP}_{\text{check}}$ and DBLP:
  $$\forall a\ \text{DBLP}_{\text{check}}(a) \leftrightarrow \exists t\, y\ \text{DBLP}(a, t, y)$$
- Forbid direct accesses to DBLP (so the result bound is irrelevant)

Existence-Check Simplification Results

Theorem

Any schema $\mathbf{S}$ *with constraints* Σ *in* ***Inclusion dependencies*** *(IDs) is* ***existence-check simplifiable***

$\rightarrow$ Under IDs, result-bounded services only serve as existence checks

Existence-Check Simplification Results

Theorem

Any schema S *with constraints* Σ *in* ***Inclusion dependencies*** *(IDs) is* ***existence-check simplifiable***

→ Under IDs, result-bounded services only serve as existence checks

As the existence-check approximation has no result bounds, we can reduce to the classical setting and deduce:

Corollary

For any schema S *with result bounds, ID constraints* Σ*, and query* Q*, deciding the existence of a monotone plan is* EXPTIME-*complete*

FD Simplification

Result-bounded services can do more than existence checks:

- Schema S: directory that returns addresses and phone numbers
  → Dir2(<u>name</u>, address, phone) with bound 1000

FD Simplification

Result-bounded services can do more than existence checks:

- Schema *S*: directory that returns addresses and phone numbers
  $\rightarrow$ Dir2(name, address, phone) with bound 1000
- Constraints: a Functional dependency (FD) ϕ : name $\rightarrow$ address
  $\rightarrow$ *Each person has at most one address*

FD Simplification

Result-bounded services can do more than existence checks:

- Schema *S*: directory that returns addresses and phone numbers
  $\rightarrow$ Dir2(name, address, phone) with bound 1000
- Constraints: a Functional dependency (FD) ϕ : name $\rightarrow$ address
  $\rightarrow$ *Each person has at most one address*
- Query: *Find the address of "Michael Benedikt"*

FD Simplification

Result-bounded services can do more than existence checks:

- Schema S: directory that returns addresses and phone numbers
  $\rightarrow$ Dir2(name, address, phone) with bound 1000
- Constraints: a Functional dependency (FD) ϕ : name $\rightarrow$ address
  $\rightarrow$ *Each person has at most one address*
- Query: *Find the address of "Michael Benedikt"*
- Plan: access Dir2, return the address of any obtained tuple

FD Simplification

Result-bounded services can do more than existence checks:

- Schema *S*: directory that returns addresses and phone numbers
  $\rightarrow$ Dir2(name, address, phone) with bound 1000
- Constraints: a Functional dependency (FD) ϕ : name $\rightarrow$ address
  $\rightarrow$ *Each person has at most one address*
- Query: *Find the address of "Michael Benedikt"*
- Plan: access Dir2, return the address of any obtained tuple

We call *S* and Σ FD-simplifiable if any query that has a plan on *S* and Σ still has one on the FD approximation:

FD Simplification

Result-bounded services can **do more** than existence checks:

- Schema S: directory that returns **addresses** and **phone numbers**
  $\rightarrow$ **Dir2**(<u>**name**</u>, **address**, **phone**) with bound 1000
- **Constraints:** a **Functional dependency** (FD) ϕ : **name** $\rightarrow$ **address**
  $\rightarrow$ *Each person has at most one address*
- **Query:** *Find the address of "Michael Benedikt"*
- **Plan:** access **Dir2**, return the **address** of any obtained tuple

We call S and Σ **FD-simplifiable** if any query that has a plan on S and Σ still has one on the **FD approximation**:

- For each relation **Dir2**(<u>**name**</u>, **address**, **phone**) with result bound, create a new relation Dir2_{FD}(<u>**name**</u>, **address**) that outputs the attributes **determined** in Σ by the input attributes

FD Simplification

Result-bounded services can do more than existence checks:

- Schema S: directory that returns addresses and phone numbers
  $\rightarrow$ Dir2(name, address, phone) with bound 1000
- Constraints: a Functional dependency (FD) ϕ : name $\rightarrow$ address
  $\rightarrow$ *Each person has at most one address*
- Query: *Find the address of "Michael Benedikt"*
- Plan: access Dir2, return the address of any obtained tuple

We call S and Σ FD-simplifiable if any query that has a plan on S and Σ still has one on the FD approximation:

- For each relation Dir2(name, address, phone) with result bound, create a new relation Dir2_{FD}(name, address) that outputs the attributes determined in Σ by the input attributes
- Forbid accesses on Dir2 and add IDs with Dir2_{FD} like before

$$\forall n\, a\ \text{Dir2}_{\text{FD}}(n, a) \leftrightarrow \exists p\ \text{Dir2}(n, a, p)$$

FD Simplification Results

Theorem

Any schema S with constraints Σ in Functional dependencies (FDs) is FD simplifiable

→ Under FDs, result-bounded services are only useful to access outputs that are functionally determined (i.e., we are guaranteed to have only one result)

FD Simplification Results

Theorem

Any schema S with constraints Σ in Functional dependencies (FDs) is FD simplifiable

→ Under FDs, result-bounded services are only useful to access outputs that are functionally determined (i.e., we are guaranteed to have only one result)

Again, there are no result bounds left in the FD approximation, so we can use this result to show:

Corollary

For any schema S with result bounds, FD constraints Σ, and query Q, deciding the existence of a monotone plan is NP-complete

Choice Simplification

With expressive constraints, the FD approximation is not enough:

Lemma

There is a service schema S, query Q, and TGDs Σ such that Q is not FD-simplifiable (hence, not existence-check-simplifiable)

Choice Simplification

With expressive constraints, the FD approximation is not enough:

Lemma

There is a service schema S, query Q, and TGDs Σ such that Q is not FD-simplifiable (hence, not existence-check-simplifiable)

A less drastic simplification is the choice simplification:

- For every service with a result bound, change the bound to be 1

$\rightarrow$ Intuition: It's important to get some tuple if one exists

Choice Simplification Results

Theorem

Any schema $\mathbf{S}$ *with constraints* Σ *in* ***equality-free first-order logic*** *(e.g., TGDs) is* ***choice simplifiable***

Choice Simplification Results

Theorem

Any schema S with constraints Σ in equality-free first-order logic (e.g., TGDs) is choice simplifiable

Thus, plan existence is decidable for decidable FO fragments, e.g., frontier-guarded TGDs (FGTGDs):

Corollary

For any schema S with result bounds, query Q, and FGTGDs Σ deciding the existence of a monotone plan is 2EXPTIME-complete

Choice Simplification Results

Theorem

Any schema **S** *with constraints* Σ *in* ***equality-free first-order logic*** *(e.g., TGDs) is* ***choice simplifiable***

Thus, plan existence is decidable for **decidable FO fragments**, e.g., **frontier-guarded TGDs** (FGTGDs):

Corollary

For any schema **S** *with result bounds, query* **Q***, and* ***FGTGDs*** Σ *deciding the existence of a monotone plan is* **2EXPTIME-*complete***

We can also show choice approximability for another fragment:

Theorem

Any schema **S** *with constraints* Σ *that are* ***FDs*** *and* ***unary IDs*** (UIDs) *is* ***choice simplifiable***

→ This implies that plan existence is **decidable** for FDs and UIDs

Overview of Proof Techniques

- Show that result-bounds can be axiomatized in simpler ways:
  - Ensure that doing the same access twice returns the same result
  - Only write the lower bound: *"if i results exist then i are returned"*

Overview of Proof Techniques

- Show that result-bounds can be axiomatized in simpler ways:
  - Ensure that doing the same access twice returns the same result
  - Only write the lower bound: *"if i results exist then i are returned"*
- Show that plan existence can be rephrased as answerability:

  → If a database I satisfies Q and I' has more accessible data than I then I' should satisfy Q as well

Overview of Proof Techniques

- Show that result-bounds can be axiomatized in simpler ways:
  - Ensure that doing the same access twice returns the same result
  - Only write the lower bound: *"if i results exist then i are returned"*
- Show that plan existence can be rephrased as answerability:

  $\rightarrow$ If a database I satisfies Q and I' has more accessible data than I then I' should satisfy Q as well
- Show simplification results using a blowup technique:
  - Start with a counterexample to answerability on the simplification
  - Blow it up to a counterexample on the original schema

Overview of Proof Techniques

- Show that result-bounds can be axiomatized in simpler ways:
  - Ensure that doing the same access twice returns the same result
  - Only write the lower bound: *"if i results exist then i are returned"*
- Show that plan existence can be rephrased as answerability:

  → If a database I satisfies Q and I' has more accessible data than I then I' should satisfy Q as well
- Show simplification results using a blowup technique:
  - Start with a counterexample to answerability on the simplification
  - Blow it up to a counterexample on the original schema
- Reduce to query containment under constraints

  → Study the result of the translation to show complexity bounds

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Theorem

Plan existence under bounded-width IDs is NP-complete

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Theorem

Plan existence under bounded-width IDs is NP-*complete*

Theorem

Plan existence under UIDs and FDs is NP-*hard and in* EXPTIME

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Theorem

*Plan existence under **bounded-width IDs** is NP-**complete***

Theorem

*Plan existence under **UIDs and FDs** is NP-**hard** and **in** EXPTIME*

- Result for arity-2 constraints

Theorem

*Plan existence is **decidable** for **guarded two-variable FO + counting***

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Theorem

Plan existence under bounded-width IDs is NP-complete

Theorem

Plan existence under UIDs and FDs is NP-hard and in EXPTIME

- Result for arity-2 constraints

Theorem

Plan existence is decidable for guarded two-variable FO + counting

- Results when the database is assumed to be finite

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Theorem

Plan existence under bounded-width IDs is NP-complete

Theorem

Plan existence under UIDs and FDs is NP-hard and in EXPTIME

- Result for arity-2 constraints

Theorem

Plan existence is decidable for guarded two-variable FO + counting

- Results when the database is assumed to be finite
- Results for non-monotone plans (= with relational difference)

Other Results in the Paper

- Better complexity bounds using a linearization technique for query containment under IDs + side information

Theorem

Plan existence under bounded-width IDs is NP-complete

Theorem

Plan existence under UIDs and FDs is NP-hard and in EXPTIME

- Result for arity-2 constraints

Theorem

Plan existence is decidable for guarded two-variable FO + counting

- Results when the database is assumed to be finite
- Results for non-monotone plans (= with relational difference)
- Example of FO constraints that are not choice simplifiable

Summary and future work

- Problem: Given a schema of services with result bounds, logical constraints, and a query, is there a plan to answer it?
- Simplification results to remove bounds, and complexity results

Summary and future work

- **Problem:** Given a schema of services with **result bounds**, logical constraints, and a query, is there a plan to answer it?
- **Simplification results** to remove bounds, and **complexity results**

Fragment	Simplification	Complexity
Inclusion dependencies (IDs)	Existence-check	EXPTIME-complete
Bounded-width IDs	Existence-check	NP-complete
Functional dependencies (FDs)	FD	NP-complete
FDs and UIDs	Choice	NP-hard, in EXPTIME
Equality-free FO	Choice	Undecidable
Frontier-guarded TGDs	Choice	2EXPTIME-complete

Summary and future work

- Problem: Given a schema of services with result bounds, logical constraints, and a query, is there a plan to answer it?
- Simplification results to remove bounds, and complexity results

Fragment	Simplification	Complexity
Inclusion dependencies (IDs)	Existence-check	EXPTIME-complete
Bounded-width IDs	Existence-check	NP-complete
Functional dependencies (FDs)	FD	NP-complete
FDs and UIDs	Choice	NP-hard, in EXPTIME
Equality-free FO	Choice	Undecidable
Frontier-guarded TGDs	Choice	2EXPTIME-complete

→ What are other possible uses of the linearization technique?

→ Do the bounds matter in practice? (approximate plans?)

Summary and future work

- **Problem:** Given a schema of services with **result bounds**, logical constraints, and a query, is there a plan to answer it?
- **Simplification results** to remove bounds, and **complexity results**

Fragment	Simplification	Complexity
Inclusion dependencies (IDs)	Existence-check	EXPTIME-complete
Bounded-width IDs	Existence-check	NP-complete
Functional dependencies (FDs)	FD	NP-complete
FDs and UIDs	Choice	NP-hard, in EXPTIME
Equality-free FO	Choice	Undecidable
Frontier-guarded TGDs	Choice	2EXPTIME-complete

→ What are other possible uses of the **linearization technique**?

→ Do the bounds matter in **practice**? (approximate plans?)

Thanks for your attention!

References

Benedikt, M., Leblay, J., Cate, B. t., and Tsamoura, E. (2016).
Generating plans from proofs: the interpolation-based approach to query reformulation.
Morgan & Claypool.