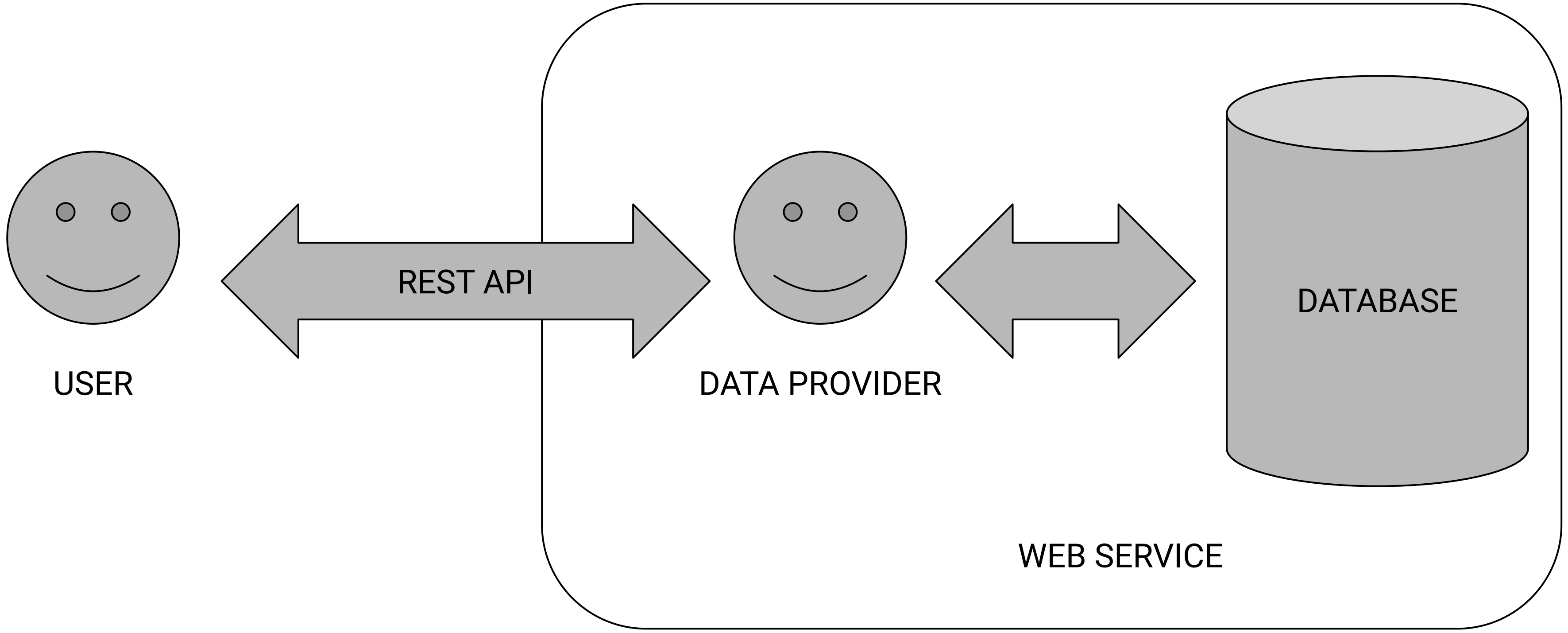
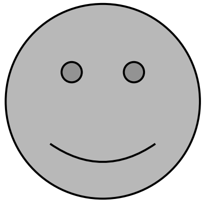
# Equivalent Rewritings on Path Views with Binding Patterns

Julien Romero, Nicoleta Preda, Antoine Amarilli, Fabian Suchanek

## Motivation

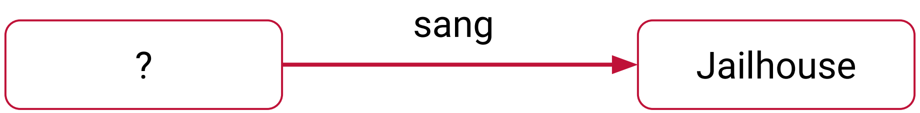


## Motivation



USER

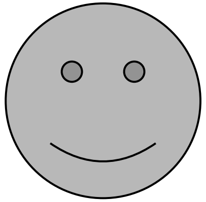
- Wants to **answer** a question (query)
  - When was Elvis Presley born?
  - What is the largest city in Europe?
  - Who is the President of France?



The Answer:

Elvis Presley

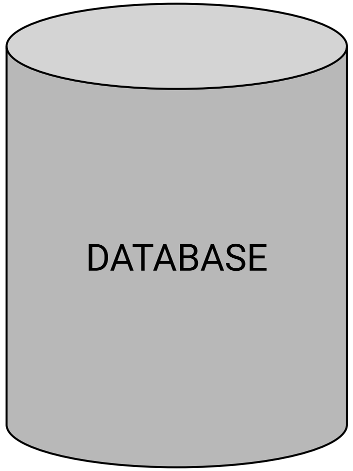
## Motivation



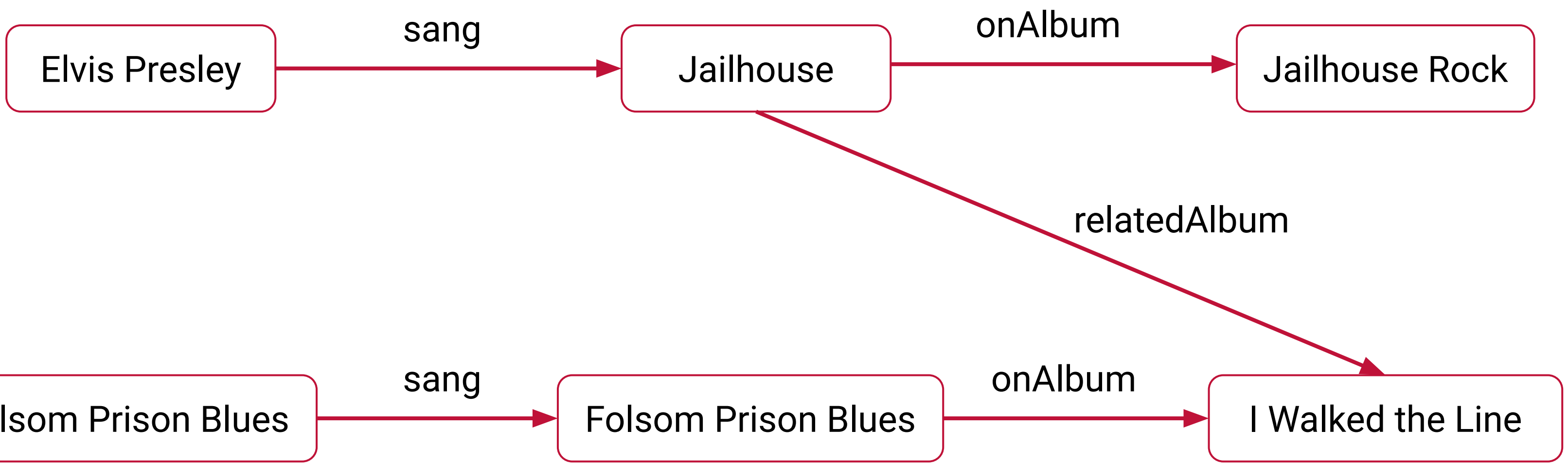
USER

- Wants to **answer** a question (query)
  - When was Elvis Presley born?
  - What is the largest city in Europe?
  - Who is the President of France
- Wants **automatic tools** to answer the questions
- Wants to be sure that all results provided by the tool are **correct**
- Wants to be sure that all results provided by the tool are **complete**
  - e.g., “What are the computer science conferences?” should not return only ESWC

## Example - Raw Database



## Example - Raw Database

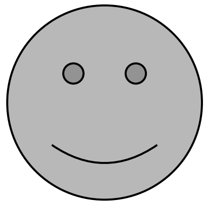


## How do we access the database?

Though a set of access functions

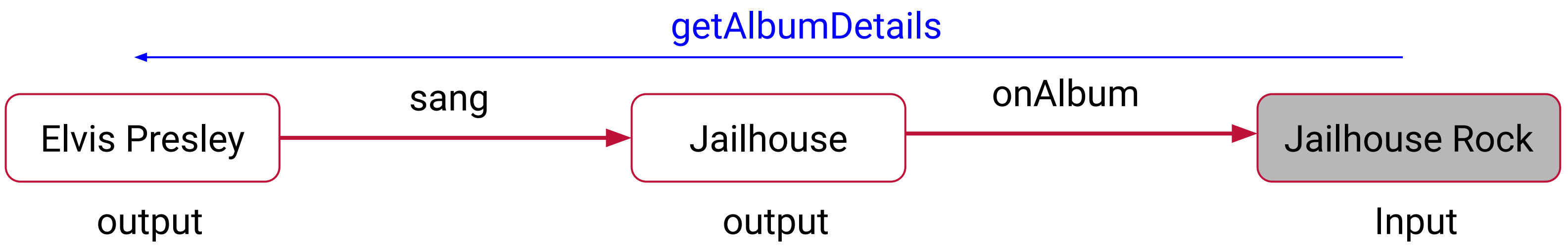
The data provider can limit the number of queries on the database, on charge each call x euros.

The access functions provide a *view* on the database, i.e. they extract the results from the database.



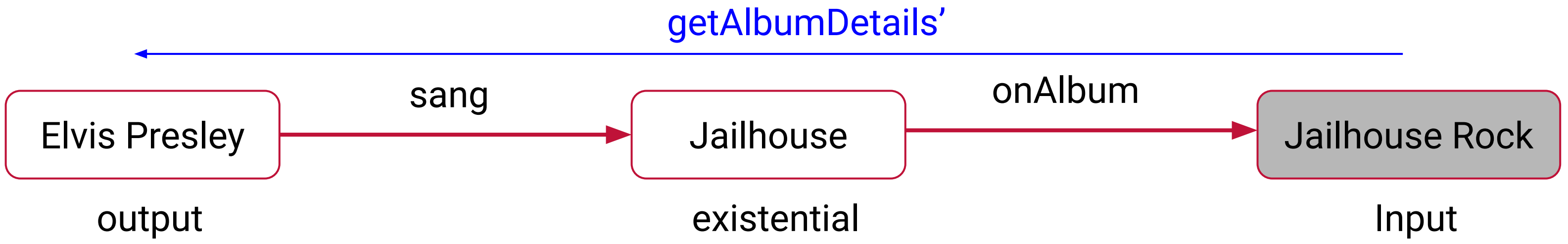
DATA PROVIDER

## Example - A view example



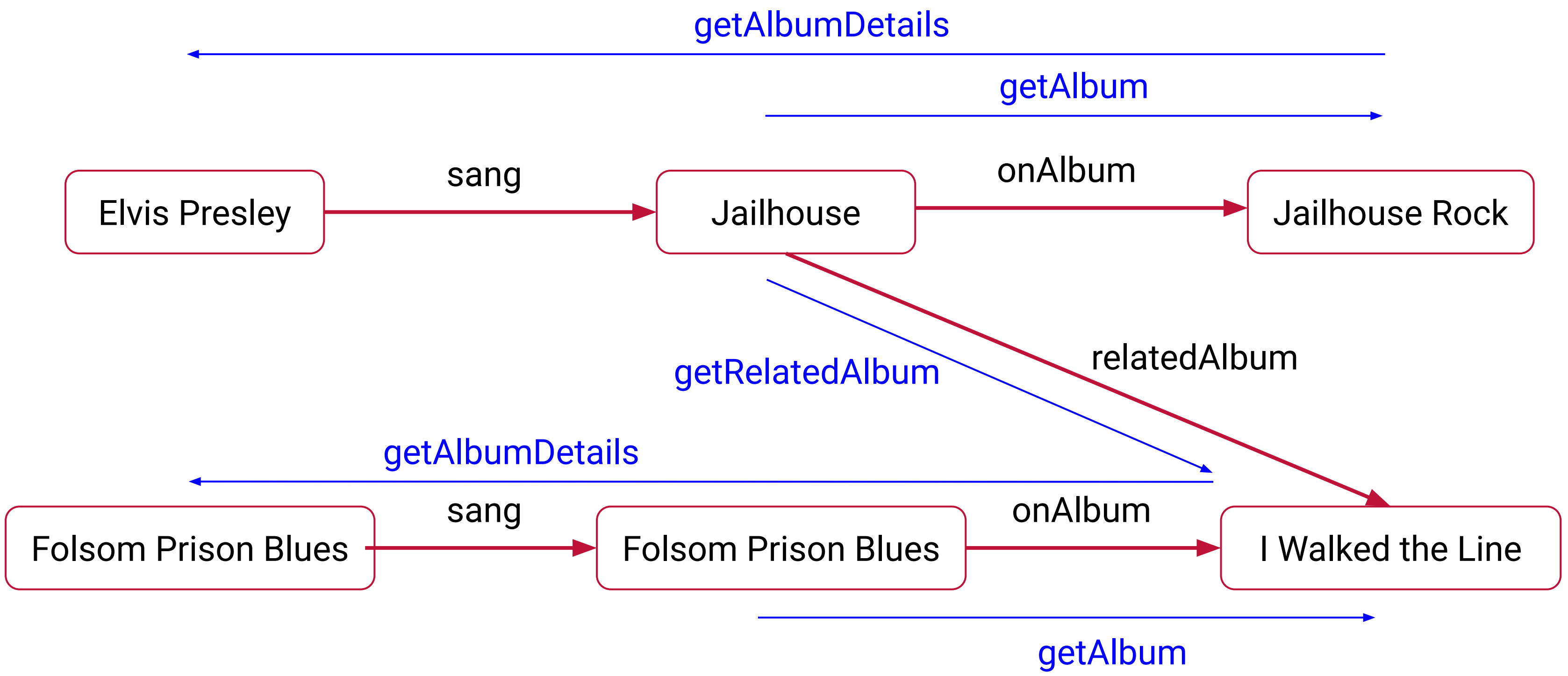
## Example - A view example

The views can also “hide” information behind existential variables, for example:



does not return *Jailhouse*.

# Example - The Result of All the Views



## Motivation

- Provides an easy interface
- Access through parameterized URLs
- Return data in XML or JSON

<u>Example:</u>

MusicBrainz is a Web Service which provides music data.

We can find all artists called “Elvis” in the US through the URL:

[http://musicbrainz.org/ws/2/artist/?query=artist:elvis%20AND%20type:person%20AND%20country:US](http://musicbrainz.org/ws/2/artist/?query=artist:elvis%20AND%20type:person%20AND%20country:US)

## Integrity Constraints

**We suppose the database satisfies a set of constraints**

We can then reason on them

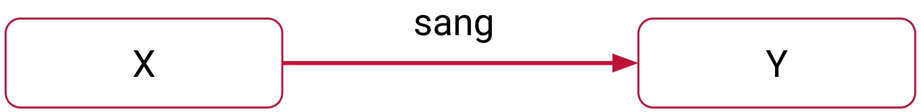
<u>Example:</u>

For a Web service about music, we could have constraints like:

- All singers sing at least one song
- All albums have at least one song
- All artists have at least written one song or sung one song

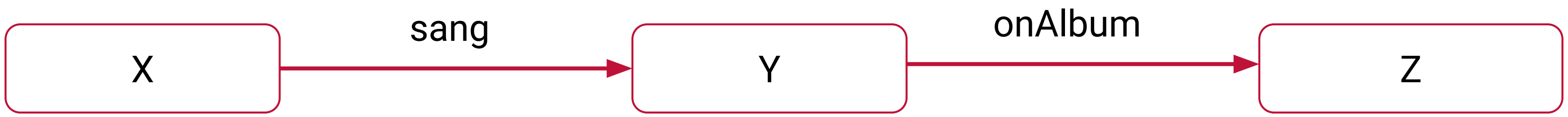
## Example - Constraint on the Data

All nodes with an ingoing *sang* relation have an outgoing *onAlbum* relation

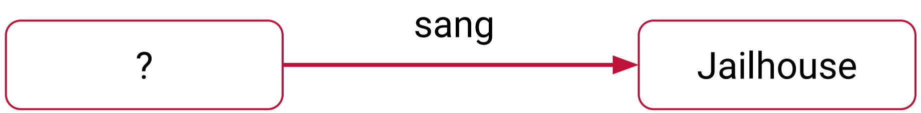


## Example - Data Provider Satisfied Constraint

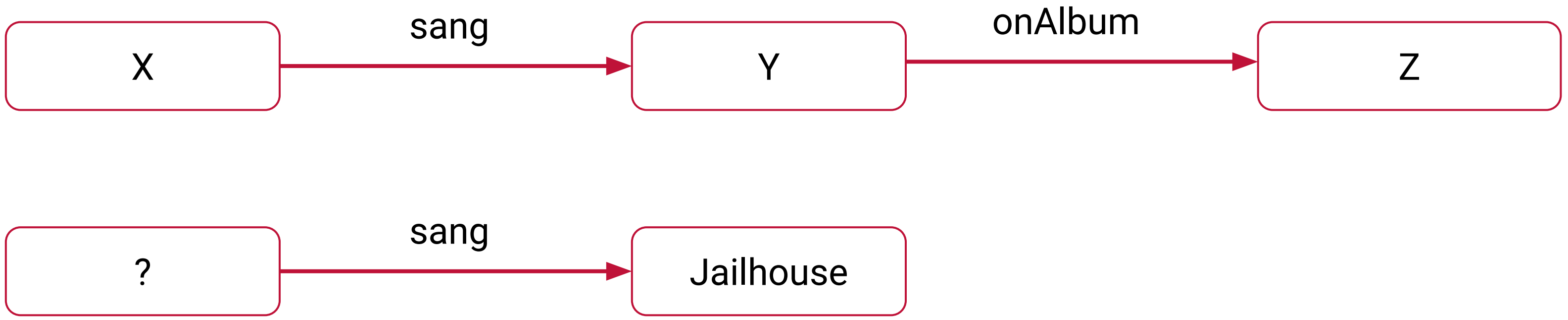
All nodes with an ingoing *sang* relation have an outgoing *onAlbum* relation



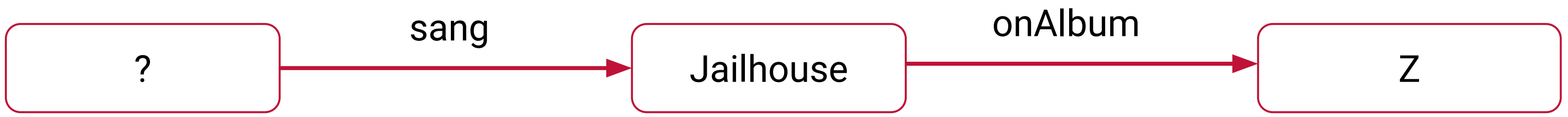
## Example - A Solution Based On the Chase



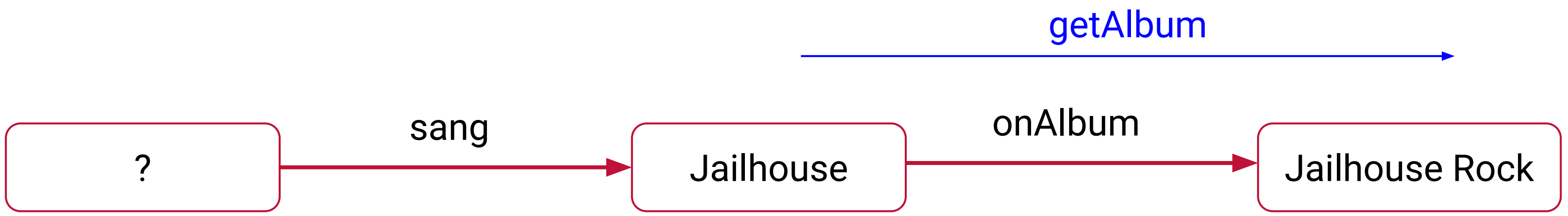
## Example - A Solution - Apply Constraint



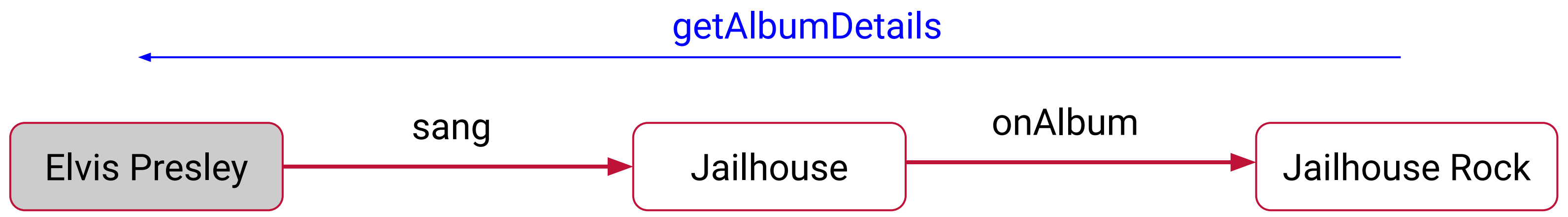
## Example - A Solution - Apply Constraint



## Example - A Solution - Calls to Web Service

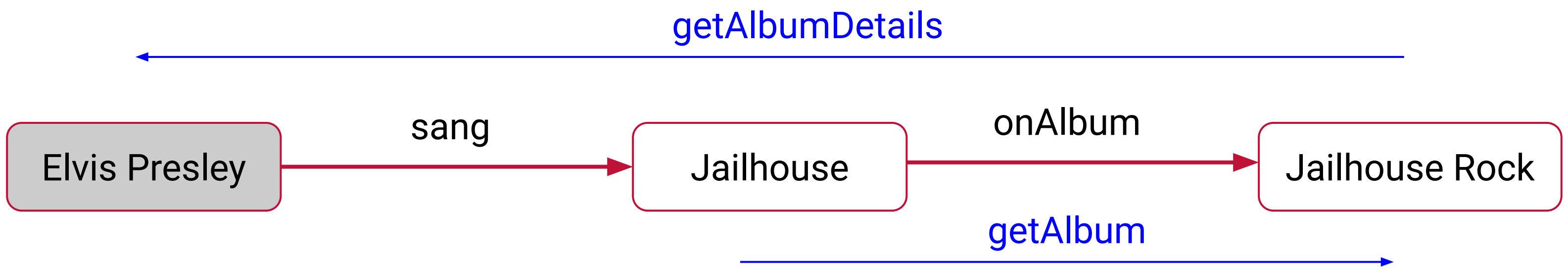


## Example - A Solution - Calls to Web Service



**VICTORY!**

## Example - Is it the only equivalent rewriting?



There might exist several equivalent rewritings for a query.

Depending on the cost of the queries in the plan, we would like to prefer one to the others.

Therefore, we would like to enumerate all possible plans, or the plans with lowest cost.

## General Informal Problem

Given

- a query
- a set of access functions
- a set of constraints

1. Does there exist an equivalent rewriting?

2. If so, is it possible to enumerate all of them?

## Our Main Result

Given

- an atomic query
- a set of access functions that have the shape of a path
- a set of Unary Inclusion Dependencies

1. We can find if there exists an equivalent rewriting in polynomial time

2. It is possible to enumerate all of them (potentially infinitely many).

## Previous Approaches - Chase Based

Using methods introduced by Benedikt et al. ([1], [2]), one can use the Chase algorithm or reason on the Chase to solve the problem.

Intuitively, starting from the initial query, one applies integrity constraints and access methods until one gets the result. It is also possible to avoid materializing the chase in some cases, as it is done in ([1]).

Advantages: General method, automated, finds equivalent rewritings

Drawbacks: Non-polynomial, sometimes does not terminate

[1] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. Querying with access patterns and integrity constraints. PVLDB, 8(6), 2015.
[2] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. PDQ: Proof-driven query answering over web-based data. VLDB, 7(13), 2014

## Previous Approaches - Maximally Contained Rewritings

These methods generate all plans that could potentially yield the result.

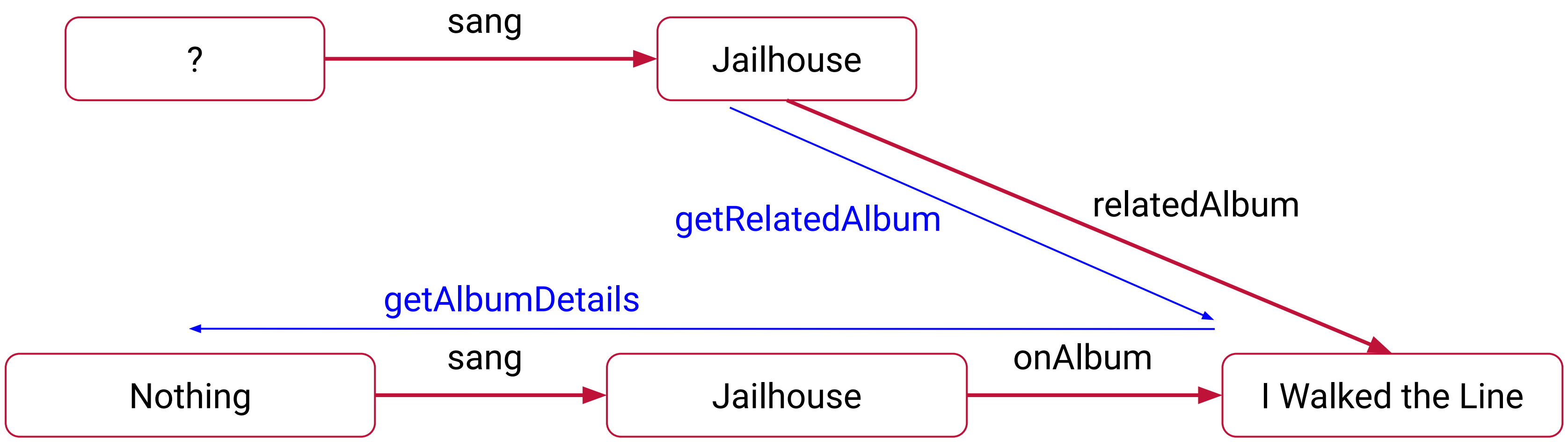
The search space is enormous, so try to restrict to some classes of plans (Susie in [3])

Advantages: Depending on the database, might find results when there is no equivalent rewriting, automated

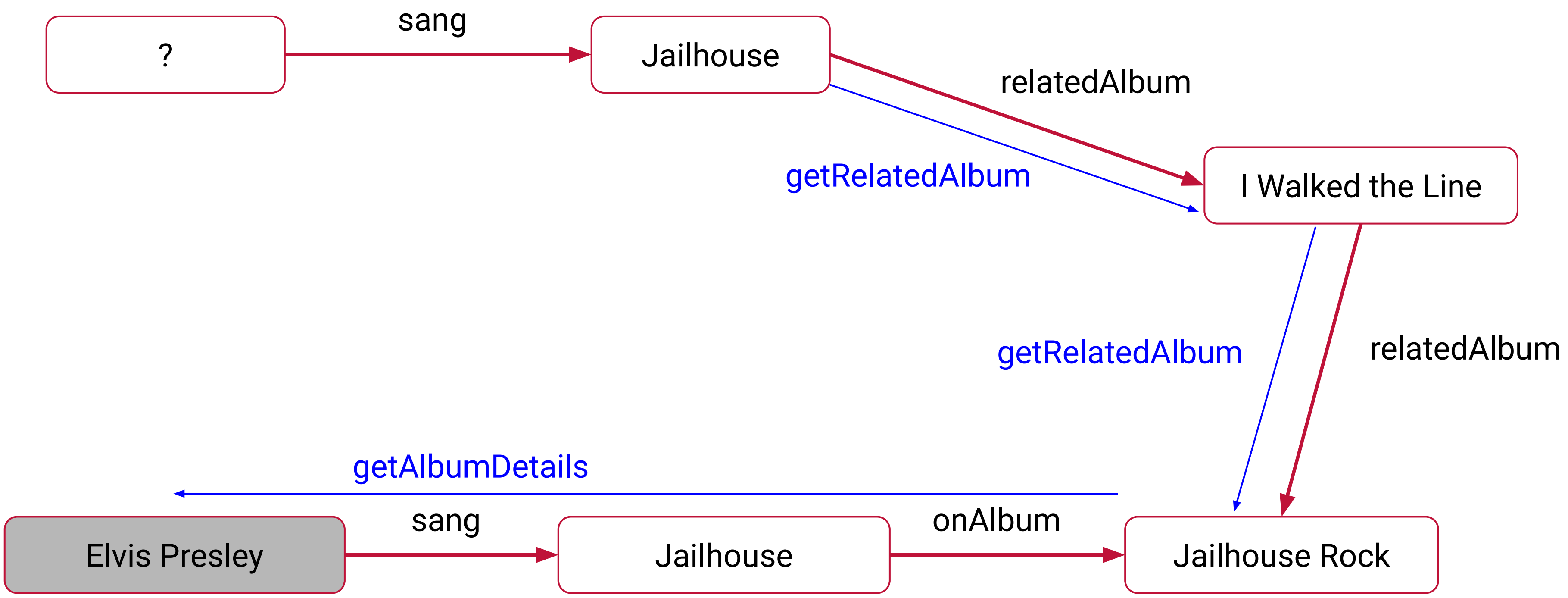
Drawbacks: No guarantee that all results are returned, might not terminate, potentially very expensive to get a result

[3] Nicoleta Preda, Fabian M. Suchanek, Wenjun Yuan, and Gerhard Weikum. SUSIE: Search Using Services and Information Extraction. In ICDE, 2013

## Maximally Contained Rewritings - Example



# Maximally Contained Rewritings - Lucky Example



## Some Definitions

- A **fact** $r(a,b)$, an **inverse fact** $r^{-}(a,b) = r(b,a)$
- A **Unary Inclusion Dependency** (UID): $r \rightarrow s$ , which means $\forall x, y : r(x,y) \Rightarrow \exists z : s(x,z)$
- An **atomic query**: $q(x) \leftarrow r(a,x)$
- A **path function**: $f(\underline{x}, x_{i_1}, \ldots, x_{i_m}) = r_1(\underline{x}, x_1), r_2(x_1, x_2), \ldots, r_n(x_{n-1}, x_n)$
- An **execution plan** $\pi_a(x)$ is a succession of function calls
- $\pi_a(x)$ is an **equivalent rewriting** if for all databases satisfying the UIDs, $\pi_a(x) = q(x)$

## Our Solution - Intuition

- When we have UIDs, the chase has the shape of a tree, where each node is determined by one edge

## Our Solution - Intuition

- When we have UIDs, the chase has the shape of a tree, where each node is determined by one edge

$$r \to s$$

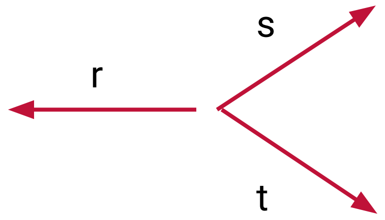
$$r \to t$$

$$t^- \to r^-$$

## Our Solution - Intuition

- When we have UIDs, the chase has the shape of a tree, where each node is determined by one edge

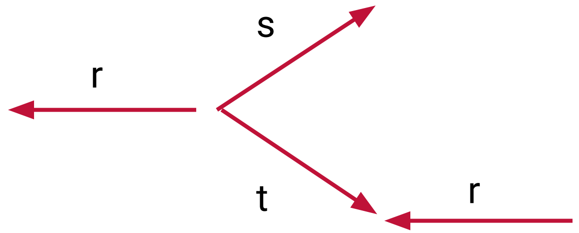
$$r \to s$$
$$r \to t$$
$$t^- \to r^-$$



## Our Solution - Intuition

- When we have UIDs, the chase has the shape of a tree, where each node is determined by one edge

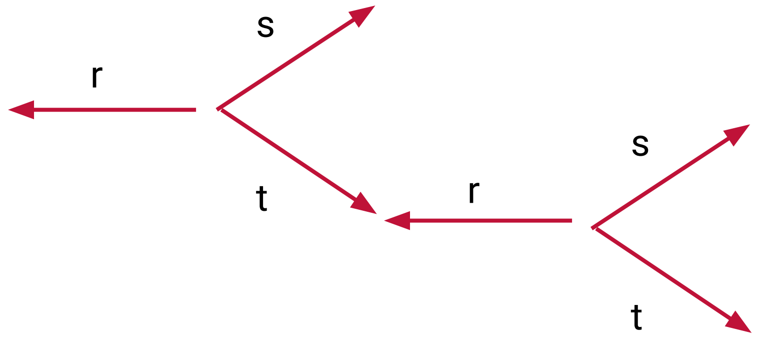
$$r \rightarrow s$$
$$r \rightarrow t$$
$$t^{-} \rightarrow r^{-}$$



## Our Solution - Intuition

- When we have UIDs, the chase has the shape of a tree, where each node is determined by one edge

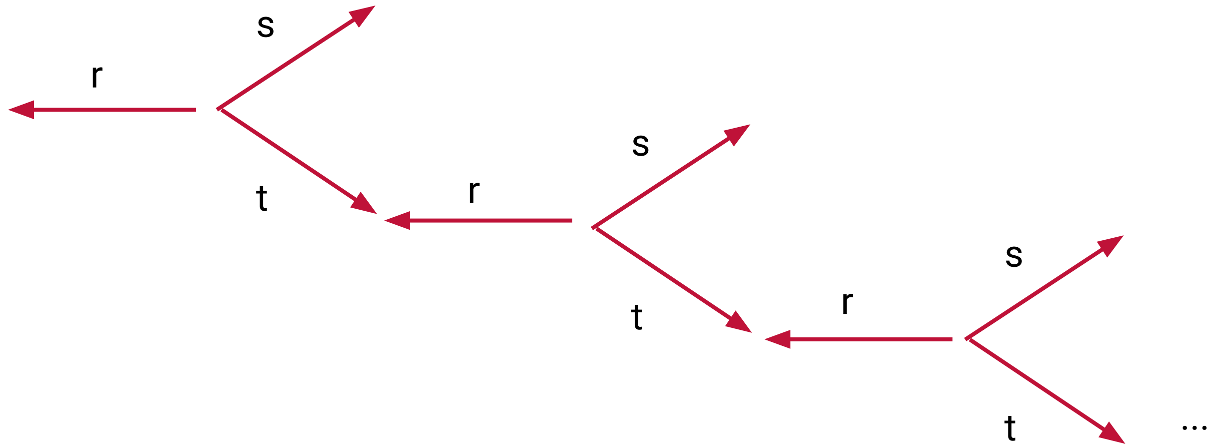
$$r \rightarrow s$$
$$r \rightarrow t$$
$$t^{-} \rightarrow r^{-}$$



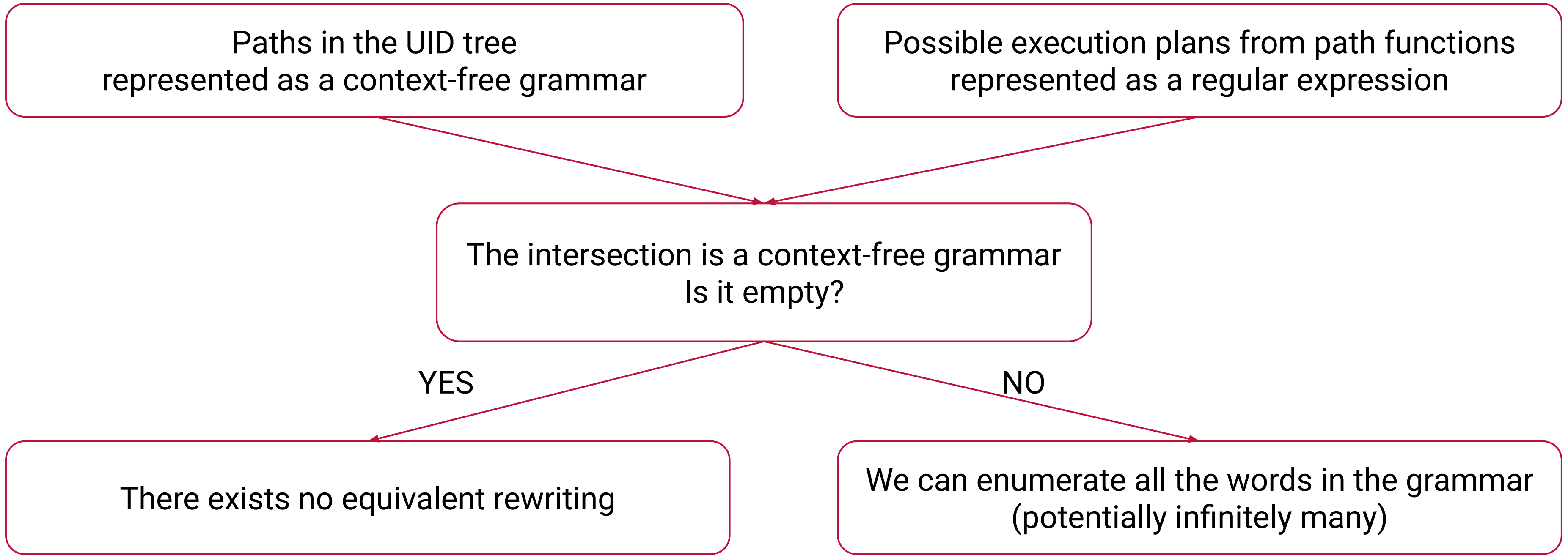
## Our Solution - Intuition

- When we have UIDs, the chase has the shape of a tree, where each node is determined by one edge

$$r \rightarrow s$$
$$r \rightarrow t$$
$$t^- \rightarrow r^-$$



## Our Solution - Intuition



Paths in the UID tree
represented as a context-free grammar

Possible execution plans from path functions
represented as a regular expression

The intersection is a context-free grammar
Is it empty?

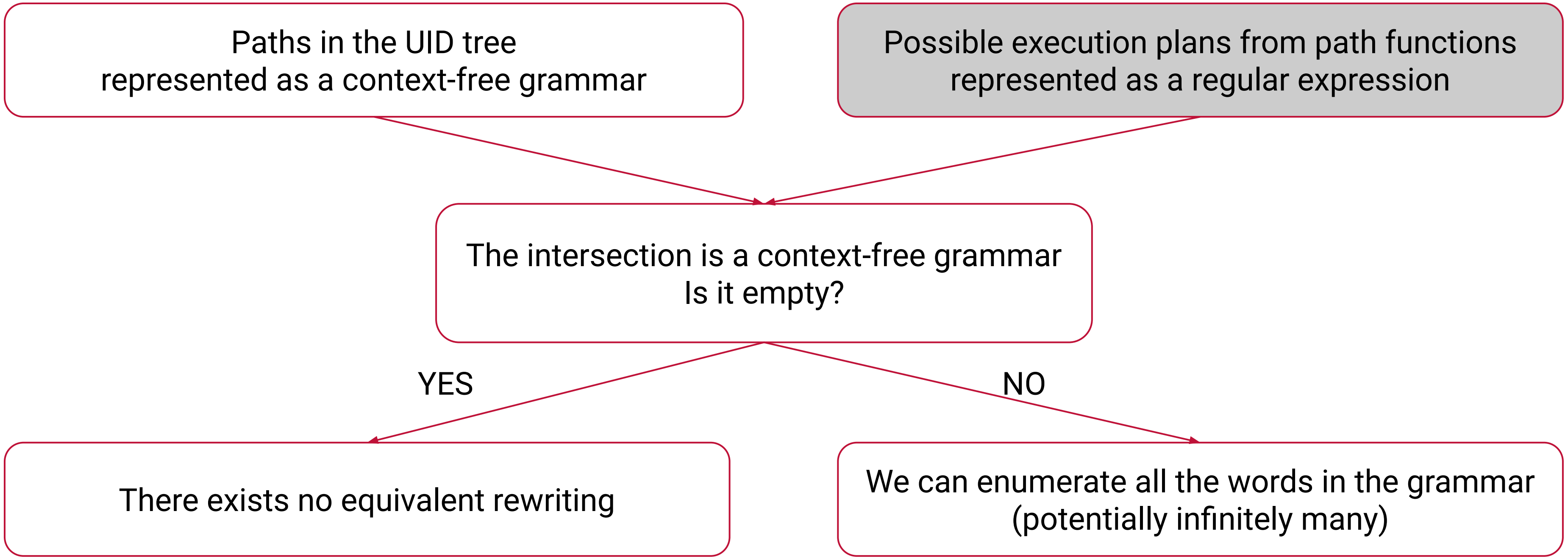
YES

NO

There exists no equivalent rewriting

We can enumerate all the words in the grammar
(potentially infinitely many)

## Our Solution - Intuition


Paths in the UID tree represented as a context-free grammar
Possible execution plans from path functions represented as a regular expression
The intersection is a context-free grammar Is it empty?
YES
NO
There exists no equivalent rewriting
We can enumerate all the words in the grammar (potentially infinitely many)

## Our Solution - Regular Expression of Possible Plans

$$f : r_1(x_0, x_1), \ldots r_n(x_{n-1}, x_n)$$

$x_i$ output variable

$$w_{f,i} = \begin{cases} r_1 \ldots r_i & \text{if } i = n \\ r_1 \ldots r_n r_n^- \ldots r_{i+1}^- & \text{if } 0 \leq i < n \end{cases}$$

Take the disjunction of the $w_{f,i}$ and repeat with a Kleene star

## Our Solution - Regular Expression of Possible Plans

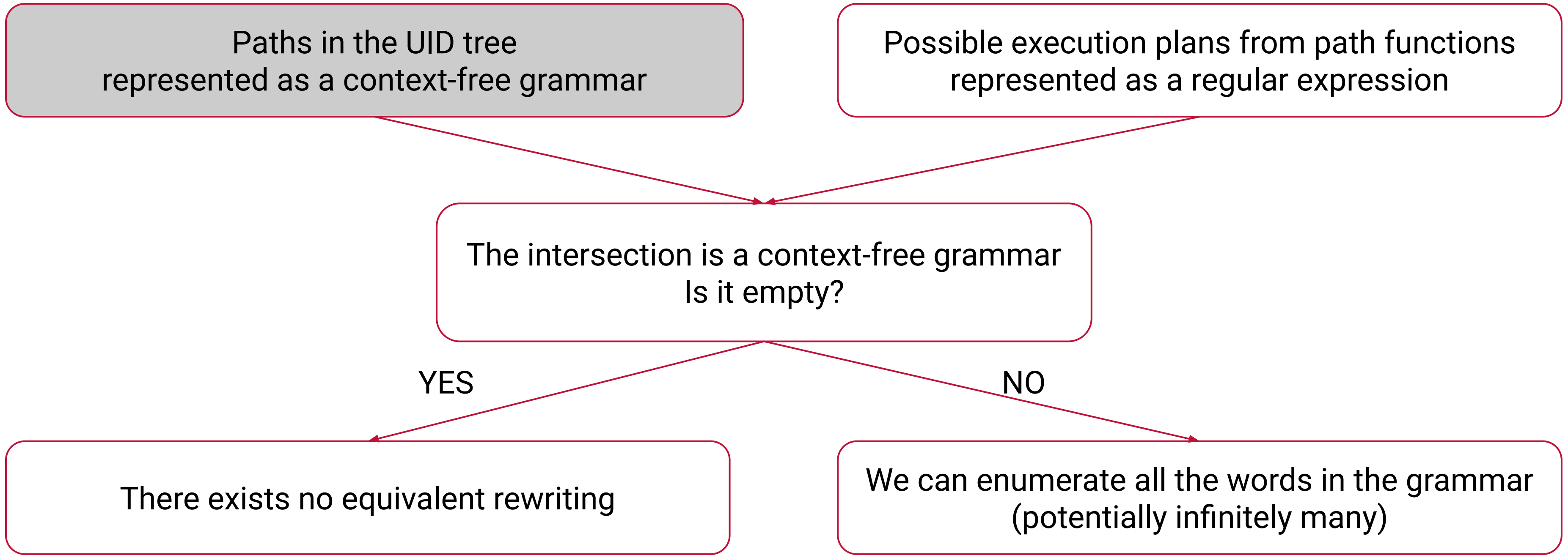
$$f : r_1(x_0, x_1), \dots r_n(x_{n-1}, x_n)$$

$x_i$ output variable

$$w_{f,i} = \begin{cases} r_1 \dots r_i & \text{if } i = n \\ r_1 \dots r_n \underbrace{r_n^- \dots r_{i+1}^-}_{\text{Enforces that the end of the function exists}} & \text{if } 0 \leq i < n \end{cases}$$

Take the disjunction of the $w_{f,i}$ and repeat with a Kleene star

## Our Solution - Intuition

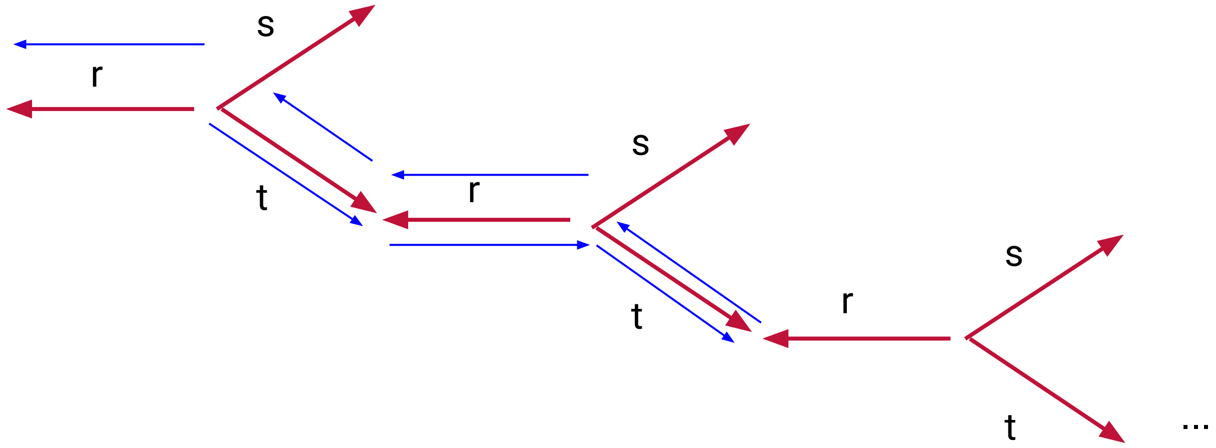

Paths in the UID tree represented as a context-free grammar
Possible execution plans from path functions represented as a regular expression
The intersection is a context-free grammar Is it empty?
YES
NO
There exists no equivalent rewriting
We can enumerate all the words in the grammar (potentially infinitely many)

## Our Solution - Context-Free Grammar - Intuition

- We represent a path in the tree of the UIDs
- A path can explore several branches
- When we get away from the input node, we must come back to answer the query

## Our Solution - Intuition

$$r \to s$$
$$r \to t$$
$$t^- \to r^-$$



## Our Solution - Context-Free Grammar

$$S \to B_r r$$

$$S \to B_r r B_{r^-} r^-$$

$$\forall r_i \rightsquigarrow r_j : B_{r_i} \to B_{r_i} L_{r_j}$$

$$\forall r_i \in \mathcal{R} : B_{r_i} \to \epsilon$$

$$\forall r_i \in \mathcal{R} : L_{r_i} \to r_i B_{r_i^-} r_i^-$$

## Our Solution - Context-Free Grammar

We get the answer by crossing an *r* edge

We get the answer by crossing an inverse *r* edge

$$S \rightarrow B_r r$$

$$S \rightarrow B_r r B_{r^-} r^-$$

$$\forall r_i \rightsquigarrow r_j : B_{r_i} \rightarrow B_{r_i} L_{r_j}$$

$$\forall r_i \in \mathcal{R} : B_{r_i} \rightarrow \epsilon$$

$$\forall r_i \in \mathcal{R} : L_{r_i} \rightarrow r_i B_{r_i^-} r_i^-$$

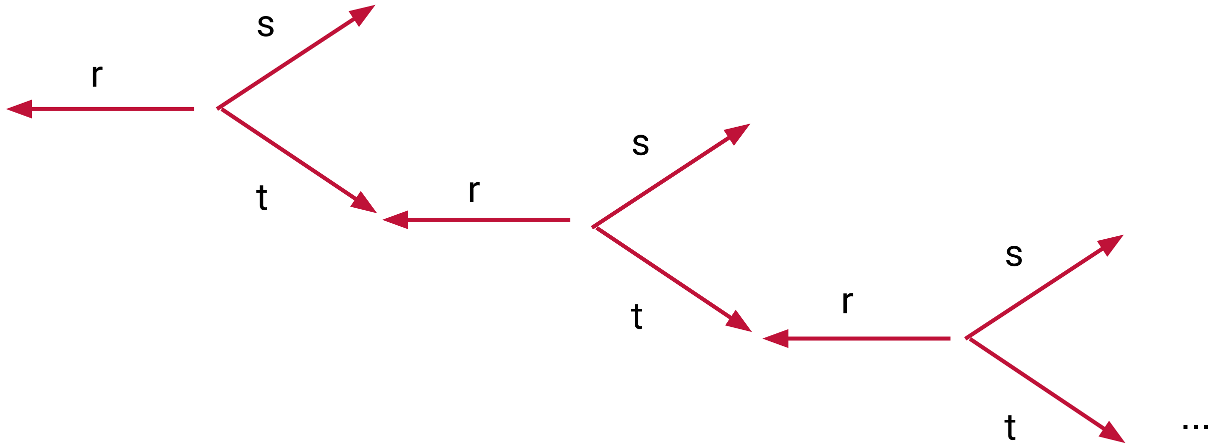
## Our Solution - Intuition - Starting Rule

$r \to s$

$r \to t$

$t^- \to r^-$

$$S \to B_r r$$



## Our Solution - Context-Free Grammar

$$S \rightarrow B_r r$$

In the tree, we are at a node which has a outgoing *r* edge

$$S \rightarrow B_r r B_{r^-} r^-$$

$$\forall r_i \rightsquigarrow r_j : B_{r_i} \rightarrow B_{r_i} L_{r_j}$$

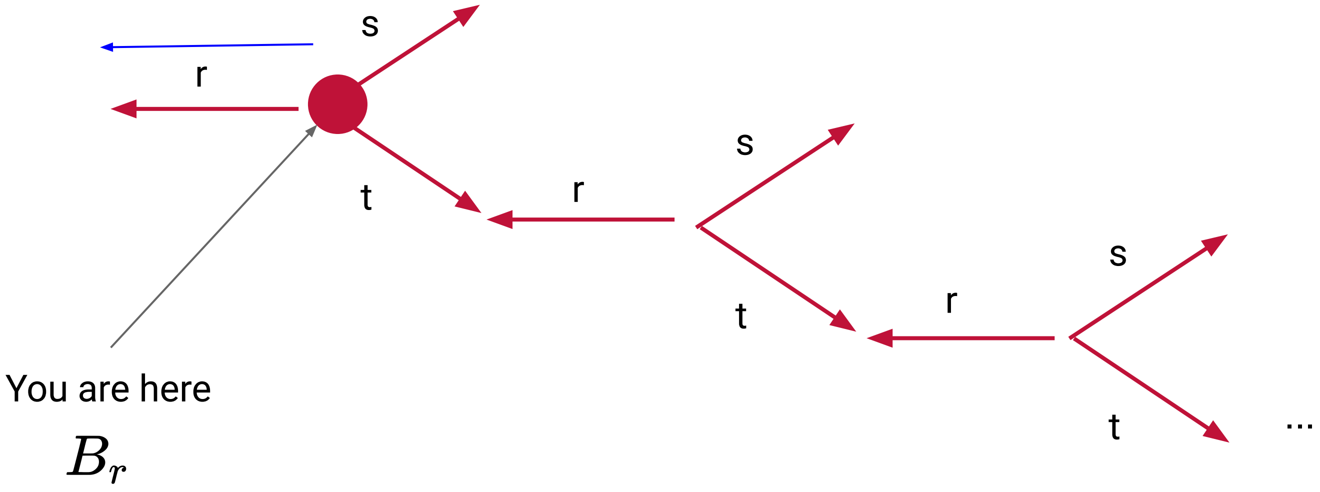
$$\forall r_i \in \mathcal{R} : B_{r_i} \rightarrow \epsilon$$

$$\forall r_i \in \mathcal{R} : L_{r_i} \rightarrow r_i B_{r_i^-} r_i^-$$

## Our Solution - Intuition - Starting Rule

$$r \rightarrow s$$
$$r \rightarrow t$$
$$t^- \rightarrow r^-$$

$$S \rightarrow B_r r$$



## Our Solution - Context-Free Grammar

$$S \rightarrow B_r r$$

$$S \rightarrow B_r r B_{r^-} r^-$$

$$\forall r_i \rightsquigarrow r_j : \boxed{B_{r_i} \rightarrow B_{r_i} L_{r_j}}$$

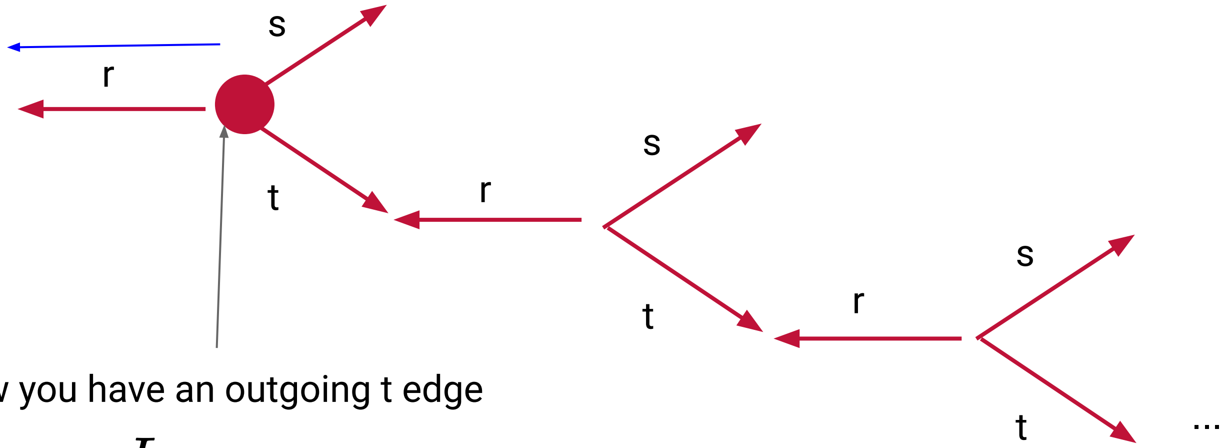
$$\forall r_i \in \mathcal{R} : B_{r_i} \rightarrow \epsilon$$

$$\forall r_i \in \mathcal{R} : L_{r_i} \rightarrow r_i B_{r_i^-} r_i^-$$

We apply a UID: We are still at a node which has an $r_i$ relation and we decide to explore another outgoing edge

## Our Solution - Intuition - Apply Constraints

$$r \rightarrow s$$
$$r \rightarrow t$$
$$t^- \rightarrow r^-$$



You know you have an outgoing t edge

$$L_t$$

## Our Solution - Context-Free Grammar

$$S \rightarrow B_r r$$

$$S \rightarrow B_r r B_{r^-} r^-$$

$$\forall r_i \rightsquigarrow r_j : B_{r_i} \rightarrow B_{r_i} L_{r_j}$$

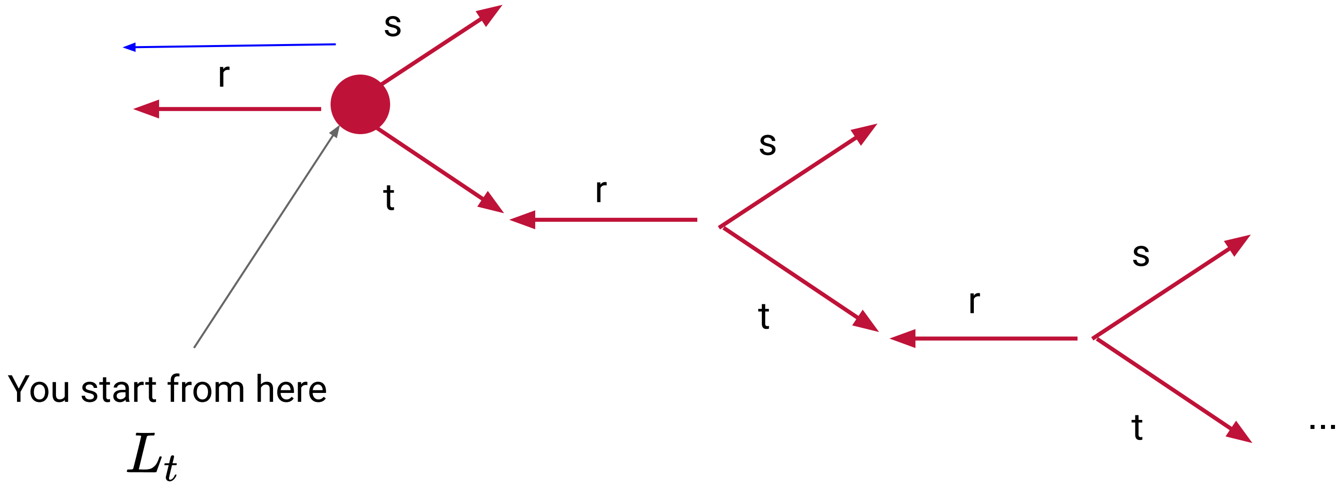
$$\forall r_i \in \mathcal{R} : B_{r_i} \rightarrow \epsilon$$

$$\forall r_i \in \mathcal{R} : L_{r_i} \rightarrow r_i B_{r_i^-} r_i^-$$

We move across a relation, explore what is next, and come back

## Our Solution - Intuition - Cross an Edge

$$r \to s$$
$$r \to t$$
$$t^- \to r^-$$

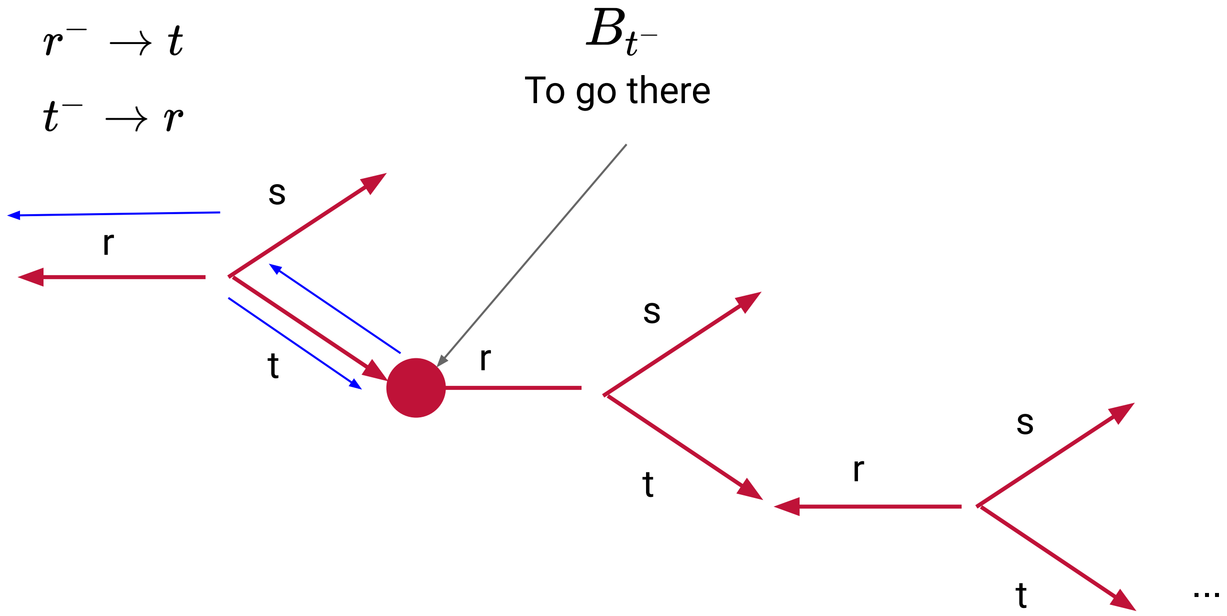


## Our Solution - Intuition - Cross an Edge

$r^- \to s$

$r^- \to t$

$t^- \to r$



## Our Solution - Context-Free Grammar

$$S \to B_r r$$

$$S \to B_r r B_{r^-} r^-$$

$$\forall r_i \rightsquigarrow r_j : B_{r_i} \to B_{r_i} L_{r_j}$$

$$\forall r_i \in \mathcal{R} : B_{r_i} \to \epsilon$$

$$\forall r_i \in \mathcal{R} : L_{r_i} \to r_i B_{r_i^-} r_i^-$$

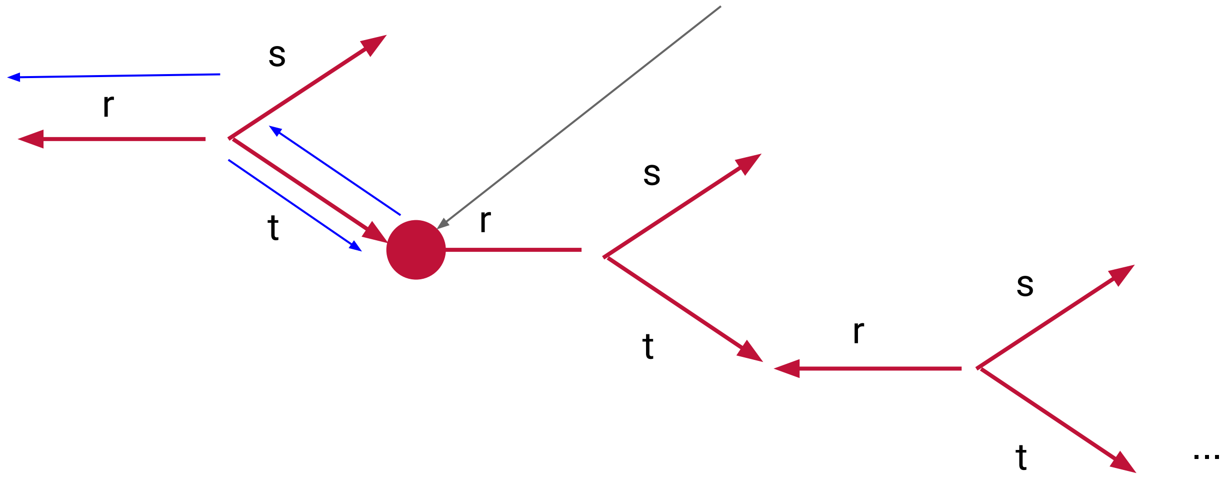
We stop the exploration of the branch

## Our Solution - Intuition - Stop

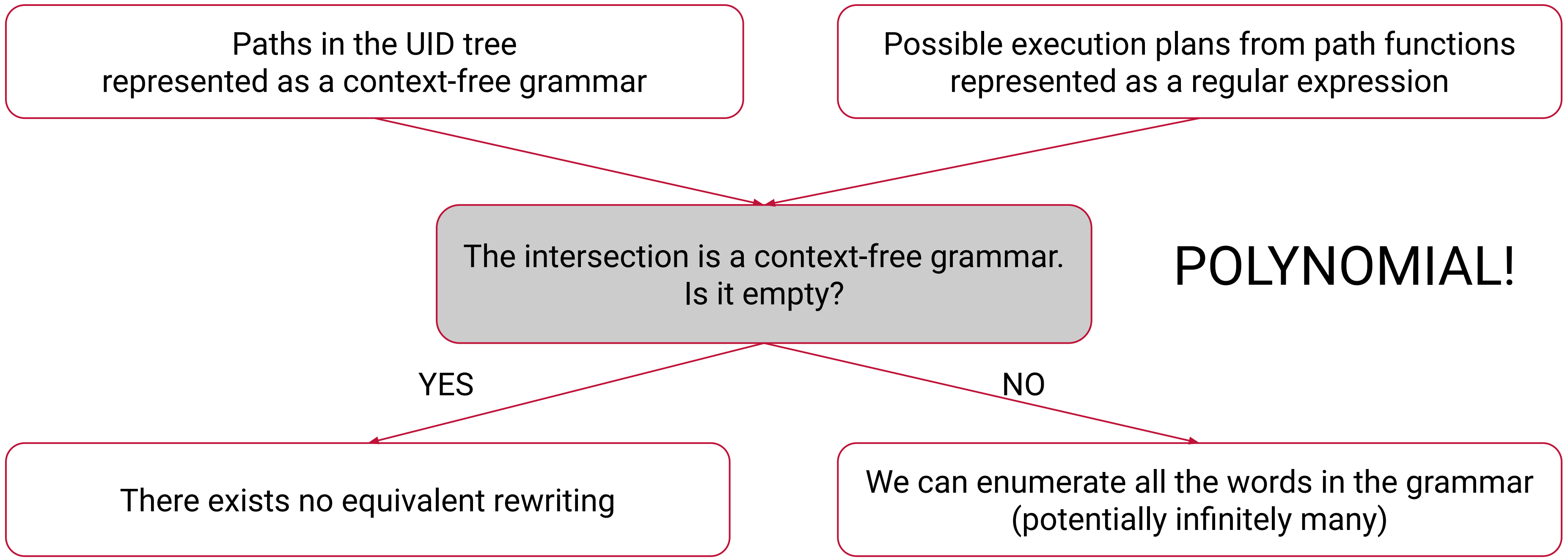
$$r \rightarrow s$$
$$r \rightarrow t$$
$$t^- \rightarrow r^-$$

$B_{t^-}$

You can decide to stop here

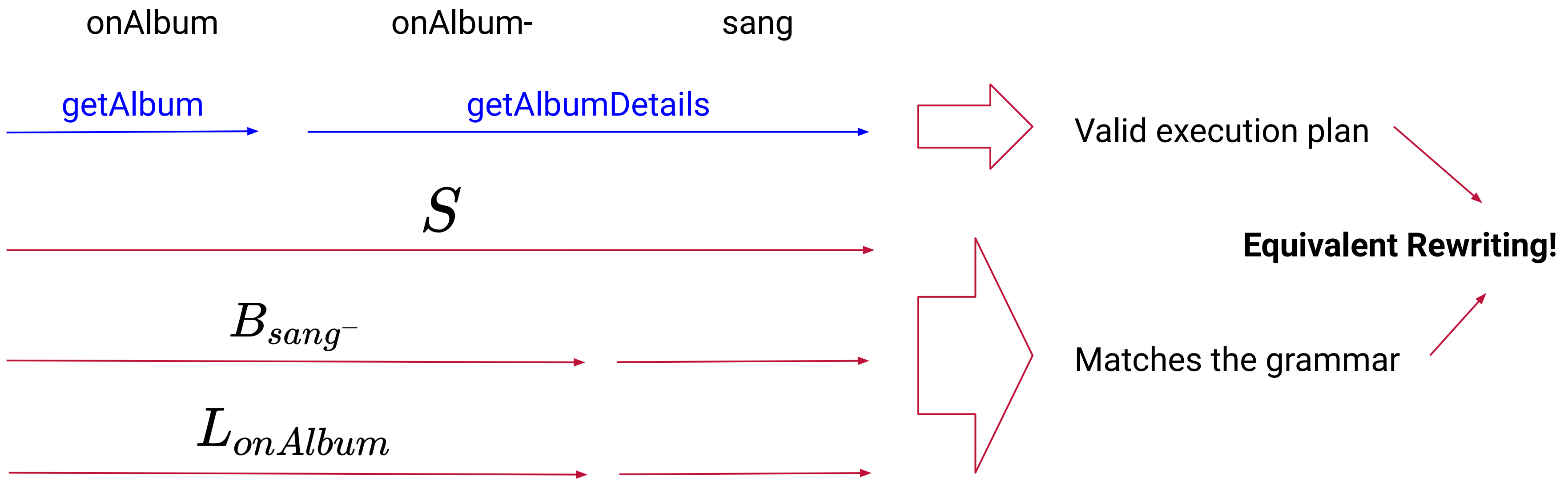


## Our Solution - Intuition

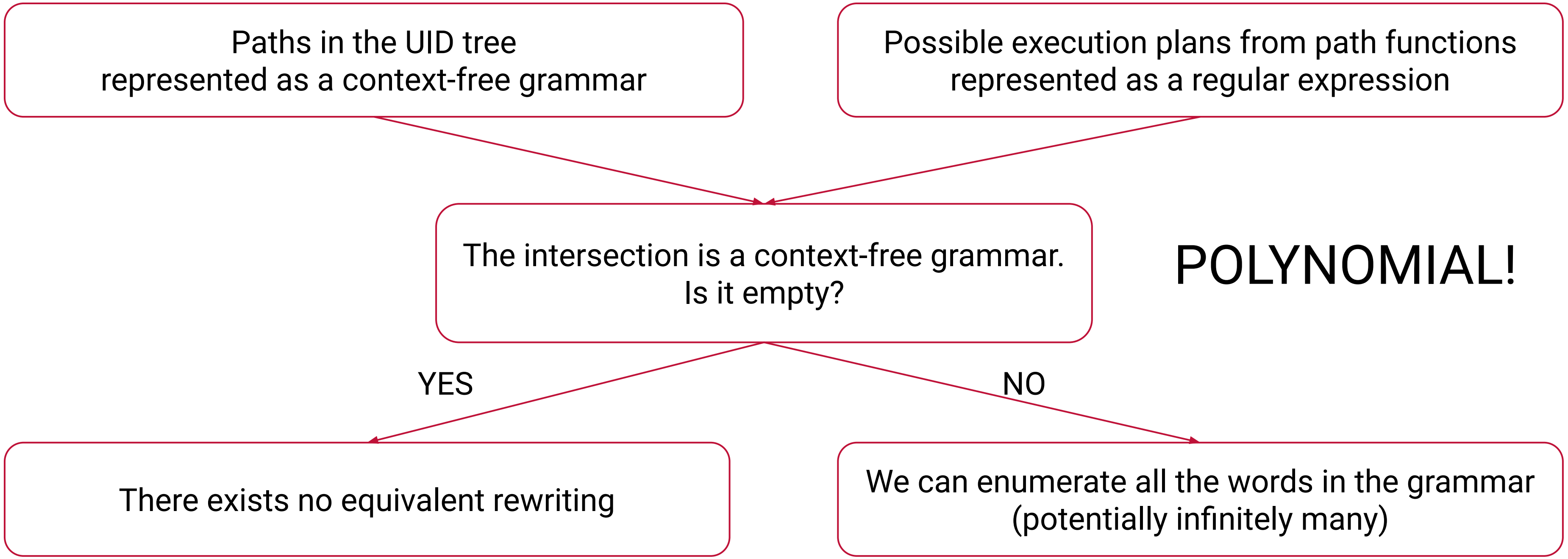

Paths in the UID tree represented as a context-free grammar
Possible execution plans from path functions represented as a regular expression
The intersection is a context-free grammar. Is it empty?
YES
NO
There exists no equivalent rewriting
We can enumerate all the words in the grammar (potentially infinitely many)

POLYNOMIAL!

## Our Solution - Intersection



## Our Solution - Intuition

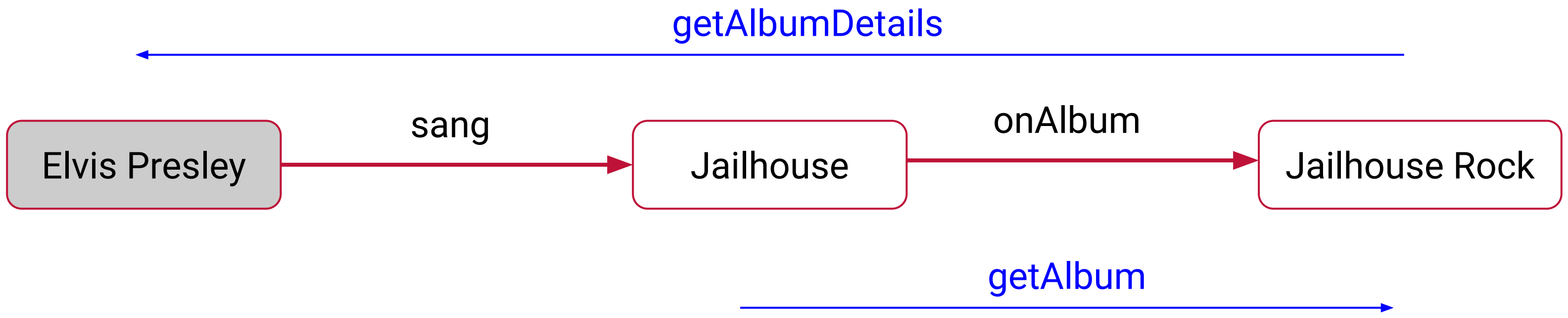

Paths in the UID tree represented as a context-free grammar
Possible execution plans from path functions represented as a regular expression
The intersection is a context-free grammar. Is it empty?
YES
NO
There exists no equivalent rewriting
We can enumerate all the words in the grammar (potentially infinitely many)

POLYNOMIAL!

## EXPERIMENTS

## SUSIE

Susie ([3]) generates all plans such that the last function call contains all the consequences of the previous calls. **It does not use integrity constraints.**

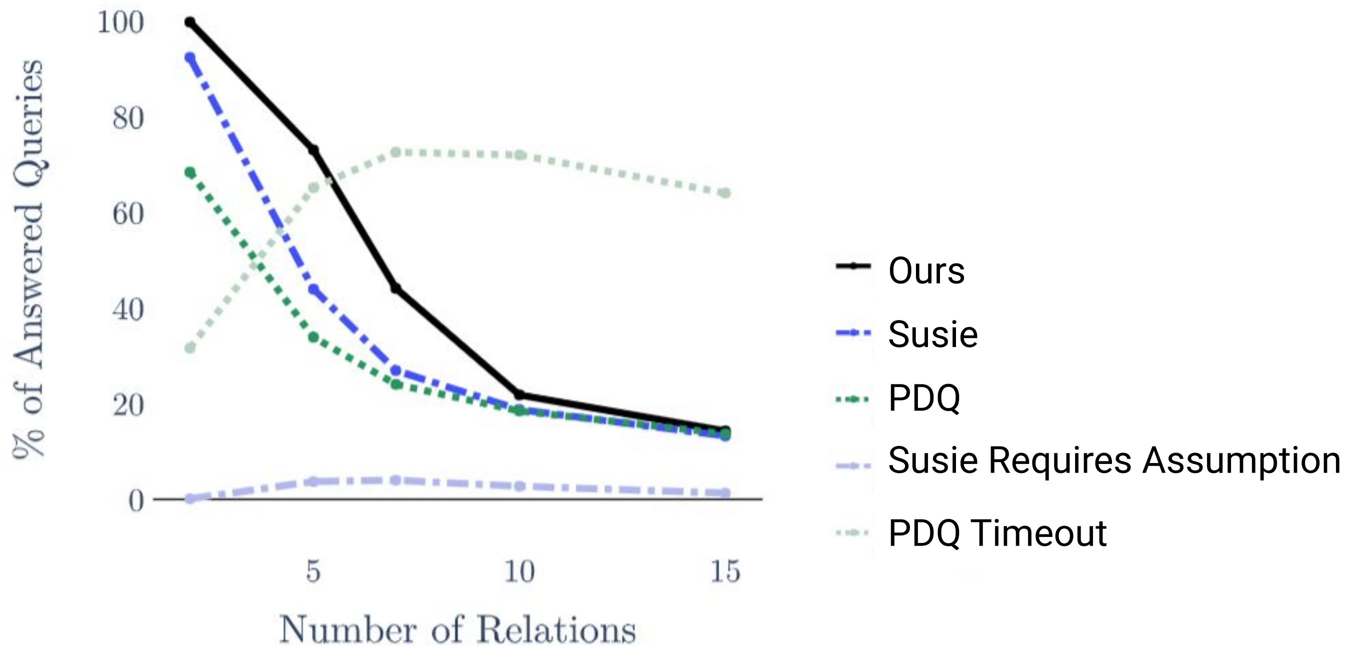


[3] Nicoleta Preda, Fabian M. Suchanek, Wenjun Yuan, and Gerhard Weikum. SUSIE: Search Using Services and Information Extraction. In ICDE, 2013

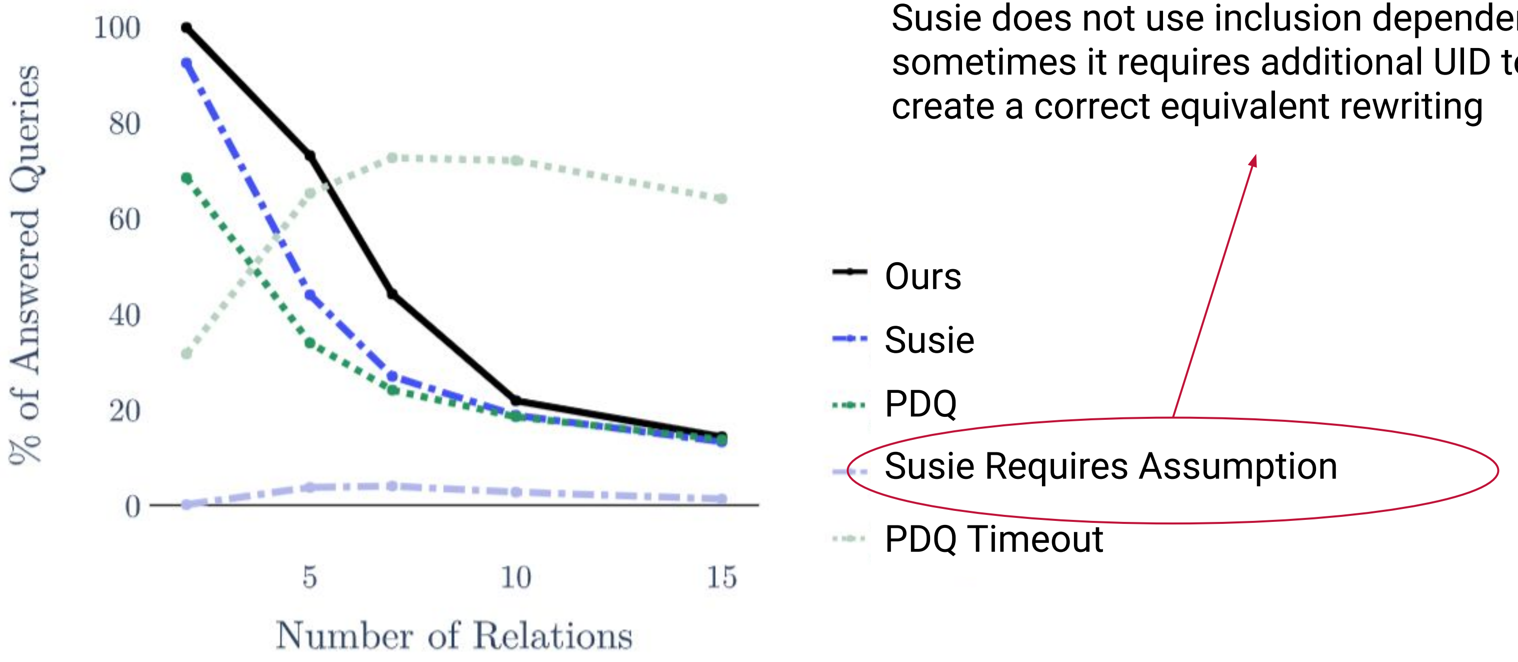
## Experiments - Synthetic Functions

- We generate path functions at random and try to answer a query
- We vary:
  - the number of relations used by the functions
  - the number of functions
  - the probability to have an existential variable in a function
- We compare with:
  - Susie, a Maximally Contained Rewritings algorithm
  - PDQ, a chase based algorithm

## Experiments - Synthetic Functions

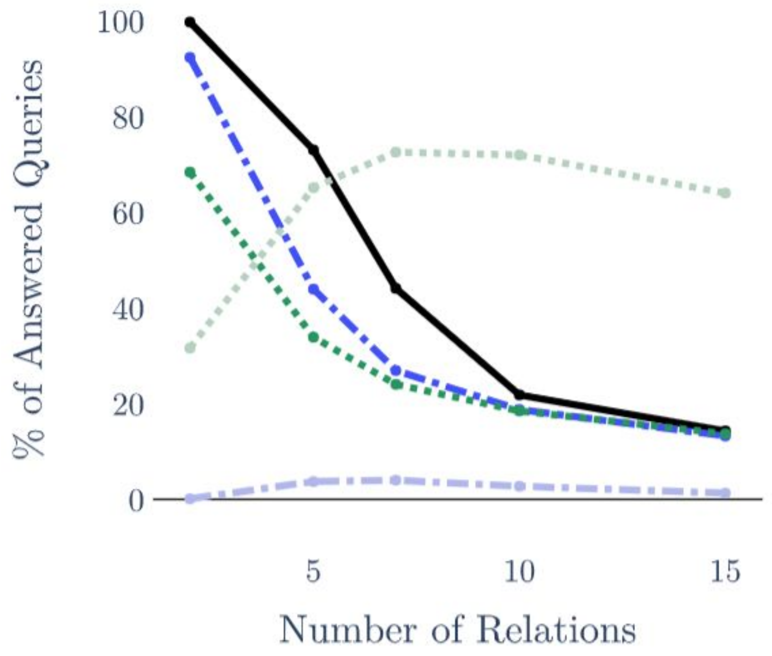


## Experiments - Synthetic Functions



Susie does not use inclusion dependencies, sometimes it requires additional UID to create a correct equivalent rewriting

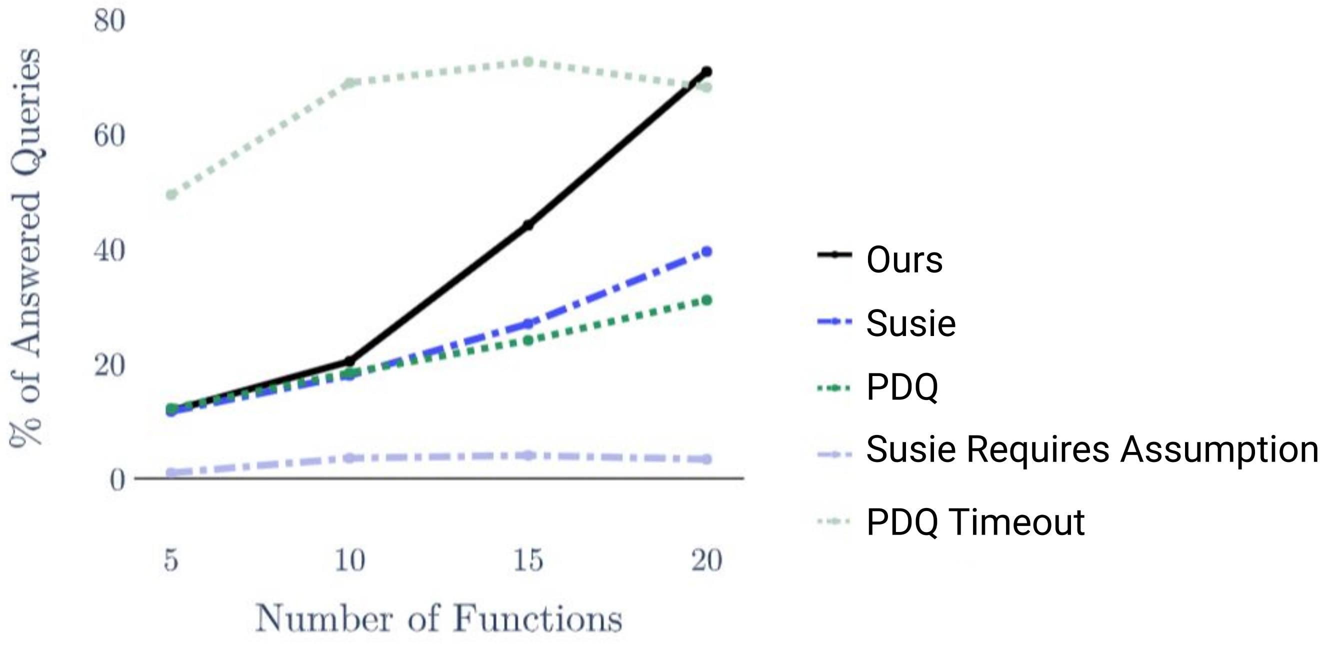
## Experiments - Synthetic Functions



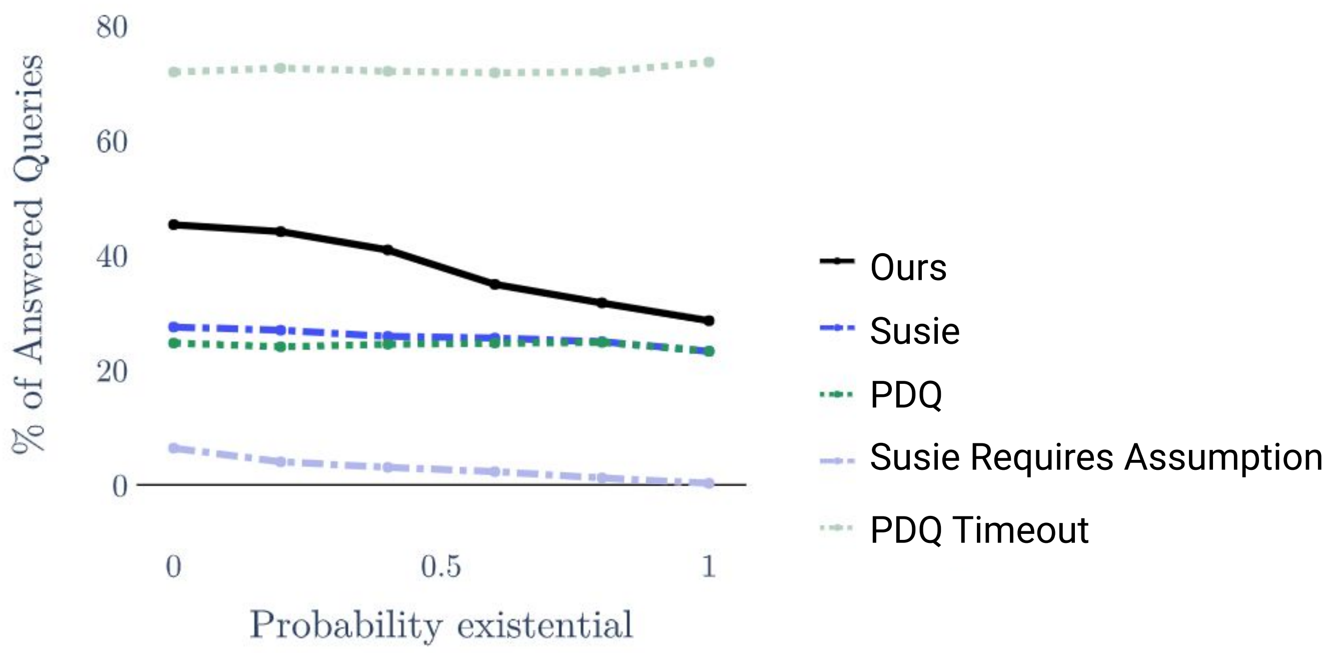
Chase Based Approach, which is not guarantee to terminate (with our problem)

- Ours
- Susie
- PDQ
- Susie Requires Assumption
- PDQ Timeout

## Experiments - Synthetic Functions



# Experiments - Synthetic Functions



## Experiments - Real World Web Services

| Web Service | # Functions | # Relations | Susie | PDQ (timeout) | Ours |
|---|---|---|---|---|---|
| Movies | 2 | 8 | 13% | **25%** (0%) | **25%** |
| Books | 13 | 28 | 57% | 64% (7%) | **68%** |
| Music | 24 | 64 | 22% | 22% (25%) | **33%** |

## Experiments - Real World Web Services

| Web Service | # Functions | # Relations | Susie | PDQ (timeout) | Ours |
|---|---|---|---|---|---|
| Movies | 2 | 8 | 13% | **25%** (0%) | **25%** |
| Books | 13 | 28 | 57% | 64% (7%) | **68%** |
| Music | 24 | 64 | 22% | 22% (25%) | **33%** |

Real-World Web Services provided by Susie

## Experiments - Real World Web Services

| Web Service | # Functions | # Relations | Susie | PDQ (timeout) | Ours |
|---|---|---|---|---|---|
| Movies | 2 | 8 | 13% | **25%** (0%) | **25%** |
| Books | 13 | 28 | 57% | 64% (7%) | **68%** |
| Music | 24 | 64 | 22% | 22% (25%) | **33%** |

We consider all possible atomic queries and report how many can be answered
For PDQ, we impose a time limit of 8 hours

## Experiments - Real World Web Services

| Web Service | # Functions | # Relations | Susie | PDQ (timeout) | Ours |
|---|---|---|---|---|---|
| Movies | 2 | 8 | 13% | **25%** (0%) | **25%** |
| Books | 13 | 28 | 57% | 64% (7%) | **68%** |
| Music | 24 | 64 | 22% | 22% (25%) | **33%** |

- As we guarantee completeness, our numbers indicate the true percentage of answerable queries
- Susie plans are easy to find for PDQ as they quickly appear in the chase
- The harder the problem, the more PDQ timeouts

## Conclusion

- We introduced an exact and tractable method to find equivalent rewriting in practical cases
- We have an extensive theoretical background to support our approach
- We showed how our method outperforms current approaches, both in synthetic and real-world examples

- Technical details are in the extended version of our paper
- We have an online demo: http://dangie.r2.enst.fr/
- The code is on Github: https://github.com/Aunsiels/query_rewriting