Scalable Data Exchange with Functional Dependencies

Bruno Marnette[1,2]
Giansalvatore Mecca[3]
Paolo Papotti[4]

1: Oxford University Computing Laboratory – Oxford, UK
2: INRIA Saclay, Webdam – Orsay, France
3: Universita della Basilicata – Potenza, Italy
4: Universita Roma Tre – Roma, Italy

Data Exchange

Automatic	(rule-based)
and scalable	(very large databases)
exchange of data	(integration, translation)
under constraints	(constrained schemas)

Standard framework

Fagin, Kolaitis, Miller, Popa, ICDT 2003

Constrained schemas

- EGDs: $A(x,y)\ A(x,z) \rightarrow y=z$ (keys, functional dependencies)
- TGDs: $A(x,y) \rightarrow \exists z,\ B(y,z)$ (foreign keys, inclusion dep.)

Source-to-target dependencies

- TGDs: $A(x,y)\ B(y) \rightarrow \exists\ z,\ C(x,z)\ \ D(z)$

Databases with labelled nulls: *#1*,*#2*...

Example

Contact	Address
name, phone	*name, city*

Alice	01543
Bob	06513
Cedric	07945

Alice	Paris
Bob	Rome
Joe	London

$C(x,y) \rightarrow \exists z,\ P(x,y,z)$

$A(x,z) \rightarrow \exists y,\ P(x,y,z)$

$P(x,y,z), P(x,y',z') \rightarrow y=y',\ z=z'$

Person
name, phone, city

Example

Contact	Address
name, phone	*name, city*

Alice	01543
Bob	06513
Cedric	07945

Alice	Paris
Bob	Rome
Joe	London

$C(x,y) \rightarrow \exists z, P(x,y,z)$

$A(x,z) \rightarrow \exists y, P(x,y,z)$

$P(x,y,z), P(x,y',z') \rightarrow y=y', z=z'$

Person
name, phone, city

Desired instance:

Alice	01543	Paris
Bob	06513	Rome
Cedric	07945	#1
Joe	#2	London

Core semantics [FKP'05]

Solution : satisfies all the dependencies

Universality : sound, contained in every solution (~hom)

Core : minimal size, minimal redundancy

properties: existence, uniqueness

Universal Solution

Alice	#0
Alice	#1
Cedric	Denis
Cedric	#2
Denis	Elise
Cedric	#3
#3	Elise
#4	Elise

OPTIMISATION →

(-60% size)

Alice	#0
Cedric	Denis
Denis	Elise

Core

State of the art

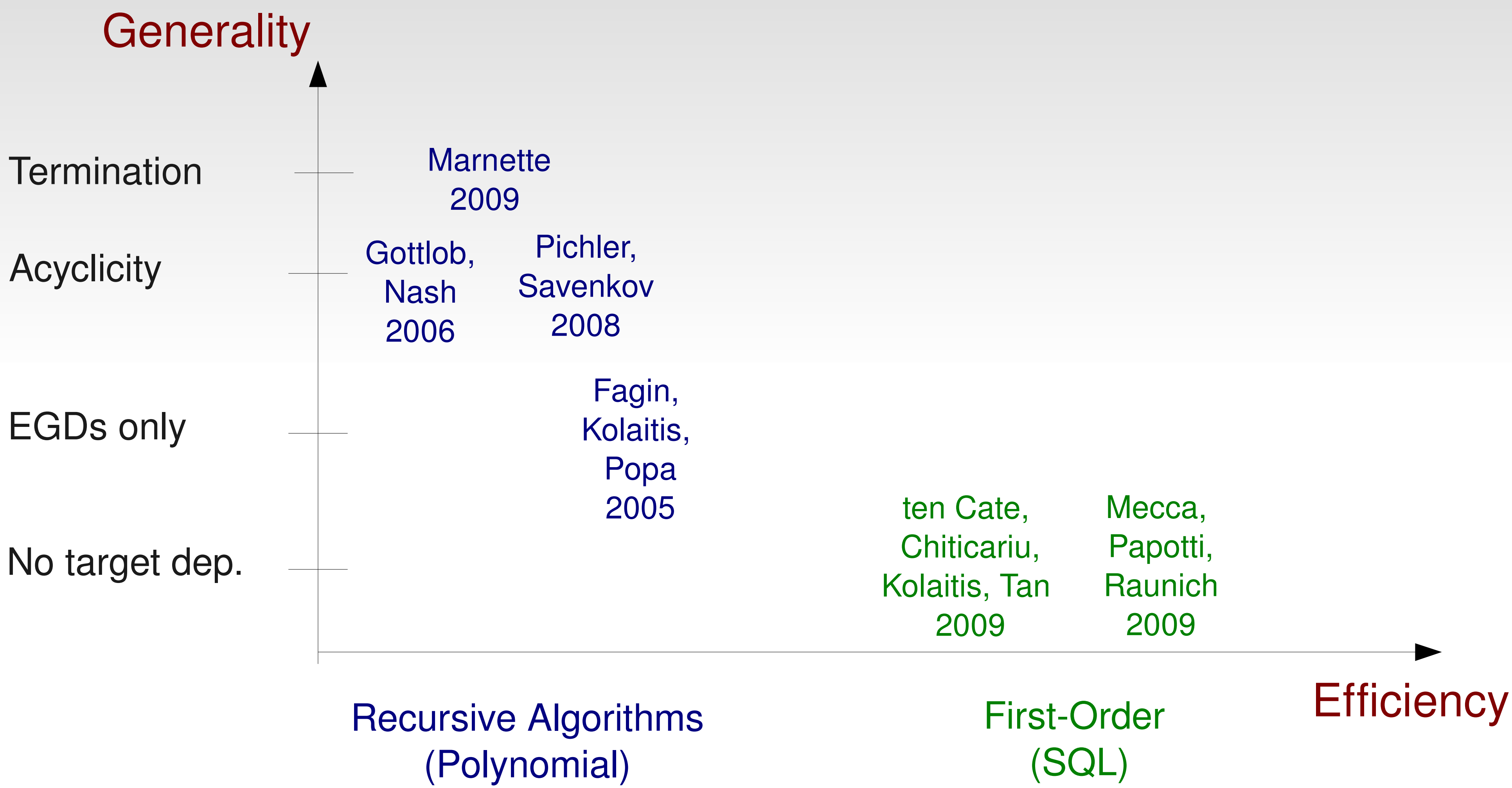
Computing universal solutions:

- A rich literature, not discussed here (see previous talk)
- Several algorithms, based on *chase procedures*
  - Efficient systems, such as Clio [PVMHF'02]

Computing core solutions:

- Two kinds of algorithms:
  - Recursive optimisation of a universal solution
  - Direct computation with SQL of a core solution

Core-computation algorithms



Our previous contributions

Marnette, PODS 2009

- Target dependencies (FDs, IDs, Keys..)
- Assumption: the *oblivious chase* terminates
- Polynomial core-computation (data-complexity)

Pros: a general polynomial-time setting
Cons: not scalable (possibly hours, for <10K tuples)

Our previous contributions

Mecca, Papotti, Raunich, SIGMOD 2009

+Spicy

Input: scenarios without target dependencies

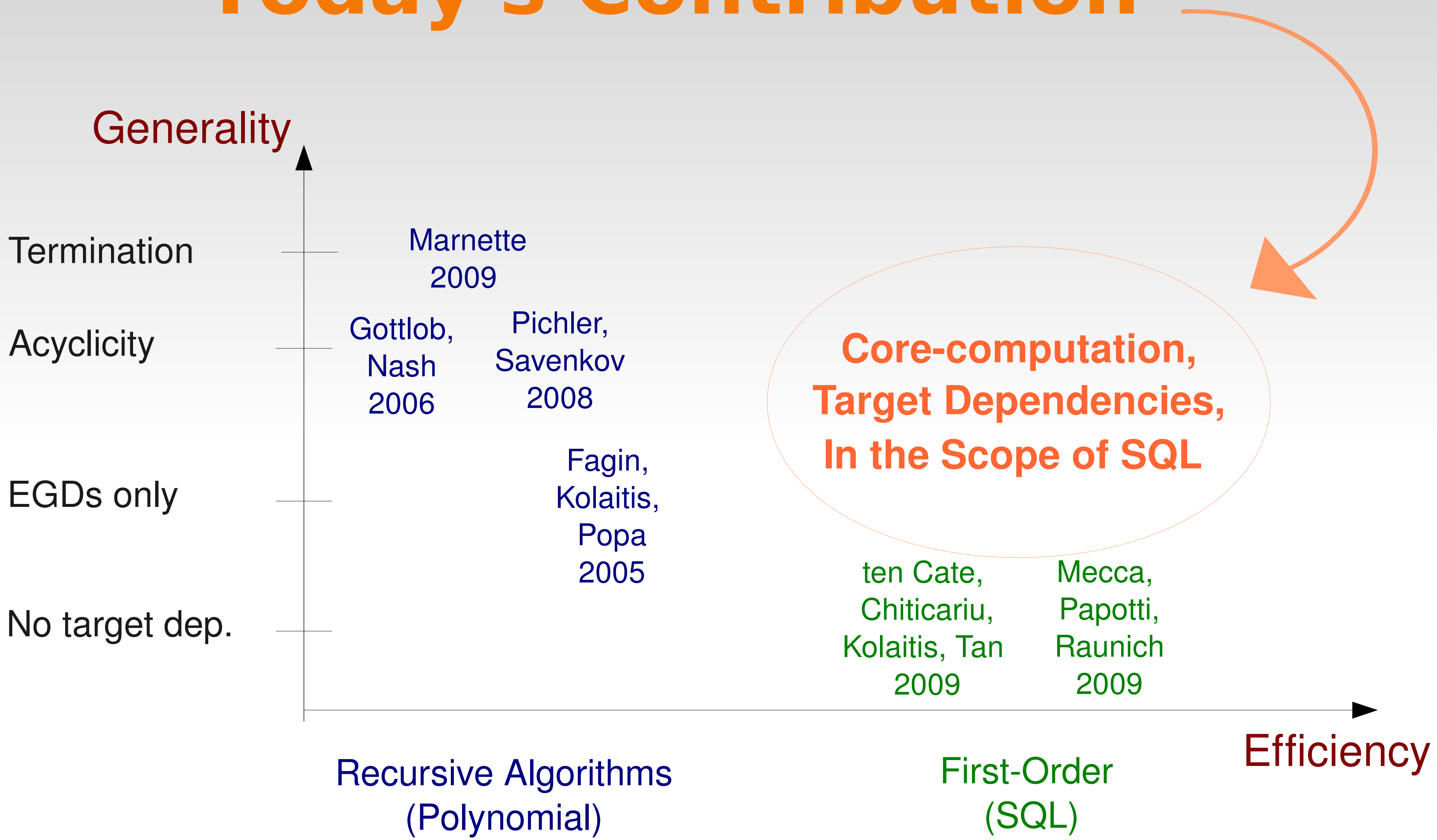
Output: SQL script to compute core soltions

- Non-trivial existence! (also shown in [tCCKT'09])
- Efficient rewriting, SQL scripts of good quality

Pros: scalability (few seconds, for milions of tuples)

Cons: no target dependencies

Today's Contribution



Outline of the talk

1) Theoretical limits and practical approach

2) System +*Spicy* and experimental results

3) Some intuition on the new algorithms

Theoretical limits and practical approach

Data Exchange with SQL

Compilation: Given scenario M, compute an SQL-script P_M

- Set of rules of the form *Body* → *Head*
- *Body:* first order query, over the source schema
- *Head* : atoms with skolem functions, over the target

Execution and properties:

- Given I, executing P_M produces a target instance $P_M(I)$
- Correctness: For every consistent source instance I $P_M(I)$ is a **universal solution** for I and M
- Optimality: In addition, $P_M(I)$ is always a **core**

Example

Script P_M

$$\text{Contact}(x,y) \wedge \text{Address}(x,z) \rightarrow \text{Person}(x,y,z)$$
$$\text{Contact}(x,y) \wedge \neg \text{Address}(x,z) \rightarrow \text{Person}(x,y,F(x))$$
$$\text{Address}(x,z) \wedge \neg \text{Contact}(x,y) \rightarrow \text{Person}(x,G(x),z)$$

Example

M { Contact (*name, phone*), Address (*name, city*) } → Person (*name, phone, city*)

Script P_M:

- Contact(x,y) ∧ Address(x,z) → Person(x,y,z)
- Contact(x,y) ∧ ¬ Address(x,z) → Person (x,y,*F*(x))
- Address(x,z) ∧ ¬ Contact(x,y) → Person(x,*G*(x),z)

I =

Alice	01543
Bob	06513
Cedric	07945

Alice	Paris
Bob	Rome
Joe	London

$P_M(I)$ =

Alice	01543	Paris
Bob	06513	Rome
Cedric	07945	*F*(Cedric)
Joe	*G*(Joe)	London

Theory and practice

Theorem: There are scenarios with only one functional dependency for which no correct SQL-script exists.

- s-t TGD: Friend(x,y) → ∃g, Group(x,g) ∧ Group(y,g)
- target FD: Group(x,g1) ∧ Group(x,g2) → g1=g2
- Need to compute the connected components of Friend

No need to give up:

- Claim: many real-life scenarios remain in the scope of SQL
- Supported by our first experiments (literature, benchmark)
- We have algorithms to recognize the bad cases

Our approach

What we tried first

- Looked for a simple *syntactic* restrictions, ensuring the existence of a correct SQL-script
- One possibility: *variables affected by at most one FD*
- Nice on paper, but too restrictive in practice

A more general approach

- Given M, compute a "good" SQL-script P_M
- Check that this script P_M is correct
- Rewrite P_M into an optimal script, which compute cores

System *+Spicy* and experimental results

The new version of +*Spicy*

Previous version, SIGMOD 2009

- Given a scenario without target dependencies, always generates a correct and optimal SQL-script

Now with target dependencies

- Does the same thing for many scenarios with target dependences (while rejecting the problematic ones)
- Good support for Keys and Functional Dependencies
- Work in progress: Foreign Keys (~Clio [PVMHF'02])

Experimental Results

Choice of scenarios

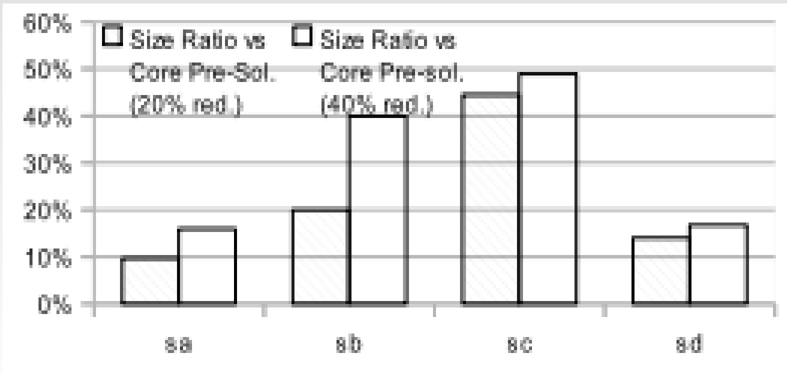
- From the litterature on data echange and core-computation
- Generated with the STBenchmark [Alexe et al, VLDB'08]

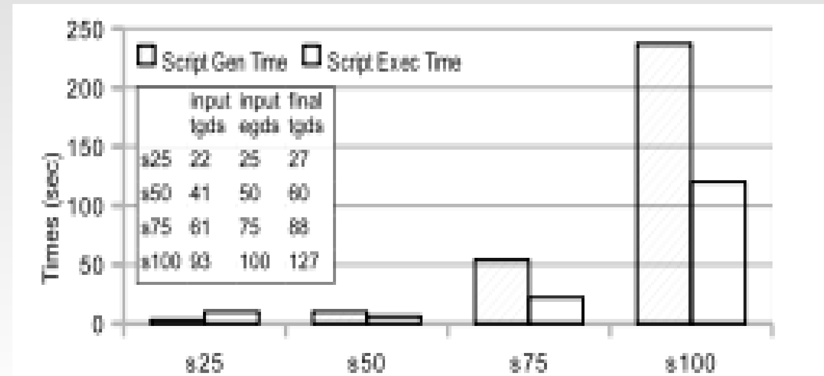
Main results

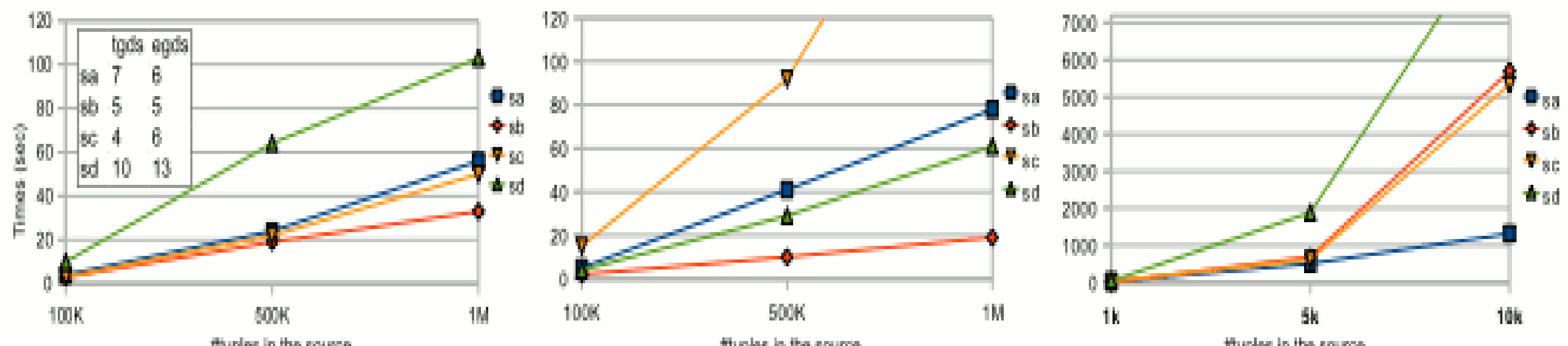
- Most senarios in the scope of SQL, despite target dependencies
- Quality of solutions: size reduction 10%-60%
- Script generation: 2s-4m for 22-93 tgds and 25-100 egds
- Script execution: 5s-100s for DBs with 100K-1M tuples

  Drastically faster than recursive algo (>1hour for 10K tuples)

More details in the paper







a. Rewriting-based SQL script (egd-compliant core solution)

b. Tgd-based SQL script (core pre-solution, non egd-compliant)

c. Custom egd chase Engine (1k-10k tuples only) (egd-compliant canonical solution)

Some intuition on the new algorithms

Main Algorithms

Input Scenario M
(set of dependencies)

(1) Overlap Computation
(2) Refined Skolemisation

↓

SQL-script P_M

(3) Test of Correctness
(4) Optimal Rewriting

↓

SQL-script P'_M

> P'_M is a correct and optimal SQL-script for M

(1) Overlap Computation

- Overlap: when two atoms in the head of two rules “unify”
- Combine rules to create new rules, with negation in the body
- Key idea: avoid introducing nulls at the affected positions

$$\text{Contact}(x,y) \rightarrow \exists z,\ \text{Person}(x,y,z)$$
$$\text{Address}(x,z) \rightarrow \exists y,\ \text{Person}(x,y,z)$$
$$\text{Person}(x,y,z), \text{Person}(x,y',z') \rightarrow y=y',\ z=z'$$

$$\text{Contact}(x,y) \wedge \text{Address}(x,z) \rightarrow \text{Person}(x,y,z)$$
$$\text{Contact}(x,y) \wedge \neg \exists z\ \text{Address}(x,z) \rightarrow \exists z,\ \text{Person}(x,y,z)$$
$$\text{Address}(x,z) \wedge \neg \exists y\ \text{Contact}(x,y) \rightarrow \exists y,\ \text{Person}(x,y,z)$$

(2) Refined Skolemisation

- Skolemisation: replace variables by uninterpreted functions
- Naïve skolemisation: function of *all* the universal variables
- The target FDs allow to find better candidates (less variables)
- Our algorithm looks both at the target FDs and the source FDs

$$A(x,y,z) \rightarrow \exists e,\ B(x,e) \wedge C(y,e)$$ (with three FDs)

$$A(x,y,z) \rightarrow B(x,F(y)) \wedge C(y,F(y))$$ $(x \rightarrow y \rightarrow F(y))$

(3) Test of Correctness

By construction of P_M: for all I, the instance P_M(I) is *sound*

It remains to check that: for all I, the instance P_M(I) is a *solution*

The problem would be undecidable if P_M was too complicated...

Theorem: The following problem is decidable in our setting: given M and the corresponding P_M, is P_M correct?

- Key argument: *small model property*
- Exponential algorithm (not too bad in practice)

(4) Optimal Rewriting

Theorem: Given a correct SQL-script P_M for a scenario M
we can always compute an optimal SQL-script

- A non-trivial generalisation of [MPR'09], [tCCKT'09]
  - Strongly based on a *bounded block-width* property
  - A null can not appear in more than k atom
- We support scenarios with target FDs and unbounded blocks
- Technical proof (sketched in the paper)

Example (unbounded blocks)

Scenario M

Mother
name , name

→

Parent	Child
name, FID	*FID , name*

Mother(x,y) → ∃ z, Parent(x,z) ∧ Child(z,y)

Parent(x,y_1) ∧ Parent(x,y_2) → $y_1 = y_2$

Script P_M

Mother(x,y) → Parent(x,*F*(x)), Child(*F*(x),y)

I =

Alice	Bob
Sophie	Cedric
Sophie	Denis

P_M(I) =

Alice	*#1*
Sophie	*#2*

#1	Bob
#2	Cedric
#2	Denis

Where *#1* = *F*(Alice) and *#2* = *F*(Sophie)

Conclusion

Experimental evidence

- Real-life scenarios with FDs can be managed with SQL
- Computing cores is very feasible (scales well)

Technical results

- Algorithms of general interest (overlaps, skolemisation)
- The property of correctness is decidable (in our setting)
- Correct SQL-scripts can always be optimised (cores)

Future work

- Other target dependencies (Inclusion Dependences)
- Alternative languages (fixed-points, stratified Datalog)

The End, Thank You

Experimental evidence

- Real-life scenarios with FDs can be managed with SQL
- Computing cores is very feasible (scales well)

Technical results

- Algorithms of general interest (overlaps, skolemisation)
- The property of correctness is decidable (in our setting)
- Correct SQL-scripts can always be optimised (cores)

Future work

- Other target dependencies (Inclusion Dependences)
- Alternative languages (fixed-points, stratified Datalog)