What we talk about when we talk about graphs

George Fletcher

Eindhoven University of Technology
The Netherlands

GraphQ 2013

Genoa, Italy

22 March 2013

What we talk about when we talk ...

Sapir-Whorf: "the structure of a language affects the ways in which its speakers conceptualize their world" (Wikipedia)

- Wilhelm von Humboldt (1767-1835): linguistics and philology
  - *The heterogeneity of language and its influence on the intellectual development of mankind* (1836)

What we talk about when we talk ...

Sapir-Whorf: "the structure of a language affects the ways in which its speakers conceptualize their world" (Wikipedia)

- Wilhelm von Humboldt (1767-1835): linguistics and philology
  - *The heterogeneity of language and its influence on the intellectual development of mankind* (1836)
- Franz Boas (1858-1942): anthropology
- Edward Sapir (1884-1939) and Benjamin Whorf (1897-1941): linguistics
  - *Language, mind, and reality* (1942)

What we talk about when we talk ...

Sapir-Whorf: "the structure of a language affects the ways in which its speakers conceptualize their world" (Wikipedia)

- Wilhelm von Humboldt (1767-1835): linguistics and philology
  - *The heterogeneity of language and its influence on the intellectual development of mankind* (1836)
- Franz Boas (1858-1942): anthropology
- Edward Sapir (1884-1939) and Benjamin Whorf (1897-1941): linguistics
  - *Language, mind, and reality* (1942)
- and in sociology, psychology, philosophy, history (e.g., Kuhn's "Structure of scientific revolutions", Wittgenstein's language games), ...
  - deep and lasting impact across the sciences

What we talk about when we talk ... about graphs

Over the past few years, we have been investigating the ways in which graph query languages affect the way in which clients structure their world.

- i.e., how the choice of query language restricts and shapes concrete graph instances.

What we talk about when we talk ... about graphs

Over the past few years, we have been investigating the ways in which graph query languages affect the way in which clients structure their world.

- i.e., how the choice of query language restricts and shapes concrete graph instances.

This talk surveys this work, which is the result of collaborations with my wonderful colleagues at Delft University of Technology, Eindhoven University of Technology, Hasselt University, Indiana University - Bloomington, and Université Libre de Bruxelles.

Full bibliographic details can be found on the last slide and in the abstract accompanying the talk.

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- “query” expressivity
- “instance” expressivity

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- "query" expressivity
- "instance" expressivity

part 2. case studies in instance expressivity

- simple graph languages
- structural indexing for efficient SPARQL query processing

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- "query" expressivity
- "instance" expressivity

part 2. case studies in instance expressivity

- simple graph languages
- structural indexing for efficient SPARQL query processing

part 3. research directions

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- **“query” expressivity**
- **“instance” expressivity**

part 2. case studies in instance expressivity

- simple graph languages
- structural indexing for efficient SPARQL query processing

part 3. research directions

Notions of language expressivity

Codd (1972)

- How can we measure the expressive power of a database query language?

Notions of language expressivity

Codd (1972)

- How can we measure the expressive power of a database query language?
- *Codd's solution:* introduce notion of "relational completeness"
  - is your language as expressive as mine (i.e., the relational calculus)?

Notions of language expressivity

Codd (1972)

- How can we measure the expressive power of a database query language?
- *Codd's solution:* introduce notion of "relational completeness"
  - is your language as expressive as mine (i.e., the relational calculus)?
- ... rather ad hoc

Notions of language expressivity

Towards language-independent notions of expressivity ....

Notions of language expressivity

Towards language-independent notions of expressivity ....

Query expressivity (Aho & Ullman 1979, Chandra & Harel 1980)

- What is the expressive power of Codd's relational calculus/algebra (to formulate general functions)?

Notions of language expressivity

Towards language-independent notions of expressivity ....

Query expressivity (Aho & Ullman 1979, Chandra & Harel 1980)

- What is the expressive power of Codd's relational calculus/algebra (to formulate general functions)?
- for example,
  - *expressible:* nonmonotonic queries
  - *not expressible:* transitive closure

Notions of language expressivity

Towards language-independent notions of expressivity ....

Query expressivity (Aho & Ullman 1979, Chandra & Harel 1980)

- What is the expressive power of Codd's relational calculus/algebra (to formulate general functions)?
- for example,
  - *expressible:* nonmonotonic queries
  - *not expressible:* transitive closure

... primary focus of research community

Notions of language expressivity

Towards language-independent notions of expressivity ....

Instance expressivity (Bancilhon and Paredaens 1978)

- What is the expressive power of Codd's relational algebra (on an arbitrary fixed instance)?

Notions of language expressivity

Towards language-independent notions of expressivity ....

Instance expressivity (Bancilhon and Paredaens 1978)

- What is the expressive power of Codd's relational algebra (on an arbitrary fixed instance)?
- fact: T is expressible from S in Codd's algebra if and only if

$$atoms(T) \subseteq atoms(S)$$

and

$$automorphism(S) \subseteq automorphism(T).$$

Notions of language expressivity

Towards language-independent notions of expressivity ....

Instance expressivity (Bancilhon and Paredaens 1978)

- What is the expressive power of Codd's relational algebra (on an arbitrary fixed instance)?
- fact: T is expressible from S in Codd's algebra if and only if

$$atoms(T) \subseteq atoms(S)$$

and

$$automorphism(S) \subseteq automorphism(T).$$

i.e., characterization in terms of the structure of S.

Instance expressivity

On an (arbitrary) fixed instance S, characterize output space of a given language $\mathcal{L}$

Instance expressivity

On an (arbitrary) fixed instance S, characterize output space of a given language $\mathcal{L}$

Given a source instance S and target instance T, can S be mapped to T in $\mathcal{L}$?

$$S \xrightarrow{? \in \mathcal{L}} T$$

Instance expressivity

On an (arbitrary) fixed instance S, characterize output space of a given language $\mathcal{L}$

Given a source instance S and target instance T, can S be mapped to T in $\mathcal{L}$?

$$S \xrightarrow{? \in \mathcal{L}} T$$

For two objects $o_1, o_2 \in S$, can they be distinguished by an expression $e \in \mathcal{L}$?

$$o_1 \in e(S) \qquad o_2 \notin e(S)$$

Instance expressivity

The BP result is for the relational calculus on relational databases. Similar structural characterizations later discovered for query languages on nested relations and object-oriented DBs.

However, no significant application was made of these results towards engineering of data management systems.

Instance expressivity

Recent results (including applications!)

Instance expressivity

Recent results (including applications!)

- **tree structured data**
  - structural characterizations of XPath fragments (Gyssens et al. PODS 2006)
  - structural indexing for XPath evaluation (Fletcher et al. *Information Systems* 2009, ...)

Instance expressivity

Recent results (including applications!)

- **tree structured data**
  - structural characterizations of XPath fragments (Gyssens et al. PODS 2006)
  - structural indexing for XPath evaluation (Fletcher et al. *Information Systems* 2009, ...)
- **(arbitrary) graph structured data**
  - structural characterizations of Tarski's relation algebra on directed edge-labeled graphs (Fletcher et al. ICDT 2011; arXiv 2012; FoIKS 2012)
  - structural characterizations of SPARQL fragments (Fletcher et al. DBPL 2011)
  - structural indexing for accelerated SPARQL evaluation (Picalausa et al. ESWC 2012)

Instance expressivity

Recent results (including applications!)

- **tree structured data**
  - structural characterizations of XPath fragments (Gyssens et al. PODS 2006)
  - structural indexing for XPath evaluation (Fletcher et al. *Information Systems* 2009, ...)
- **(arbitrary) graph structured data**
  - structural characterizations of Tarski's relation algebra on directed edge-labeled graphs (Fletcher et al. ICDT 2011; arXiv 2012; FoIKS 2012)
  - structural characterizations of SPARQL fragments (Fletcher et al. DBPL 2011)
  - structural indexing for accelerated SPARQL evaluation (Picalausa et al. ESWC 2012)

Also, this week in Genoa, see Antonopoulos et al. (ICDT) and ten Cate et al. (EDBT tutorial), and references therein!

Instance expressivity

Recent results (including applications!)

- **tree structured data**
  - structural characterizations of XPath fragments (Gyssens et al. PODS 2006)
  - structural indexing for XPath evaluation (Fletcher et al. *Information Systems* 2009, ...)
- **(arbitrary) graph structured data** My focus in this talk
  - structural characterizations of Tarski's relation algebra on directed edge-labeled graphs (Fletcher et al. ICDT 2011; arXiv 2012; FoIKS 2012)
  - structural characterizations of SPARQL fragments (Fletcher et al. DBPL 2011)
  - structural indexing for accelerated SPARQL evaluation (Picalausa et al. ESWC 2012)

Also, this week in Genoa, see Antonopoulos et al. (ICDT) and ten Cate et al. (EDBT tutorial), and references therein!

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- "query" expressivity
- "instance" expressivity

part 2. case studies in instance expressivity

- **simple graph languages**
- structural indexing for efficient SPARQL query processing

part 3. research directions

web, linked data, dataspaces, social networks, biological networks, ...

graph structured data

Paths in graphs

Relation Algebra already proposed by Alfred Tarski in the 1940s as a basic query language for reasoning about paths in graphs

Paths in graphs

- clear understanding of expressive power of path navigation is essential
- we study Tarski's relation algebra, on arbitrary graphs (as binary relations)
  - query expressiveness
  - instance expressiveness

Graphs

We are interested in navigating over graphs whose edges are labeled by symbols from a finite, nonempty set of labels Λ.

A graph is a relational structure G, consisting of

- a set of nodes V and,
- for every $R \in \Lambda$, a relation $G(R) \subseteq V \times V$, the set of edges with label R.

Graphs

For example, suppose we have

$$V = \mathit{people} \cup \mathit{hospitals} \cup \mathit{diseases}$$

and edge labels

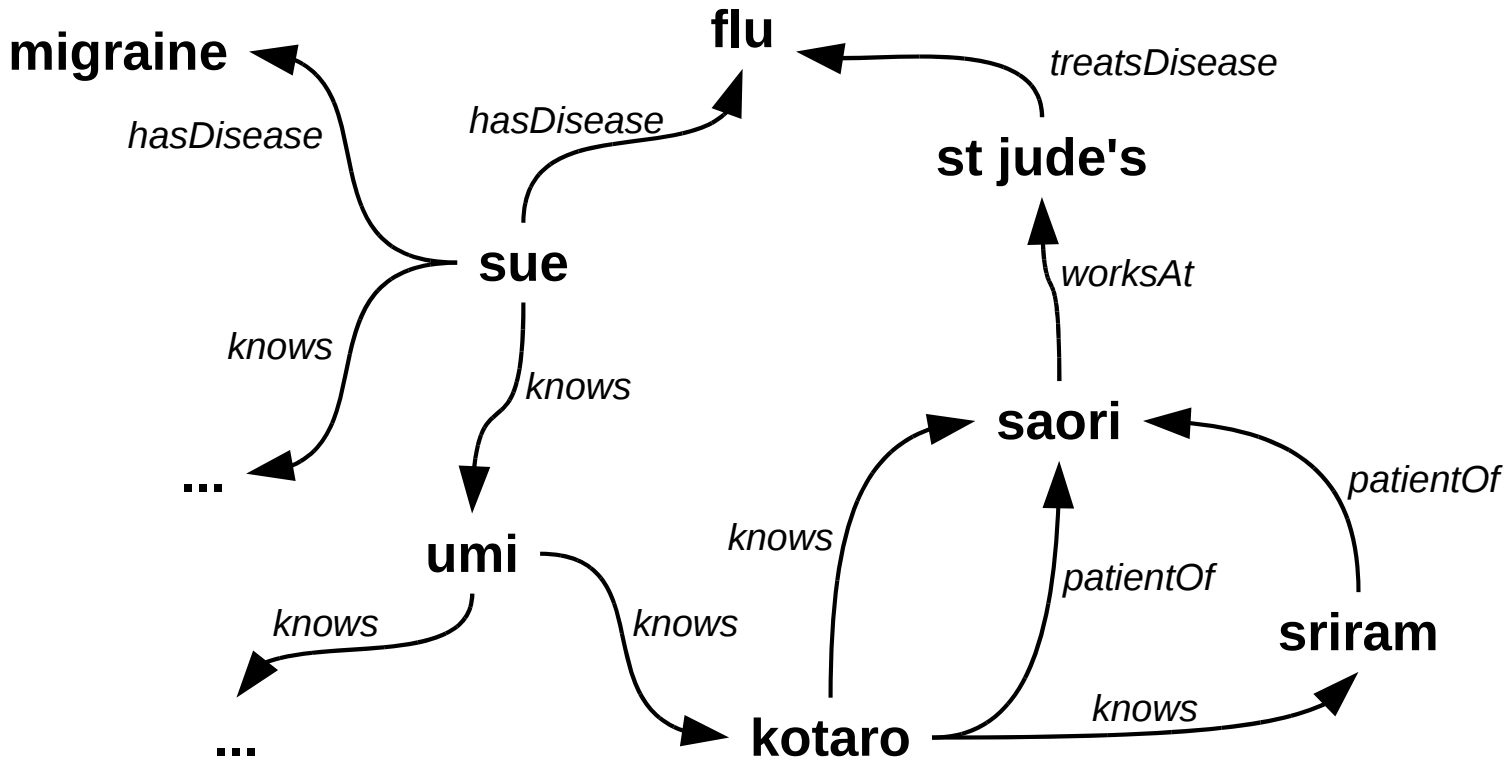
$$\Lambda = \{\mathsf{knows}, \mathsf{worksAt}, \mathsf{patientOf}, \mathsf{hasDisease}, \mathsf{treatsDisease}\}$$

with semantics restricted as:

$$\begin{aligned}
\mathsf{knows} &\subseteq \mathit{people} \times \mathit{people} \\
\mathsf{worksAt} &\subseteq \mathit{people} \times \mathit{hospitals} \\
\mathsf{patientOf} &\subseteq \mathit{people} \times \mathit{people} \\
\mathsf{hasDisease} &\subseteq \mathit{people} \times \mathit{diseases} \\
\mathsf{treatsDisease} &\subseteq \mathit{hospitals} \times \mathit{diseases}.
\end{aligned}$$

Graphs

A small fragment of such a graph



Basic language features

Basic navigational language: algebra $\mathcal{N}$ whose expressions are built recursively from

- the edge labels Λ,
- the primitive $\emptyset$, and
- the primitive id,

using

- composition $(e_1 \circ e_2)$, and
- union $(e_1 \cup e_2)$.

Basic language features

Basic navigational language: algebra $\mathcal{N}$ whose expressions are built recursively from

- the edge labels Λ,
- the primitive $\emptyset$, and
- the primitive id,

using

- composition $(e_1 \circ e_2)$, and
- union $(e_1 \cup e_2)$.

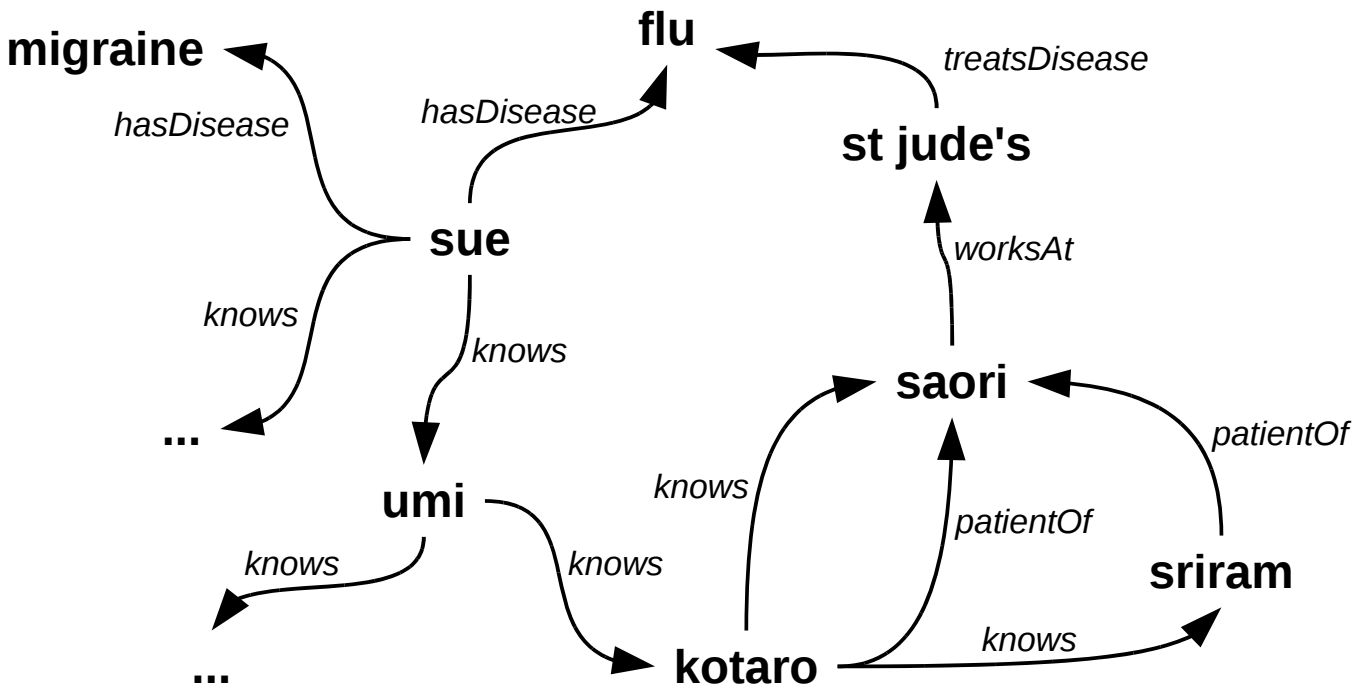
On input graph G, each expression $e \in \mathcal{N}$ defines a path query $e(G) \subseteq \mathrm{adom}(G) \times \mathrm{adom}(G)$, i.e., a binary relation on the *active domain* of G.

Basic language features

In particular, the semantics of $\mathcal{N}$ is inductively defined as follows:

$$
\begin{aligned}
R(G) &= G(R)\,; \\
\emptyset(G) &= \emptyset\,; \\
id(G) &= \{(m,m) \mid m \in \mathrm{adom}(G)\}\,; \\
e_1 \circ e_2(G) &= \{(m,n) \mid \exists p\,((m,p) \in e_1(G)\ \&\ (p,n) \in e_2(G))\}\,; \\
e_1 \cup e_2(G) &= e_1(G) \cup e_2(G)\,.
\end{aligned}
$$

Basic language features



Example: by person, the doctors of their friends

$$\text{knows} \circ \text{patientOf}(G) = \{(\mathit{umi}, \mathit{saori}), (\mathit{kotaro}, \mathit{saori}), \ldots\}$$

Nonbasic language features

The basic algebra $\mathcal{N}$ is extended with the following features:

- diversity (di),
- converse (e^{-1}),
- intersection ($e_1 \cap e_2$),
- difference ($e_1 \setminus e_2$),
- projections ($\pi_1(e)$ and $\pi_2(e)$), and,
- coprojections ($\overline{\pi}_1(e)$ and $\overline{\pi}_2(e)$).

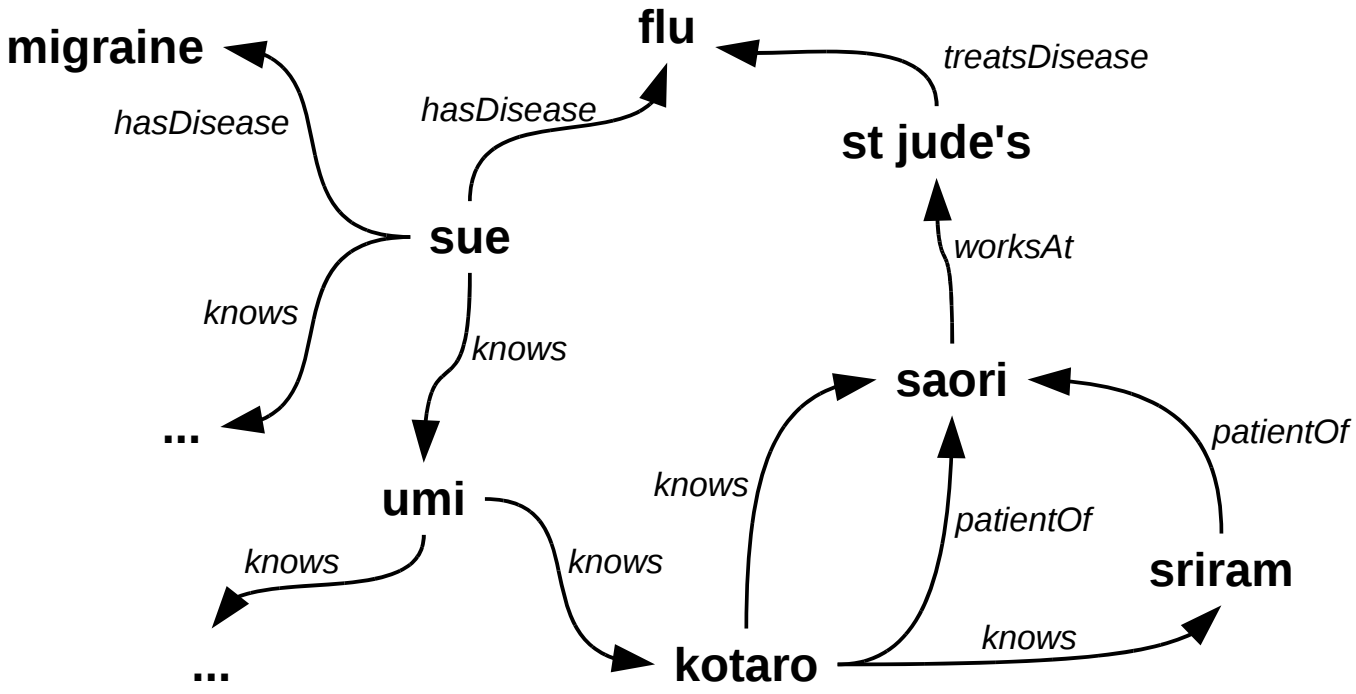
Tarski's algebra consists of the language having all basic and nonbasic features.

Nonbasic language features

The semantics of these language extensions is as follows:

$$
\begin{aligned}
di(G) &= \{(m,n) \mid m,n \in \mathsf{adom}(G) \;\&\; m \neq n\}\,;\\
e^{-1}(G) &= \{(m,n) \mid (n,m) \in e(G)\}\,;\\
e_1 \cap e_2(G) &= e_1(G) \cap e_2(G)\,;\\
e_1 \setminus e_2(G) &= e_1(G) \setminus e_2(G)\,;\\
\pi_1(e)(G) &= \{(m,m) \mid m \in \mathsf{adom}(G) \;\&\; \exists n\,(m,n) \in e(G)\}\,;\\
\pi_2(e)(G) &= \{(m,m) \mid m \in \mathsf{adom}(G) \;\&\; \exists n\,(n,m) \in e(G)\}\,;\\
\overline{\pi}_1(e)(G) &= \{(m,m) \mid m \in \mathsf{adom}(G) \;\&\; \neg\exists n\,(m,n) \in e(G)\}\,;\\
\overline{\pi}_2(e)(G) &= \{(m,m) \mid m \in \mathsf{adom}(G) \;\&\; \neg\exists n\,(n,m) \in e(G)\}\,.
\end{aligned}
$$

Nonbasic language features



Example: people with untreatable diseases

$$\text{hasDisease} \setminus (\text{hasDisease} \circ \pi_2(\text{treatsDisease}))(G) =$$
$$\{(\textit{sue}, \textit{migraine}), \ldots\}$$

Language equivalence

A marked structure $\mathbf{G}$ is a triple (G, a, b) where G is a graph, and (a, b) is an ordered pair of nodes from G.

For two marked structures $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$, we write $\mathbf{G}_1 \equiv \mathbf{G}_2$ if $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable in the RA, i.e., for every expression e in the algebra, whenever $(a_1, b_1) \in e(G_1)$, it also holds that $(a_2, b_2) \in e(G_2)$, and vice versa.

Structural equivalence

Let G_1 and G_2 be two graphs with node sets V_1 and V_2, respectively. A non-empty relation $Z \subseteq V_1^2 \times V_2^2$ is a bisimulation between G_1 and G_2 if it satisfies the following conditions

Structural equivalence

Let G_1 and G_2 be two graphs with node sets V_1 and V_2, respectively. A non-empty relation $Z \subseteq V_1^2 \times V_2^2$ is a bisimulation between G_1 and G_2 if it satisfies the following conditions

Atoms if (a_1, b_1, a_2, b_2) is in Z, then $(a_1, b_1) \in R(G_1)$ if and only if $(a_2, b_2) \in R(G_2)$, for all $R \in \Lambda$;

Structural equivalence

Let G_1 and G_2 be two graphs with node sets V_1 and V_2, respectively. A non-empty relation $Z \subseteq V_1^2 \times V_2^2$ is a bisimulation between G_1 and G_2 if it satisfies the following conditions

Atoms if (a_1, b_1, a_2, b_2) is in Z, then $(a_1, b_1) \in R(G_1)$ if and only if $(a_2, b_2) \in R(G_2)$, for all $R \in \Lambda$;

Forth if $(a_1, b_1, a_2, b_2) \in Z$, then

- for each $c_1 \in V_1$ there exist $c_2 \in V_2$ such that both (a_1, c_1, a_2, c_2) and (c_1, b_1, c_2, b_2) are in Z;
- if $a_1 = b_1$ then $a_2 = b_2$; and,
- $(b_1, a_1, b_2, a_2) \in Z$.

Structural equivalence

Let G_1 and G_2 be two graphs with node sets V_1 and V_2, respectively. A non-empty relation $Z \subseteq V_1^2 \times V_2^2$ is a bisimulation between G_1 and G_2 if it satisfies the following conditions

Atoms if (a_1, b_1, a_2, b_2) is in Z, then $(a_1, b_1) \in R(G_1)$ if and only if $(a_2, b_2) \in R(G_2)$, for all $R \in \Lambda$;

Forth if $(a_1, b_1, a_2, b_2) \in Z$, then

- for each $c_1 \in V_1$ there exist $c_2 \in V_2$ such that both (a_1, c_1, a_2, c_2) and (c_1, b_1, c_2, b_2) are in Z;
- if $a_1 = b_1$ then $a_2 = b_2$; and,
- $(b_1, a_1, b_2, a_2) \in Z$.

Back is the same as *Forth*, only with the roles of G_1 and G_2 reversed.

Structural equivalence

A marked structure $\mathbf{G}_1 = (G_1, a_1, b_1)$ is said to be bisimilar to a marked structure $\mathbf{G}_2 = (G_2, a_2, b_2)$ if there is a bisimulation Z between G_1 and G_2 containing (a_1, b_1, a_2, b_2).

Structural equivalence

A marked structure $\mathbf{G}_1 = (G_1, a_1, b_1)$ is said to be bisimilar to a marked structure $\mathbf{G}_2 = (G_2, a_2, b_2)$ if there is a bisimulation Z between G_1 and G_2 containing (a_1, b_1, a_2, b_2).

Coupling Theorem

Let $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$ be finite marked structures. Then

$$\mathbf{G}_1 \equiv \mathbf{G}_2 \quad \Leftrightarrow \quad \mathbf{G}_1 \text{ is bisimilar to } \mathbf{G}_2.$$

Structural equivalence

A marked structure $\mathbf{G}_1 = (G_1, a_1, b_1)$ is said to be bisimilar to a marked structure $\mathbf{G}_2 = (G_2, a_2, b_2)$ if there is a bisimulation Z between G_1 and G_2 containing (a_1, b_1, a_2, b_2).

Coupling Theorem

Let $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$ be finite marked structures. Then

$$\mathbf{G}_1 \equiv \mathbf{G}_2 \quad \Leftrightarrow \quad \mathbf{G}_1 \text{ is bisimilar to } \mathbf{G}_2.$$

We similarly obtained novel bisimulation characterizations for a wide range of fragments of the algebra.

Structural equivalence

A marked structure $\mathbf{G}_1 = (G_1, a_1, b_1)$ is said to be bisimilar to a marked structure $\mathbf{G}_2 = (G_2, a_2, b_2)$ if there is a bisimulation Z between G_1 and G_2 containing (a_1, b_1, a_2, b_2).

Coupling Theorem

Let $\mathbf{G}_1 = (G_1, a_1, b_1)$ and $\mathbf{G}_2 = (G_2, a_2, b_2)$ be finite marked structures. Then

$$\mathbf{G}_1 \equiv \mathbf{G}_2 \quad \Leftrightarrow \quad \mathbf{G}_1 \text{ is bisimilar to } \mathbf{G}_2.$$

We similarly obtained novel bisimulation characterizations for a wide range of fragments of the algebra.

For positive algebra fragments, we similarly obtained new simulation characterizations, where the *Back* condition is dropped.

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- "query" expressivity
- "instance" expressivity

part 2. case studies in instance expressivity

- simple graph languages
- **structural indexing for efficient SPARQL query processing**

part 3. research directions

Structural indexing

Up to this point, our investigations of Tarski's algebra have focused on the relative expressive power of the various fragments of the algebra.

We have also obtained structural characterizations for a core fragment of SPARQL, the W3C's recommendation language for the RDF graph data model, with an eye towards "structural" index design.

Structural indexing

Up to this point, our investigations of Tarski's algebra have focused on the relative expressive power of the various fragments of the algebra.

We have also obtained structural characterizations for a core fragment of SPARQL, the W3C's recommendation language for the RDF graph data model, with an eye towards "structural" index design.

The basic idea here is to group together structurally equivalent RDF triples, since the language cannot distinguish them, and build access mechanisms on top of these "blocks."

We then use this index to accelerate query processing on a reduced search space (Picalausa et al. ESWC 2012).

Partitioning massive graphs under bisimulation

Before we proceed, we note that this approach only works if computing bisimulation partitioning of a graph is practical.

Partitioning massive graphs under bisimulation

Before we proceed, we note that this approach only works if computing bisimulation partitioning of a graph is practical.

Efficient *main memory* approaches to bisimulation partitioning have been studied since the 80's, as bisimilarity is a fundamental notion arising in a wide range of contexts (e.g., set theory, distributed computing, process modeling, ...).

However, there has been no approach to compute bisimulation on massive disk-resident graphs.

Partitioning massive graphs under bisimulation

To address this, we have developed the first I/O-efficient approaches to bisimulation partitioning of massive graphs (Hellings et al. SIGMOD 2012; Luo et al. arXiv 2012).

We have also developed the first effective MapReduce solution for this problem (Luo et al. BNCOD 2013).

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

RDF and SPARQL Query Processing

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people who are indirectly related.

Sue	Larry
Joe	Sarah
Sue	Hiromi
John	Sarah

Adding join information

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

We mark all triples (s_1, p_1, o) such that their **object** o occurs as the **subject** of some other triple (o, p_2, o_2)

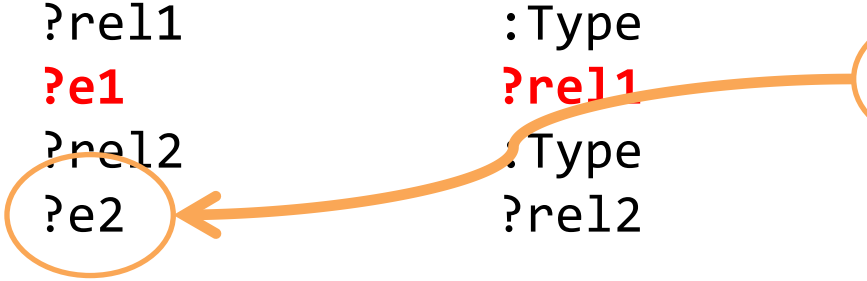
Using join information

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Find the people indirectly related.

```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```



Motivation

- Traditional relational SPARQL query engines fetch triples corresponding to individual triple patterns **independently**

- Rich history of introducing "structural" join information into query engines
  - Join Indexes: Precompute a single join (e.g. R.a = S.b)
  - Object Oriented Indexes: Precompute join of single path in class hierarchy
  - Structural Indexes (for XML and RDF): Group **nodes** according to similarity of edge labels (fixed set)

- By grouping together **triples** that can be joined in a "similar fashion", we can avoid fetching useless triples from disk.

Real-World SPARQL Queries

Definition: The **equality type** of two triples $t = (t_1, t_2, t_3)$ and $u = (u_1, u_2, u_3)$ is the set $eqtp(t,u) = \{ (i,j) \mid t_i = u_j, \text{ and } 1 \leq i, j \leq 3\}$ of positions where the triples share an equal value.

Real-World SPARQL Queries

Definition: The **equality type** of two triples t = (t_1, t_2, t_3) and u = (u_1, u_2, u_3) is the set eqtp*(t,u) = { (i,j) | $t_i = u_j$,* and *1 ≤ i, j ≤ 3}* of positions where the triples share an equal value.

t: Sue Manages Joe

u: Joe Manages Larry

eqtp(t,u) = {(3,1), (2,2)}

Structural Index

Definition: A **structural index** is an edge labeled graph (V,E), where
The nodes V form a partition of the RDF dataset
The edges E are labeled by the equality types between triples in nodes

Subject	Predicate	Object
Sue	Manages	Joe
Joe	Manages	Larry
Larry	Manages	Sarah
Sue	FriendOf	John
John	FriendOf	Hiromi
Hiromi	FriendOf	Sarah

Subject	Predicate	Object
Sue	Type	CEO
Manages	Type	socialRelation
FriendOf	Type	socialRelation
Likes	Type	socialRelation

Structural Index

Definition: A **structural index** is an edge labeled graph (V,E), where
The nodes V form a partition of the RDF dataset
The edges E are labeled by the equality types between triples in nodes

1		
Sue	FriendOf	John
Sue	Manages	Joe

2		
John	FriendOf	Hiromi
Joe	Manages	Larry

3		
Hiromi	FriendOf	Sarah
Larry	Manages	Sarah

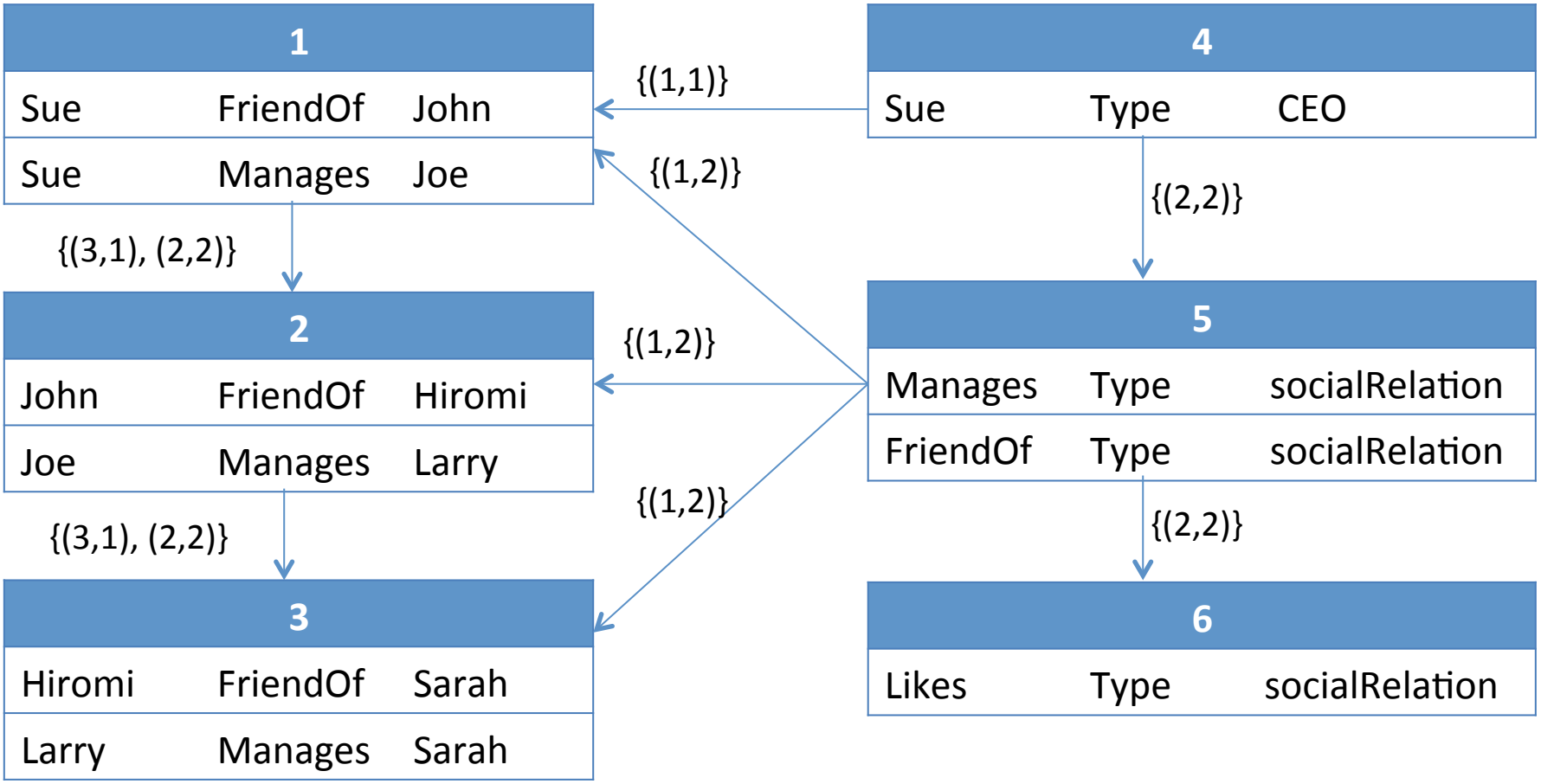
4		
Sue	Type	CEO

5		
Manages	Type	socialRelation
FriendOf	Type	socialRelation

6		
Likes	Type	socialRelation

Structural Index

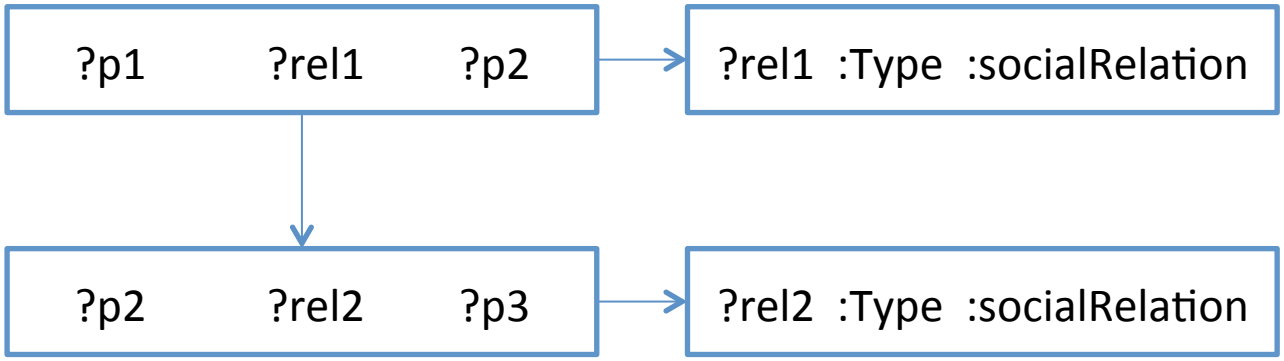
Definition: A **structural index** is an edge labeled graph (V,E), where
The nodes V form a partition of the RDF dataset
The edges E are labeled by the equality types between triples in nodes



Real-World SPARQL Queries

Fact: Most queries posed in practice only use basic graph patterns (BGPs).
99% of real-world BGP queries are found to be **acyclic**.

[Picalausa, Vansummeren – in SWIM2011]



Structural Characterization

Fact: Most queries posed in practice only use basic graph patterns (BGPs). **99%** of real-world BGP queries are found to be **acyclic**.

[Picalausa, Vansummeren – in SWIM2011]

Definition: A BGP query is **pure** if it contains only variables.

Structural Characterization

Fact: Most queries posed in practice only use basic graph patterns (BGPs). **99%** of real-world BGP queries are found to be **acyclic**.

[Picalausa, Vansummeren – in SWIM2011]

Definition: A BGP query is **pure** if it contains only variables.

Theorem: Given two triples t, and u, the following are equivalent:

- t is in Q(D) if and only if u is in Q(D), for every *pure acyclic BGP* Q
- t **simulates** u **guardedly** and vice-versa

[Fletcher, Hidders, Vansummeren, Luo, Picalausa, De Bra — DBPL 2011]

Structural Characterization

Theorem: Given two triples t, and u, the following are equivalent:

- t is in Q(D) if and only if u is in Q(D), for every *pure acyclic BGP* Q
- t **simulates** u **guardedly** and vice-versa

[Fletcher, Hidders, Vansummeren, Luo, Picalausa, De Bra — DBPL 2011]

Consider a RDF dataset D. A triple t of D **simulates** a triple u of D **guardedly** if for every triple t' of D, there exists some triple u' of D such that eqtp(t,t') ⊆ eqtp(u,u') and t' simulates u'.

(Sue, FriendOf, John) —{(3,1), (2,2)}→ (John, FriendOf, Hiromi) —{(2,1)}→ (FriendOf, Type, relation)

(Joe, Manages, Larry) —{(3,1), (2,2)}→ (Lary, Manages, Sarah) —{(2,1)}→ (Manages, Type, relation)

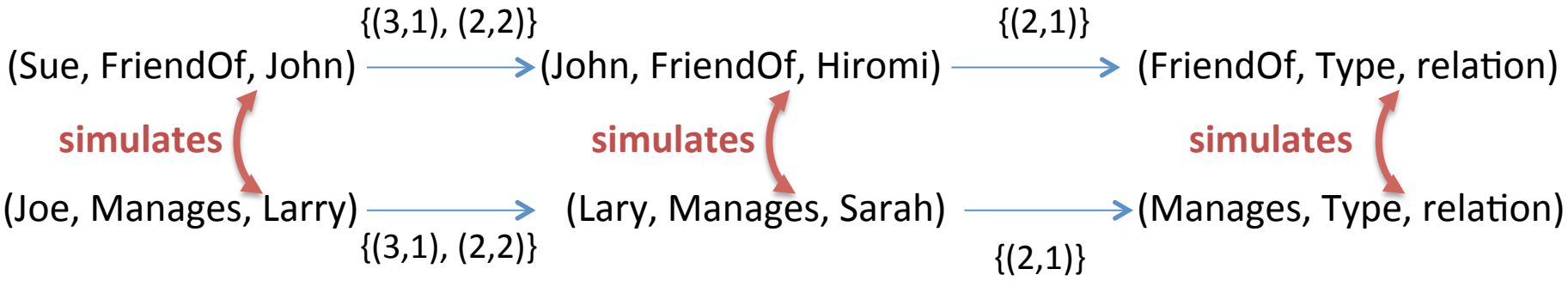
Structural Characterization

Theorem: Given two triples t, and u, the following are equivalent:

- t is in Q(D) if and only if u is in Q(D), for every *pure acyclic BGP* Q
- t **simulates** u **guardedly** and vice-versa

[Fletcher, Hidders, Vansummeren, Luo, Picalausa, De Bra — DBPL 2011]

Consider a RDF dataset D. A triple t of D **simulates** a triple u of D **guardedly** if for every triple t' of D, there exists some triple u' of D such that eqtp(t,t') ⊆ eqtp(u,u') and t' simulates u'.



Structural Characterization

Theorem: Given two triples t, and u, the following are equivalent:

- t is in Q(D) if and only if u is in Q(D), for every *pure acyclic BGP* Q
- t **simulates** u **guardedly** and vice-versa

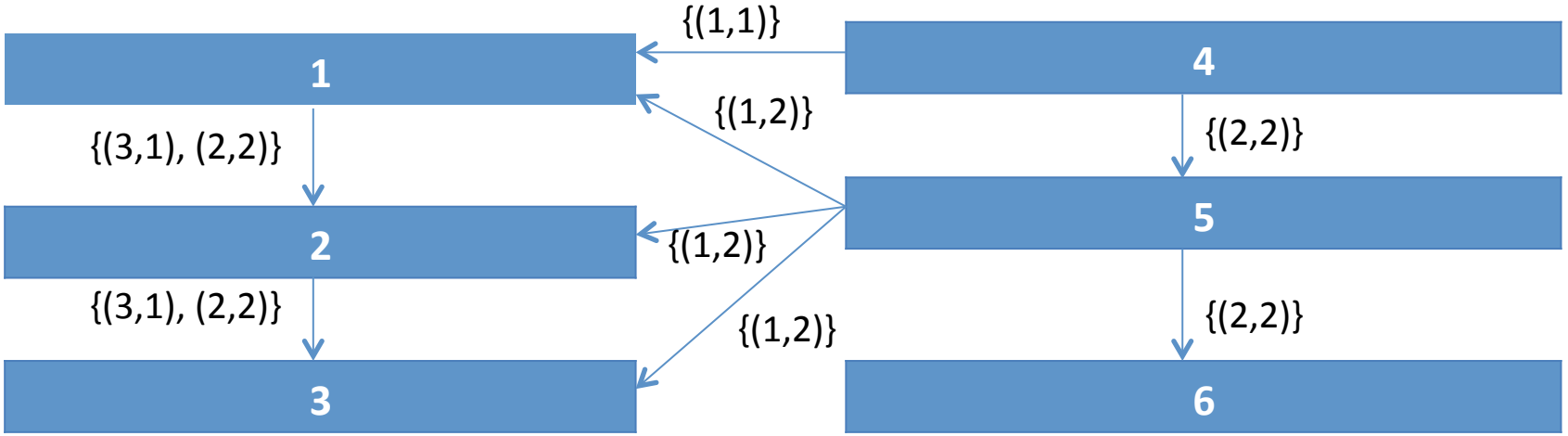
[Fletcher, Hidders, Vansummeren, Luo, Picalausa, De Bra — DBPL 2011]

Consider a RDF dataset D. A triple t of D **simulates** a triple u of D **guardedly** if for every triple t' of D, there exists some triple u' of D such that eqtp(t,t') $\subseteq$ eqtp(u,u') and t' simulates u'.

Idea: partition D under simulation

Structural Index Storage

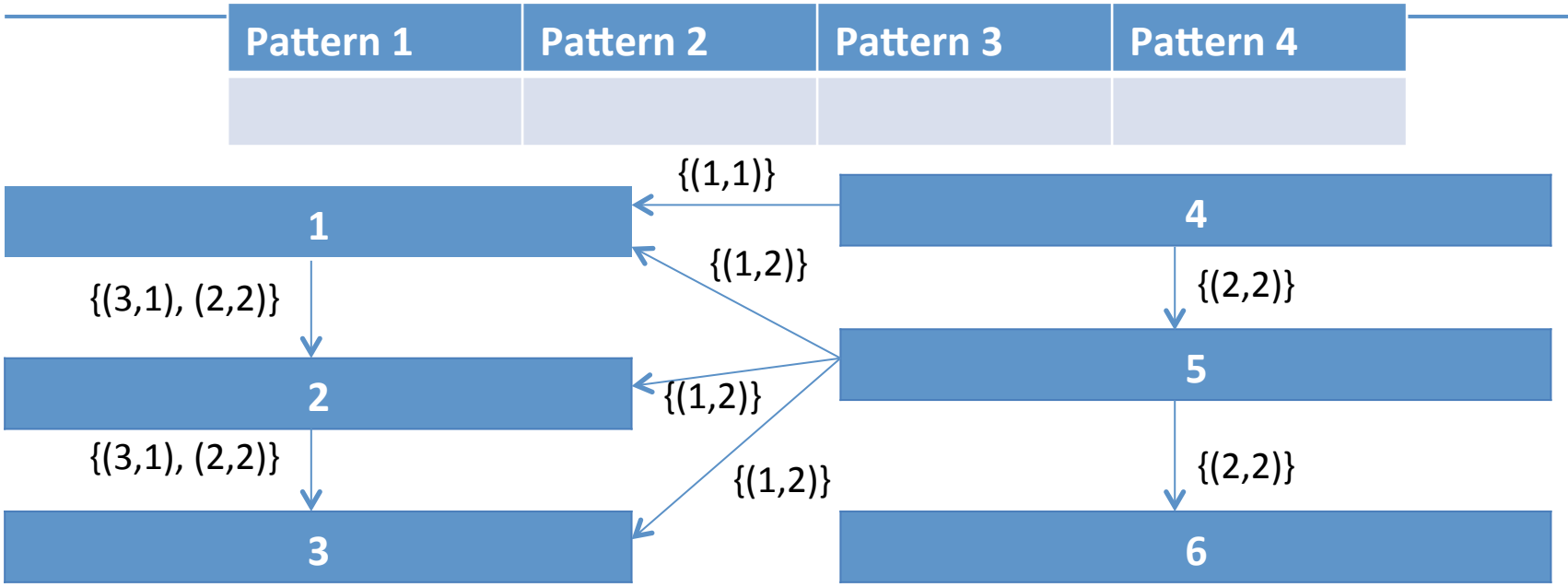
Ideally, the structural index is sufficiently small to be kept in main memory



Each triple (subject, predicate, object) is stored as a quad (subject, predicate, object, partition)

Subject	Predicate	Object	Partition
Sue	Manages	Joe	1
Joe	Manages	Larry	2
Larry	Manages	Sarah	3
Sue	FriendOf	John	1

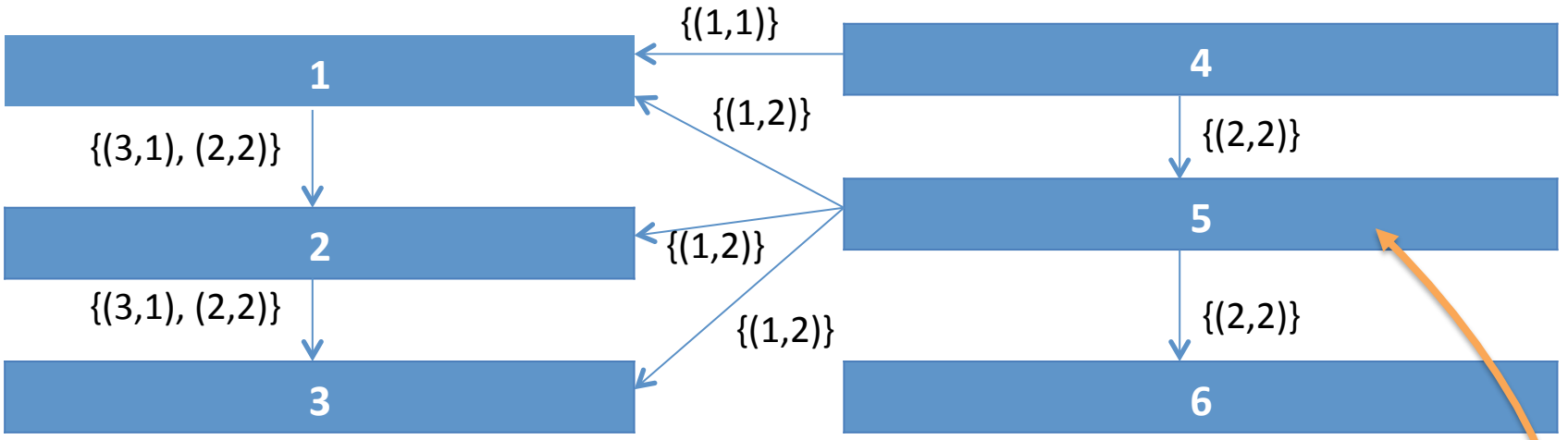
Querying the Index




```
SELECT ?e1 ?e3 WHERE {
    ?rel1           :Type          :socialRelation .
    ?e1             ?rel1          ?e2 .
    ?rel2           :Type          :socialRelation .
    ?e2             ?rel2          ?e3 .
}
```

Querying the Index

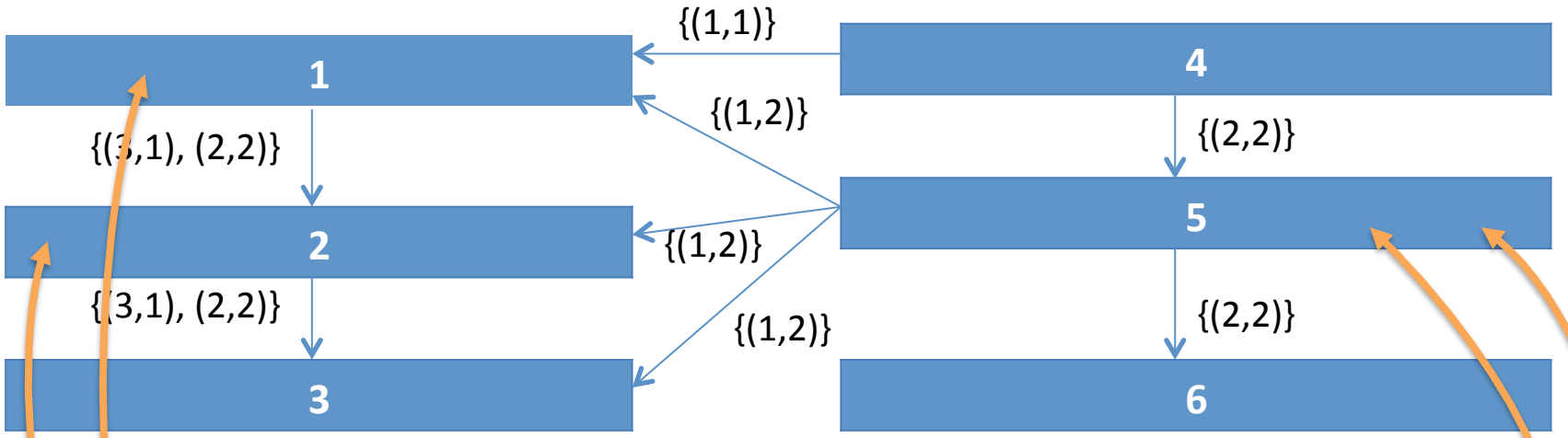
Pattern 1	Pattern 2	Pattern 3	Pattern 4
5			




```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

Querying the Index

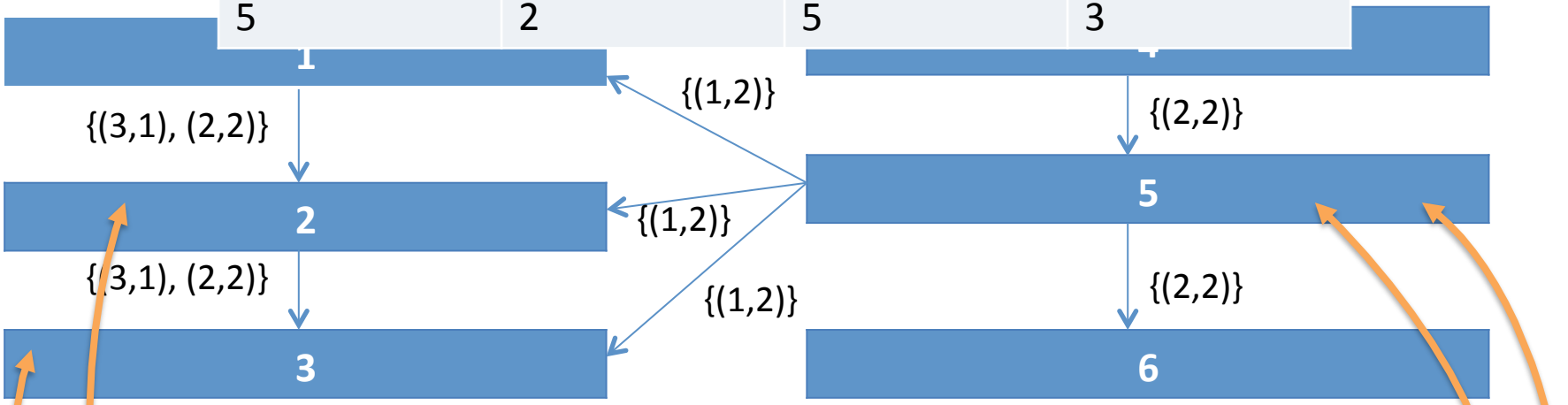
Pattern 1	Pattern 2	Pattern 3	Pattern 4
5	1	5	2




```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

Querying the Index

Pattern 1	Pattern 2	Pattern 3	Pattern 4
5	1	5	2
5	2	5	3




```
SELECT ?e1 ?e3 WHERE {
    ?rel1          :Type          :socialRelation .
    ?e1            ?rel1          ?e2 .
    ?rel2          :Type          :socialRelation .
    ?e2            ?rel2          ?e3 .
}
```

Query Processing Strategies

Input: The SPARQL query

All embeddings of the query into the structural index

Pattern 1	Pattern 2	Pattern 3	Pattern 4
5	1	5	2
5	2	5	3

Output: A physical query plan

Empirical study

SaintDB: Implement quad-store atop RDF-3x open-source state-of-the-art value-based triple store.

Empirical analysis on community benchmark data/queries demonstrates competitiveness with RDF-3X on broad range of query scenarios, with up to multiple orders of magnitude reduction in query processing costs.

Structural indexing for SPARQL

SaintDB: quad-store based structural indexing and query processing

- We introduced the first triple-based structural index for RDF.
- This index is formally coupled to practical core fragment of SPARQL.
- Our initial empirical study shows that the approach is profitable

What we talk about when we talk about graphs

part 1. a brief history of query language expressivity

- "query" expressivity
- "instance" expressivity

part 2. case studies in instance expressivity

- simple graph languages
- structural indexing for efficient SPARQL query processing

part 3. research directions

Open research directions

- study expressiveness of practical extensions of relation algebra and BGPs
- study other natural classes of graphs (e.g., DAGs, series-parallel graphs)

Open research directions

- study expressiveness of practical extensions of relation algebra and BGPs
- study other natural classes of graphs (e.g., DAGs, series-parallel graphs)
- query processing with structural indexes

Open research directions

- study expressiveness of practical extensions of relation algebra and BGPs
- study other natural classes of graphs (e.g., DAGs, series-parallel graphs)
- query processing with structural indexes
- study other basic applications, e.g.,
  - query language design in social network analysis (cf. Marx and Masuch, *Social Networks* 25(1), 2003; Fan ICDT 2012)
  - structure-sensitive privacy and security mechanisms
  - dynamic structure (e.g., ontology) extraction, via language-distinguishability
  - visualizing language-induced structures (e.g., interplay of ontological knowledge)

Thank you! Questions?

What we talk about when we talk about graphs

George Fletcher

Eindhoven University of Technology
The Netherlands

GraphQ 2013

References

- G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, and S. Vansummeren. Similarity and bisimilarity notions appropriate for characterizing indistinguishability in fragments of the calculus of relations. CoRR, abs/1210.2688, 2012.
- G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. In ICDT, pages 197-207, Uppsala, Sweden, 2011.
- G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. The impact of transitive closure on the boolean expressiveness of navigational query languages on graphs. In FoIKS, pages 124-143, Kiel, Germany, 2012.
- G. H. L. Fletcher, M. Gyssens, J. Paredaens, and D. Van Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. IEEE Trans. Knowl. Data Eng., 21(6):939 - 942, 2009.
- G. H. L. Fletcher, J. Hidders, S. Vansummeren, Y. Luo, F. Picalausa, and P. De Bra. On guarded simulations and acyclic first-order languages. In DBPL, Seattle, 2011.
- G. H. L. Fletcher, J. Van Den Bussche, D. Van Gucht, and S. Vansummeren. Towards a theory of search queries. ACM TODS, 35:28:1-28:33, 2010.
- G. H. L. Fletcher, D. Van Gucht, Y. Wu, M. Gyssens, S. Brenes, and J. Paredaens. A methodology for coupling fragments of XPath with structural indexes for XML documents. Inf. Syst., 34(7):657-670, 2009.
- M. Gyssens, J. Paredaens, D. Van Gucht, and G. H. L. Fletcher. Structural characterizations of the semantics of XPath as navigation tool on a document. In PODS, pages 318-327, Chicago, IL, USA, 2006.
- J. Hellings, G. H. L. Fletcher, and H. Haverkort. Efficient external-memory bisimulation on DAGs. In SIGMOD, pages 553-564, Scottsdale, AZ, USA, 2012.
- Y. Luo, Y. de Lange, G. H. L. Fletcher, P. De Bra, J. Hidders, and Y. Wu. Bisimulation reduction of big graphs on MapReduce. To appear in BNCOD, Oxford, UK, 2013.
- Y. Luo, G. H. L. Fletcher, J. Hidders, Y. Wu, and P. De Bra. I/O-efficient algorithms for localized bisimulation partition construction and maintenance on massive graphs. CoRR, abs/1210.0748, 2012.
- F. Picalausa, Y. Luo, G. H. L. Fletcher, J. Hidders, and S. Vansummeren. A structural approach to indexing triples. In ESWC, pages 406-421, Heraklion, Greece, 2012.