On Tarski's Relation Algebra

QUERYING TREES AND CHAINS

and

THE SEMI-JOIN ALGEBRA

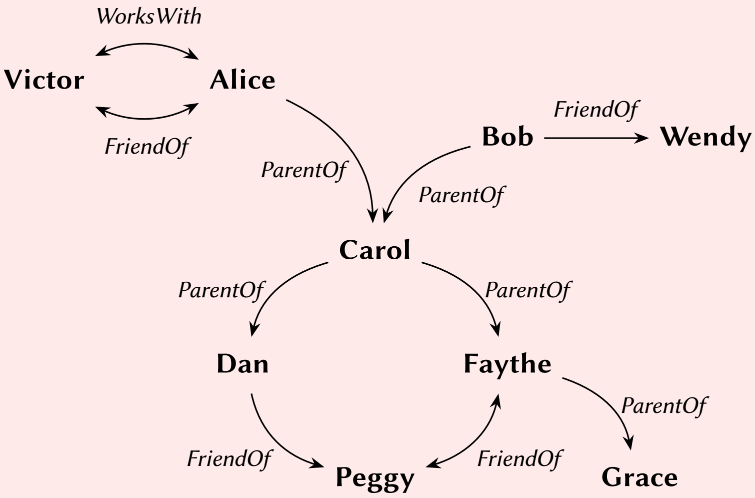
Jelle Hellings

Part I

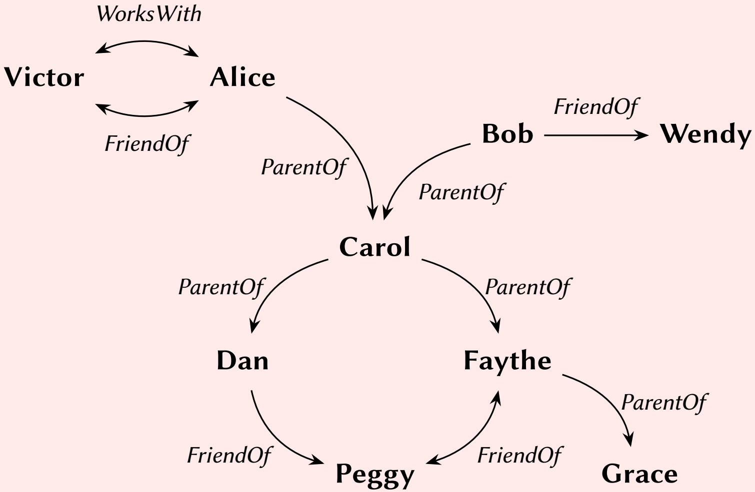
On Tarski's Relation Algebra

INTRODUCTION

Relation algebra: graphs and queries

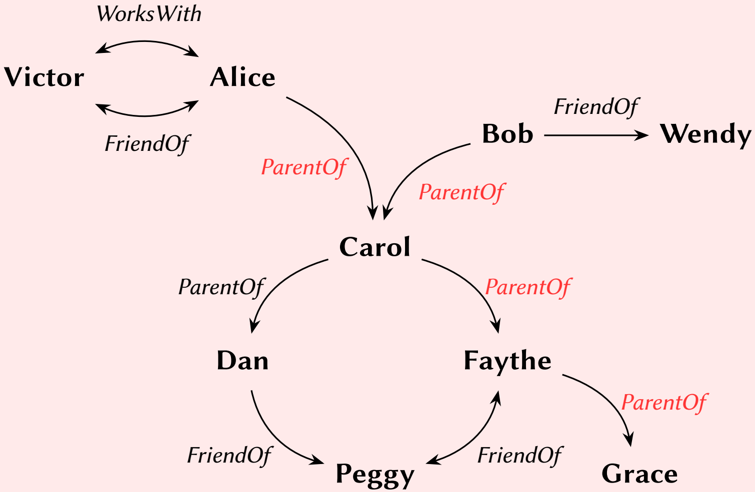


Relation algebra: graphs and queries



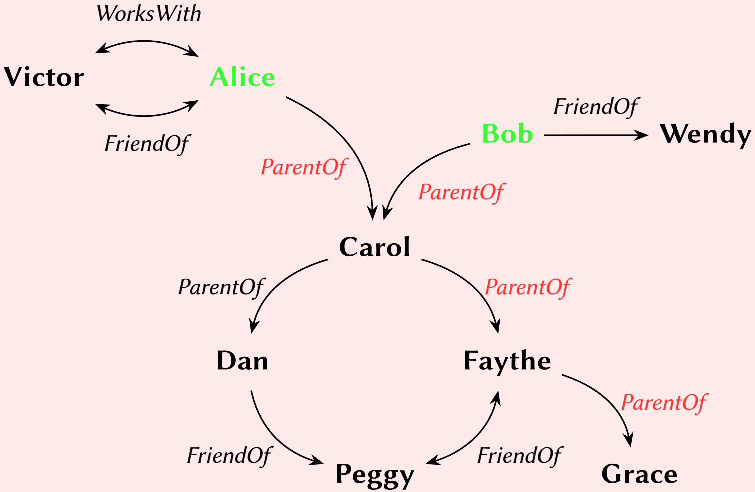
$$FriendsOfGgp = \pi_1[ParentOf \circ ParentOf \circ ParentOf] \circ FriendOf$$

Relation algebra: graphs and queries



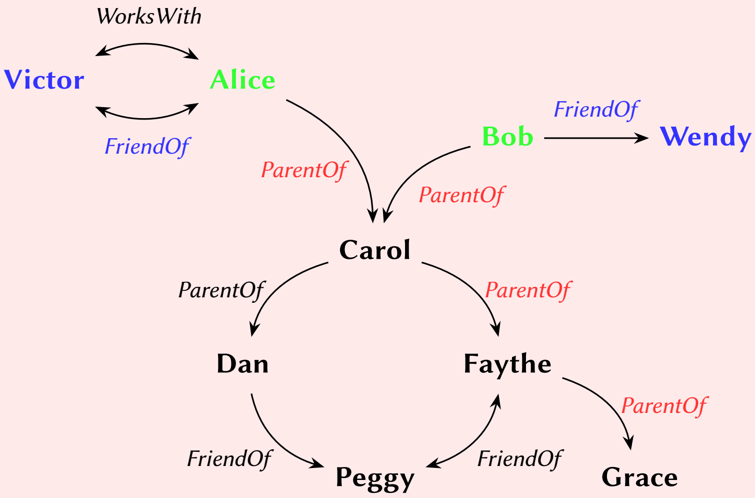
$$\mathit{FriendsOfGgp} = \pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}] \circ \mathit{FriendOf}$$

Relation algebra: graphs and queries



$$FriendsOfGgp = \pi_1[ParentOf \circ ParentOf \circ ParentOf] \circ FriendOf$$

Relation algebra: graphs and queries



$$FriendsOfGgp = \pi_1[ParentOf \circ ParentOf \circ ParentOf] \circ FriendOf$$

Relation algebra: other operators

$$\begin{aligned}
\mathit{ChildOf} &= \mathit{ParentOf}^{\frown} \\
\mathit{AcquaintanceOf} &= \mathit{FriendOf} \cup \mathit{WorksWith} \\
\mathit{WorkFriendOf} &= \mathit{FriendOf} \cap \mathit{WorksWith} \\
\mathit{NonWorkFriend} &= \mathit{FriendOf} - \mathit{WorksWith} \\
\mathit{Parent} &= \pi_1[\mathit{ParentOf}] \\
\mathit{NonParent} &= \overline{\pi}_1[\mathit{ParentOf}] \\
\mathit{GrandParentOf} &= \mathit{ParentOf} \circ \mathit{ParentOf} \\
\mathit{AncestorOf} &= [\mathit{ParentOf}]^+
\end{aligned}$$

Also: constants $\emptyset$, id, di.

Graph query languages

Tarski's Relation Algebra, FO[3]

$*$	$\emptyset$	id	$\cup$	$\circ$	$\frown$	π	$\overline{\pi}$	$\cap$	$-$	di

RPQs

2RPQs

Nested RPQs

Navigational XPath, GXPath

Questions about the relation algebra

Questions

- Are all these operators necessary?
- What does each operator add?
- When can we replace complex operators by simpler ones?

Questions about the relation algebra

Questions

- Are all these operators necessary?
- What does each operator add?
- When can we replace complex operators by simpler ones?

What is the expressive power of fragments of the relation algebra?

Related work and our results

Previous work by Fletcher et al.

Relative expressive power of relation algebra fragments on graphs.

This work

Improve understanding of the relation algebra

- when querying trees or chains;
- in comparison with the semi-join algebra.

Background: when are queries equivalent?

Two types of queries and equivalences

Path-queries. The exact query result is important:

$$\mathit{FriendOf} \cap \mathit{WorksWith} \equiv_{\text{path}} \mathit{FriendOf} - (\mathit{FriendOf} - \mathit{WorksWith}).$$

Boolean-queries. The existence of a query result is important:

$$\mathit{FriendOf}^{\frown} \circ \mathit{ParentOf} \equiv_{\text{bool}} \pi_1[\mathit{FriendOf}] \cap \pi_1[\mathit{ParentOf}].$$

Definition

Language $\mathcal{L}_1$ is z-subsumed by $\mathcal{L}_2$ if every query in $\mathcal{L}_1$ is z-equivalent to a query in $\mathcal{L}_2$ (denoted by $\mathcal{L}_1 \preceq_z \mathcal{L}_2$).

Background: query fragments

Let $\mathcal{F} \subseteq \{\mathrm{di}, ^{\smallfrown}, \pi, \overline{\pi}, \cap, -, ^*\}$.

- We write $\mathcal{N}(\mathcal{F})$: only allows $\emptyset$, id, ℓ, $\circ$, $\cup$, and all operators in $\mathcal{F}$.
- We write $\mathcal{N}(\underline{\mathcal{F}})$ to represent all operators expressible in $\mathcal{N}(\mathcal{F})$ using the following basic rewrite rules:

$$\begin{aligned}
\pi_1[e] &= \overline{\pi}_j[\overline{\pi}_1[e]] = (e \circ [e]^{-1}) \cap \mathrm{id} = (e \circ (\mathrm{id} \cup \mathrm{di})) \cap \mathrm{id};\\
\pi_2[e] &= \overline{\pi}_j[\overline{\pi}_2[e]] = ([e]^{-1} \circ e) \cap \mathrm{id} = ((\mathrm{id} \cup \mathrm{di}) \circ e) \cap \mathrm{id};\\
\overline{\pi}_i[e] &= \mathrm{id} - \pi_i[e];\\
e_1 \cap e_2 &= e_1 - (e_1 - e_2).
\end{aligned}$$

Examples

- $\mathcal{N}(\underline{\overline{\pi}}) = \mathcal{N}(\pi, \overline{\pi})$.
- $\mathcal{N}(\underline{^{\smallfrown}, -}) = \mathcal{N}(^{\smallfrown}, \pi, \overline{\pi}, \cap, -)$.

Part II

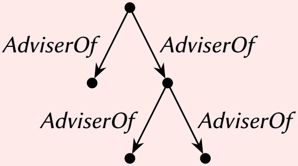
On Tarski's Relation Algebra

QUERYING TREES AND CHAINS

Why studying trees

Definition

A tree is an acyclic graph in which exactly one node, the *root*, has no incoming edges, and all other nodes have exactly one incoming edge.

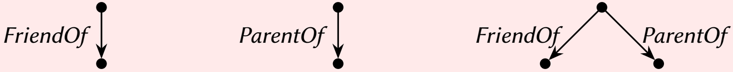


- Hierarchical relations: taxonomies, corporate structures.
- XML and JSON data.
- Nested relational data.

Initial classification

		downward	non-downward			
		1-subtree reducible		2-subtree reducible	3-subtree reducible	
non-*-free	non-monotone	$\mathcal{N}(\pi, \overline{\pi}, \cap, -, {}^*)$	$\mathcal{N}({}^\smallfrown, \pi, \overline{\pi}, \cap, {}^*)$	$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi, \overline{\pi}, {}^*)$	$\mathcal{N}$	non-monotone
	monotone	$\mathcal{N}(\pi, \cap, {}^*)$	$\mathcal{N}({}^\smallfrown, \pi, \cap, {}^*)$	$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi, {}^*)$	$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi, \cap, {}^*)$	monotone
non-local, *-free	non-monotone			$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi, \overline{\pi})$	$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi, \overline{\pi}, \cap, -)$	non-monotone
	monotone			$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi)$	$\mathcal{N}(\mathrm{di}, {}^\smallfrown, \pi, \cap)$	monotone
local	non-monotone	$\mathcal{N}(\pi, \overline{\pi}, \cap, -)$	$\mathcal{N}({}^\smallfrown, \pi, \overline{\pi}, \cap)$		$\mathcal{N}({}^\smallfrown, \pi, \overline{\pi}, \cap, -)$	non-monotone
	monotone	$\mathcal{N}(\cap, -)$ $\mathcal{N}(\pi, \cap)$	$\mathcal{N}({}^\smallfrown, \pi, \cap)$			monotone

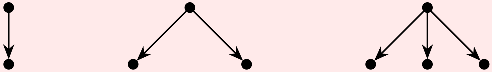
Recognizing branches: labeled trees



Only select the three on the right

- Not possible in $\mathcal{N}(^*)$.
- Using projection: $\pi_1[\mathit{FriendOf}] \circ \pi_1[\mathit{ParentOf}]$.
- Using converse: $\mathit{FriendOf}^{\frown} \circ \mathit{ParentOf}$.

Recognizing branches: unlabeled trees



Definition

A *k-subtree-reduction step* on tree $\mathcal{T}$ consists of removing a subtree rooted at a node n with parent m, given that parent m has at least k other children isomorphic to n.

Theorem

1. $\mathcal{N}(\mathrm{di}, \cap)$ *and* $\mathcal{N}(\smallfrown, -)$ *are closed under* 3*-subtree-reductions.*
2. $\mathcal{N}(\mathrm{di}, \smallfrown, \pi, \overline{\pi}, {}^{*})$ *is closed under* 2*-subtree-reductions.*
3. $\mathcal{N}(\smallfrown, \pi, \overline{\pi}, \cap, {}^{*})$ *is closed under* 1*-subtree-reductions.*

Monotonicity

- Without negation you cannot 'forbid' structures.
- $\overline{\pi}$ always provides negation: $\mathit{NonParent} = \overline{\pi}_1[\mathit{ParentOf}]$.
- Without negation, only a few Boolean queries are possible.

Theorem

1. *On unlabeled chains, we have* $\mathcal{N}(\mathrm{di}, {}^\frown, \pi, \cap, {}^*) \preceq_{\mathrm{bool}} \mathcal{N}()$.
2. *On unlabeled trees, we have* $\mathcal{N}({}^\frown, \pi, \cap, {}^*) \preceq_{\mathrm{bool}} \mathcal{N}()$.

Theorem

Already on unlabeled chains, we have $\mathcal{N}(\overline{\pi}) \npreceq_{\mathrm{bool}} \mathcal{N}(\mathrm{di}, {}^\frown, \pi, \cap, {}^*)$.

The Kleene-star

- Is not expressible in first-order logic.
- Path queries: always adds expressive power.
- Labeled structures: always adds expressive power.

Theorem

1. *On unlabeled chains, we have* $\mathcal{N}(\mathrm{di}, ^\frown, \pi, \cap, ^*) \preceq_{\mathrm{bool}} \mathcal{N}()$.
2. *On unlabeled trees, we have* $\mathcal{N}(^\frown, \pi, \cap, ^*) \preceq_{\mathrm{bool}} \mathcal{N}()$.

Theorem

Let $\mathcal{F} \subseteq \{\mathrm{di}, ^\frown, \pi\}$. *On unlabeled trees, we have* $\mathcal{N}(\mathcal{F} \cup \{^*\}) \preceq_{\mathrm{bool}} \mathcal{N}(\mathcal{F})$.

Downward queries: path semantics

$$\begin{tikzcd}
 & \overset{\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)}{\mathcal{N}(\overline{\pi},^*)} \\
\overset{\mathcal{N}(\pi,\overline{\pi},\cap,-)}{\mathcal{N}(\overline{\pi})} \arrow[ur] & \\
 & \overset{\mathcal{N}(\pi,\cap,^*)}{\mathcal{N}(\pi,^*)} \arrow[uu] \\
\overset{\mathcal{N}(\pi,\cap)}{\mathcal{N}(\pi)} \arrow[uu] \arrow[ur] & \\
 & \overset{\mathcal{N}(\cap,-,^*)}{\mathcal{N}(^*)} \arrow[uu] \\
\overset{\mathcal{N}(\cap,-)}{\mathcal{N}()} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

- $\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$ is downward.
- $^\frown$ and di are not downward.
- Downward queries can be expressed by condition automata.

Theorem

Let $\mathcal{F} \subseteq \{\pi,\overline{\pi},\cap,-,^*\}$. On labeled trees, we have $\mathcal{N}(\mathcal{F}) \preceq_{\text{path}} \mathcal{N}(\underline{\mathcal{F}} - \{\cap,-\})$.

Downward queries: Boolean semantics

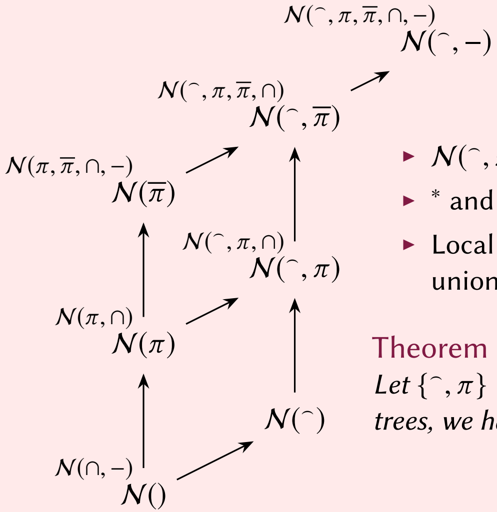
Unlabeled Trees

$$
\begin{tikzcd}
\begin{array}{r} \mathcal{N}(\pi,\overline{\pi},\cap,-,^*) \\ \mathcal{N}(\overline{\pi},^*) \end{array} \\
\begin{array}{r} \mathcal{N}(\pi,\overline{\pi},\cap,-) \\ \mathcal{N}(\overline{\pi}) \end{array} \arrow[u] \\
\begin{array}{r} \mathcal{N}(\cap,-,^*) \\ \mathcal{N}(\pi,\cap,^*) \\ \mathcal{N}() \end{array} \arrow[u]
\end{tikzcd}
$$

Labeled Trees

$$
\begin{tikzcd}
 & \begin{array}{r} \mathcal{N}(\pi,\overline{\pi},\cap,-,^*) \\ \mathcal{N}(\overline{\pi},^*) \end{array} \\
\begin{array}{r} \mathcal{N}(\pi,\overline{\pi},\cap,-) \\ \mathcal{N}(\overline{\pi}) \end{array} \arrow[ur] & \\
 & \begin{array}{r} \mathcal{N}(\pi,\cap,^*) \\ \mathcal{N}(\pi,^*) \end{array} \arrow[uu] \\
\begin{array}{r} \mathcal{N}(\pi,\cap) \\ \mathcal{N}(\pi) \end{array} \arrow[uu] \arrow[ur] & \\
 & \begin{array}{r} \mathcal{N}(\cap,-,^*) \\ \mathcal{N}(^*) \end{array} \arrow[uu] \\
\begin{array}{r} \mathcal{N}(\cap,-) \\ \mathcal{N}() \end{array} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

Local queries: path semantics



- $\mathcal{N}(^\frown, \pi, \overline{\pi}, \cap, -)$ is local.
- * and di are not local.
- Local queries can be expressed by unions of condition tree queries.

Theorem

Let $\{^\frown, \pi\} \subseteq \mathcal{F} \subseteq \{^\frown, \pi, \overline{\pi}, \cap\}$. On labeled trees, we have $\mathcal{N}(\mathcal{F}) \preceq_{\text{path}} \mathcal{N}(\mathcal{F} - \{\cap\})$.

Local queries: Boolean semantics

Unlabeled Trees

$\mathcal{N}(^\frown, \pi, \overline{\pi}, \cap, -)$

$\mathcal{N}(\pi, \overline{\pi}, \cap, -)$
$\mathcal{N}(^\frown, \pi, \overline{\pi}, \cap)$

$\mathcal{N}(\cap, -)$
$\mathcal{N}(^\frown, \pi, \cap)$

$$\begin{tikzcd} & \mathcal{N}(^\frown, -) \\ \mathcal{N}(\overline{\pi}) \arrow[ur] & \\ \mathcal{N}() \arrow[u] & \end{tikzcd}$$

Labeled Trees

$\mathcal{N}(^\frown, \pi, \overline{\pi}, \cap, -)$

$\mathcal{N}(\pi, \overline{\pi}, \cap, -)$
$\mathcal{N}(^\frown, \pi, \overline{\pi}, \cap)$

$\mathcal{N}(\pi, \cap)$
$\mathcal{N}(^\frown, \pi, \cap)$

$\mathcal{N}(\cap, -)$

$$\begin{tikzcd} & \mathcal{N}(^\frown, -) \\ \mathcal{N}(\overline{\pi}) \arrow[ur] & \\ \mathcal{N}(\pi) = \mathcal{N}(^\frown) \arrow[u] & \\ \mathcal{N}() \arrow[u] & \end{tikzcd}$$

Conclusion

- We fully established relationships between:
  - downward fragments;
  - local fragments.
- On trees: solved most cases not involving *.
- On chains: solved most cases not involving di or *.

Main open problem

Let $\{\mathrm{di}, \overline{\pi}, \cap\} \subseteq \mathcal{F} \subseteq \{\mathrm{di}, ^\frown, \pi, \overline{\pi}, \cap, ^*\}$ and let $z \in \{\mathrm{bool}, \mathrm{path}\}$. With respect to either labeled trees, unlabeled trees, labeled chains, or unlabeled chains, do we have a collapse $\mathcal{N}(\mathcal{F} \cup \{-\}) \leq_z \mathcal{N}(\mathcal{F})$ or not?

Part III

On Tarski's Relation Algebra

THE SEMI-JOIN ALGEBRA

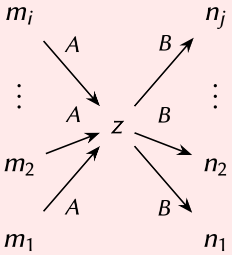
Naive query evaluation: an inefficient example

Return pairs of (great-grandparent, friend)

$$\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}] \circ \mathit{FriendOf}.$$

1. Compute (grandparent, grandchild):
   $X = \mathit{ParentOf} \circ \mathit{ParentOf}$
2. Compute (great-grandparent, great-grandchild):
   $Y = \mathit{ParentOf} \circ X$
3. Throw away the great-grandchildren:
   $Z = \pi_1[Y]$
4. Compute (great-grandparent, friend):
   $\text{Result} = Z \circ \mathit{FriendOf}$

Introducing the semi-join algebra

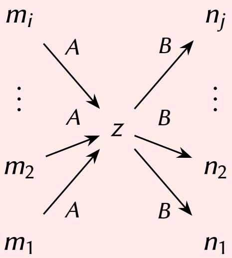


Composition: querying for paths

Consider $A \circ B$:

- yields (begin, end)-nodes connected by AB.
- yields $i \cdot j$ node pairs in total.

Introducing the semi-join algebra



Composition: querying for paths

Consider $A \circ B$:

- yields (begin, end)-nodes connected by AB.
- yields $i \cdot j$ node pairs in total.

Checking for existence of paths

Consider the semi-join $A \ltimes B$:

- yields the edges in A that connect to B.
- yields i node pairs in total.

Optimize query evaluation: using semi-joins

Return pairs of (great-grandparent, friend)

$$\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}] \circ \mathit{FriendOf}.$$

1. Compute (grandparent, ???):
   $X = \mathit{ParentOf} \ltimes \mathit{ParentOf}$
2. Compute (great-grandparent, ???):
   $Y = \mathit{ParentOf} \ltimes (X)$
3. Throw away ???:
   $Z = \pi_1[Y]$
4. Compute (great-grandparent, friend):
   $\text{Result} = Z \rtimes \mathit{FriendOf}$

$$\pi_1[\mathit{ParentOf} \ltimes (\mathit{ParentOf} \ltimes \mathit{ParentOf})] \rtimes \mathit{FriendOf}.$$

Projection-equivalence

Requiring the rewrite to guarantee *path-equivalence* is too strong!

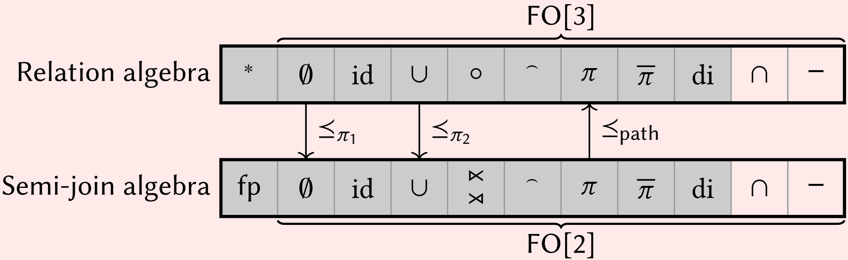
Left-projection-equivalence. We only need the first projection.

$$\mathit{ParentOf} \circ \mathit{ParentOf} \equiv_{\pi_1} \mathit{ParentOf} \ltimes \mathit{ParentOf}$$
$$\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf}] \equiv_{\text{path}} \pi_1[\mathit{ParentOf} \ltimes \mathit{ParentOf}]$$

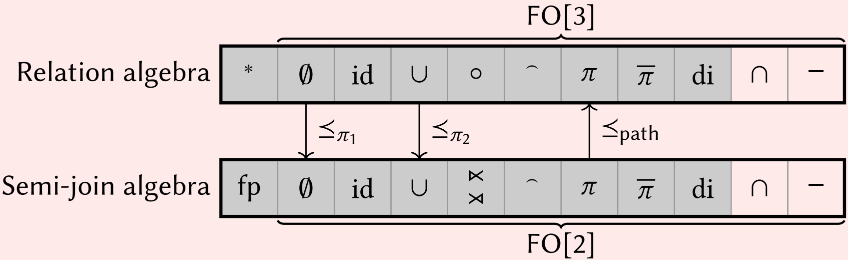
Right-projection-equivalence. We only need the second projection.

$$\mathit{ParentOf} \circ \mathit{ParentOf} \equiv_{\pi_2} \mathit{ParentOf} \rtimes \mathit{ParentOf}$$
$$\pi_2[\mathit{ParentOf} \circ \mathit{ParentOf}] \equiv_{\text{path}} \pi_2[\mathit{ParentOf} \rtimes \mathit{ParentOf}]$$

The main result

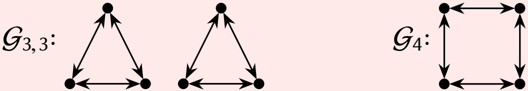


The main result



Intersection and difference

- Consider $(\mathcal{E} \circ \mathcal{E}) \cap \mathcal{E}$:



- We can allow ∩ and − on basic expressions (edges, ∅, id, di).

Partial rewriting to the semi-join algebra

Rewrite functions $\tau(e) \equiv_{\text{path}} e$, $\tau_{\pi_1}(e) \equiv_{\pi_1} e$ and $\tau_{\pi_2}(e) \equiv_{\pi_2} e$

Helper functions $\tau_{\circ_1}(e; \varepsilon) \equiv_{\pi_1} e \ltimes \varepsilon$ and $\tau_{\circ_2}(e; \varepsilon) \equiv_{\pi_2} \varepsilon \rtimes e$

Example

$e = \pi_1[((\mathit{WorksOn} \circ \mathit{WorksOn}^{\frown}) \cap \mathit{FriendOf}) \circ \mathit{EditorOf}] \circ \mathit{StudentOf}$

Rewriting e using $\tau(e)$:

$$
\begin{aligned}
\tau(e) &= \tau_{\pi_2}(\pi_1[((W \circ W^{\frown}) \cap F) \circ E]) \rtimes \tau(S) \\
&= \pi_1[\tau_{\pi_1}(((W \circ W^{\frown}) \cap F) \circ E)] \rtimes S \\
&= \pi_1[\tau_{\circ_1}((W \circ W^{\frown}) \cap F; E)] \rtimes S \\
&= \pi_1[(\tau(W \circ W^{\frown}) \cap \tau(F)) \ltimes E \rtimes S] \\
&= \pi_1[((\tau(W) \circ \tau(W^{\frown})) \cap F) \ltimes E \rtimes S] \\
&= \pi_1[((W \circ W^{\frown}) \cap F) \ltimes E] \rtimes S.
\end{aligned}
$$

Query rewriting and optimization

- Cost of each operator ✗.
- Input size of each operator ✗.
- Number of evaluation steps ✗.

Query rewriting and optimization

- Cost of each operator ✓.
- Input size of each operator ✗.
- Number of evaluation steps ✗.

Cost of each operator

FO[3]

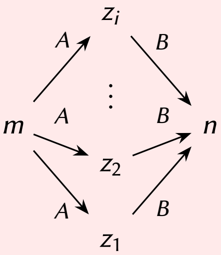
Relation algebra	$*$	$\emptyset$	id	$\cup$	$\circ$	$\smallfrown$	π	$\overline{\pi}$	di	$\cap$	$-$
Semi-join algebra	fp	$\emptyset$	id	$\cup$	$\ltimes$ $\rtimes$	$\smallfrown$	π	$\overline{\pi}$	di	$\cap$	$-$

FO[2]

Query rewriting and optimization

- Cost of each operator ✓.
- Input size of each operator ✗.
- Number of evaluation steps ✗.

Input size of each operator



$$A \circ B \equiv_{\pi_1} A \ltimes B$$

- $A \circ B$ yields $\{(m, n)\}$.
- $A \ltimes B$ yields $\{(m, z_1), (m, z_2), \ldots, (m, z_i)\}$.

Query rewriting and optimization

- Cost of each operator ✓.
- Input size of each operator ✓.
- Number of evaluation steps ✗.

Input size of each operator

$$\begin{tikzcd}
& z_i \arrow[ddr, "B"] & \\
& \vdots & \\
m \arrow[uur, "A"] \arrow[r, "A"] \arrow[dr, "A"'] & z_2 \arrow[r, "B"] & n \\
& z_1 \arrow[ur, "B"'] &
\end{tikzcd}$$

$$A \circ B \equiv_{\pi_1} A \ltimes B$$

- $A \circ B$ yields $\{(m, n)\}$.
- $A \ltimes B$ yields $\{(m, z_1), (m, z_2), \ldots, (m, z_i)\}$.
- Fix: add projection-steps into algorithms.

Query rewriting and optimization

- Cost of each operator ✓.
- Input size of each operator ✓.
- Number of evaluation steps $\sim$.

Number of evaluation steps

- Number of evaluation steps can increase.
- Complexity of each individual step can significantly decrease.

Example

Consider $e = \pi_1[(\ell \circ \ell) \circ (\ell \circ \ell)]$ and $\tau(e) = \pi_1[\ell \ltimes (\ell \ltimes (\ell \ltimes \ell))]$.
We can evaluate e in three steps:

1. compute $X = \ell \circ \ell$;
2. compute $Y = X \circ X$;
3. compute $\pi_1[Y]$.

Other expensive constructs: id, di, and $\overline{\pi}$

Idea: replace common use cases

- Replace id by equality-selections

$$(\mathit{FriendOf} \circ \mathit{WorksWith}) \cap \mathrm{id} \equiv_{\mathrm{path}} \sigma_{=}(\mathit{FriendOf} \circ \mathit{WorksWith}).$$

- Replace di by inequality-selections

$$(\mathit{FriendOf} \circ \mathit{FriendOf}) \cap \mathrm{di} \equiv_{\mathrm{path}} \sigma_{\neq}(\mathit{FriendOf} \circ \mathit{FriendOf}).$$

- Replace $\overline{\pi}$ by anti-semi-joins

$$\mathit{FriendOf} \circ \overline{\pi}_1[\mathit{ParentOf}] \equiv_{\mathrm{path}} \mathit{FriendOf} \,\overline{\ltimes}\, \mathit{ParentOf}.$$

Further optimization: filter steps in practical queries

Find friend suggestions for Alice:

1. Compute all friends-of-friends (excluding friends and oneself):

$$\mathit{FriendSuggestions} = (\mathit{FriendOf} \circ \mathit{FriendOf}) - (\mathit{FriendOf} \cup \mathrm{id}).$$

2. Filter the first column on Alice.
3. Keep the second column.

Further optimization: filter steps in practical queries

Find friend suggestions for Alice:

1. Compute all friends-of-friends (excluding friends and oneself):

$$\mathit{FriendSuggestions} = (\mathit{FriendOf} \circ \mathit{FriendOf}) - (\mathit{FriendOf} \cup \mathrm{id}).$$

2. Filter the first column on Alice.
3. Keep the second column.

Incorporate filter-step in the relation algebra

$$\pi_2[(\langle \mathit{Alice} \rangle \rtimes \mathit{FriendOf}) \rtimes (\mathit{FriendOf} \ \overline{\rtimes} \\ \pi_2[(\langle \mathit{Alice} \rangle \rtimes \mathit{FriendOf}) \cup \langle \mathit{Alice} \rangle])]$$

Conclusion

Queries in the relation algebra can be partially rewritten into semi-join algebra queries that are easier to evaluate.

Future work

- Implementation in real systems.
- Interactions with other optimization techniques.
- Elimination of intersection and difference.

Part IV

On Tarski's Relation Algebra

CONCLUSION

Conclusion

Questions

- Are all these operators necessary?
- What does each operator add?
- When can we replace complex operators by simpler ones?

This work

Improve understanding of the relation algebra

- when querying trees or chains;
- in comparison with the semi-join algebra.