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

\[ \text{FriendsOfGgp} = \pi_1 \left( \text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf} \right) \circ \text{FriendOf} \]
Relation algebra: graphs and queries

\[ \text{FriendsOfGgp} = \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf} \]
Relation algebra: graphs and queries

FriendsOfGgp = π₁[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf}
Relation algebra: graphs and queries

$$\text{FriendsOfGgp} = \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf}$$
Relation algebra: graphs and queries

\[ \text{FriendsOfGgp} = \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf} \]
Relation algebra: other operators

\[
\text{ChildOf} = \text{ParentOf}^\sim
\]
\[
\text{AcquaintanceOf} = \text{FriendOf} \cup \text{WorksWith}
\]
\[
\text{WorkFriendOf} = \text{FriendOf} \cap \text{WorksWith}
\]
\[
\text{NonWorkFriend} = \text{FriendOf} - \text{WorksWith}
\]
\[
\text{Parent} = \pi_1[\text{ParentOf}]
\]
\[
\text{NonParent} = \overline{\pi}_1[\text{ParentOf}]
\]
\[
\text{GrandParentOf} = \text{ParentOf} \circ \text{ParentOf}
\]
\[
\text{AncestorOf} = [\text{ParentOf}]^+
\]

Also: constants \( \emptyset, \text{id}, \text{di} \).
Graph query languages

Tarski’s Relation Algebra, FO[3]

*  ∅  id  ∪  ∩  π  ¬π  ∩  ¬  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:

\[ \text{FriendOf} \cap \text{WorksWith} \equiv_{\text{path}} \text{FriendOf} - (\text{FriendOf} - \text{WorksWith}). \]

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

\[ \text{FriendOf} \sim \circ \text{ParentOf} \equiv_{\text{bool}} \pi_1[\text{FriendOf}] \cap \pi_1[\text{ParentOf}]. \]

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 \( \mathcal{L}_1 \leq_z \mathcal{L}_2 \)).
Background: query fragments

Let \( \mathcal{F} \subseteq \{ \text{di}, \setminus, \pi, \overline{\pi}, \cap, -, * \} \).

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

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

Examples

- \( \mathcal{N}(\overline{\pi}) = \mathcal{N}(\pi, \overline{\pi}) \).
- \( \mathcal{N}(\setminus, -) = \mathcal{N}(\setminus, \pi, \overline{\pi}, \cap, -) \).
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

<table>
<thead>
<tr>
<th></th>
<th>Downward</th>
<th>Non-downward</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-*-free</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-local, *-free</td>
<td>$\mathcal{N}(\pi, \overline{\pi}, \cap, -, \ast)$</td>
<td>$\mathcal{N}(\overline{\cap}, \pi, \overline{\pi}, \cap, \ast)$</td>
</tr>
<tr>
<td>Local</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{N}(\pi, \overline{\pi}, \cap, -)$</td>
<td>$\mathcal{N}(\overline{\cap}, \pi, \overline{\pi}, \cap)$</td>
<td>$\mathcal{N}(\overline{\cap}, \pi, \overline{\pi}, \cap, -)$</td>
</tr>
<tr>
<td>$\mathcal{N}(\cap, -)$</td>
<td>$\mathcal{N}(\overline{\cap}, \pi, \overline{\pi})$</td>
<td>$\mathcal{N}(\overline{\cap}, \pi, \overline{\pi})$</td>
</tr>
</tbody>
</table>

- **1-subtree reducible**
- **2-subtree reducible**
- **3-subtree reducible**

- **Non-monotone**
- **Monotone**
- **Non-*-free**
- ***-free**
Recognizing branches: labeled trees

Only select the three on the right

- Not possible in $\mathcal{N}(\ast)$.
- Using projection: $\pi_1[FriendOf] \circ \pi_1[ParentOf]$.
- Using converse: $FriendOf^\wedge \circ ParentOf$. 
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. $\mathcal{N}(\text{di}, \cap)$ and $\mathcal{N}(\neg, -)$ are closed under $3$-subtree-reductions.
2. $\mathcal{N}(\text{di}, \neg, \pi, \bar{\pi}, *)$ is closed under $2$-subtree-reductions.
3. $\mathcal{N}(\neg, \pi, \bar{\pi}, \cap, *)$ is closed under $1$-subtree-reductions.
Monotonicity

- Without negation you cannot ‘forbid’ structures.
- $\pi$ always provides negation: $\text{NonParent} = \pi_1[\text{ParentOf}]$.
- Without negation, only a few Boolean queries are possible.

Theorem

1. On unlabeled chains, we have $N(\text{di, } \neg, \pi, \cap, *) \leq_{\text{bool}} N()$.
2. On unlabeled trees, we have $N(\neg, \pi, \cap, *) \leq_{\text{bool}} N()$.

Theorem

Already on unlabeled chains, we have $N(\pi) \not\leq_{\text{bool}} N(\text{di, } \neg, \pi, \cap, *)$. 
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 $N(di, \neg, \pi, \cap, \ast) \leq_{\text{bool}} N()$.
2. On unlabeled trees, we have $N(\neg, \pi, \cap, \ast) \leq_{\text{bool}} N()$.

Theorem

Let $\mathcal{F} \subseteq \{di, \neg, \pi\}$. On unlabeled trees, we have $N(\mathcal{F} \cup \{\ast\}) \leq_{\text{bool}} N(\mathcal{F})$. 
Downward queries: path semantics

\[ N(\pi, \bar{\pi}, \cap, -, \ast) \]

\[ N(\bar{\pi}, \ast) \]

\[ N(\pi, \cap, \ast) \]

\[ N(\pi, \ast) \]

\[ N(\cap, \ast) \]

\[ N(\ast) \]

\[ N() \]

- \( N(\pi, \bar{\pi}, \cap, -, \ast) \) is downward.
- \( \neg \) and \( d_i \) are not downward.
- Downward queries can be expressed by condition automata.

**Theorem**

Let \( \mathcal{F} \subseteq \{\pi, \bar{\pi}, \cap, -, \ast\} \). On labeled trees, we have \( N(\mathcal{F}) \leq_{\text{path}} N(\bar{\mathcal{F}} - \{\cap, -\}) \).
Downward queries: Boolean semantics

Unlabeled Trees

\[ N(\pi, \bar{\pi}, \cap, -, *) \]
\[ N(\bar{\pi}, *) \]
\[ N(\pi, \bar{\pi}, \cap, -) \]
\[ N(\bar{\pi}) \]
\[ N(\cap, -, *) \]
\[ N(\cap, *) \]
\[ N(\pi, \cap, *) \]
\[ N(\pi, \cap) \]
\[ N(\cap, -) \]
\[ N(*) \]

Labeled Trees

\[ N(\pi, \bar{\pi}, \cap, -, *) \]
\[ N(\bar{\pi}, *) \]
\[ N(\pi, \bar{\pi}, \cap, -) \]
\[ N(\bar{\pi}) \]
\[ N(\cap, -, *) \]
\[ N(*) \]
\[ N(\pi, \cap, *) \]
\[ N(\pi, \cap) \]
\[ N(\cap, -) \]
\[ N(\pi, *) \]
Local queries: path semantics

\[ N(\neg, \pi, \overline{\pi}, \cap, -) \]

\[ N(\neg, -) \]

\[ N(\neg, \pi, \overline{\pi}, \cap) \]

\[ N(\neg, \pi) \]

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

**Theorem**

Let \( \{\neg, \pi\} \subseteq \mathcal{F} \subseteq \{\neg, \pi, \overline{\pi}, \cap\} \). On labeled trees, we have \( N(\mathcal{F}) \preceq_{\text{path}} N(\mathcal{F} - \{\cap\}) \).
Local queries: Boolean semantics

Unlabeled Trees

\[ \mathcal{N}(\land, \pi, \bar{\pi}, \cap, \neg) \]
\[ \mathcal{N}(\land, \neg) \]
\[ \mathcal{N}(\bar{\pi}) \]
\[ \mathcal{N}(\pi, \bar{\pi}, \cap, \neg) \]
\[ \mathcal{N}(\land, \pi, \bar{\pi}, \cap) \]
\[ \mathcal{N}(\pi, \bar{\pi}, \cap) \]
\[ \mathcal{N}(\neg, \pi, \bar{\pi}, \cap) \]
\[ \mathcal{N}(\pi, \bar{\pi}, \cap) \]

Labeled Trees

\[ \mathcal{N}(\land, \pi, \bar{\pi}, \cap, \neg) \]
\[ \mathcal{N}(\land, \neg) \]
\[ \mathcal{N}(\bar{\pi}) \]
\[ \mathcal{N}(\pi, \bar{\pi}, \cap, \neg) \]
\[ \mathcal{N}(\land, \pi, \bar{\pi}, \cap) \]
\[ \mathcal{N}(\pi, \bar{\pi}, \cap) \]
\[ \mathcal{N}(\neg, \pi, \bar{\pi}, \cap) \]
\[ \mathcal{N}(\pi, \bar{\pi}, \cap) \]

\[ \mathcal{N}(\pi) = \mathcal{N}(\neg) \]
Conclusion

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

Main open problem

Let $\{di, \overline{\pi}, \cap\} \subseteq \mathcal{F} \subseteq \{di, \neg, \pi, \overline{\pi}, \cap, \ast\}$ and let $z \in \{\text{bool, path}\}$. 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)

\[ \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf}. \]

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

\[ \pi_1[ParentOf \circ ParentOf \circ ParentOf] \circ FriendOf . \]

1. Compute (grandparent, ???):
   \[ X = ParentOf \Join ParentOf \]

2. Compute (great-grandparent, ???):
   \[ Y = ParentOf \Join (X) \]

3. Throw away ???:
   \[ Z = \pi_1[Y] \]

4. Compute (great-grandparent, friend):
   Result = \[ Z \Join FriendOf \]

\[ \pi_1[ParentOf \Join (ParentOf \Join ParentOf)] \Join FriendOf . \]
Projection-equivalence

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

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

\[\text{ParentOf} \circ \text{ParentOf} \equiv_{\pi_1} \text{ParentOf} \bowtie \text{ParentOf} \]
\[\pi_1[\text{ParentOf} \circ \text{ParentOf}] \equiv_{\text{path}} \pi_1[\text{ParentOf} \bowtie \text{ParentOf}]\]

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

\[\text{ParentOf} \circ \text{ParentOf} \equiv_{\pi_2} \text{ParentOf} \bowtie \text{ParentOf} \]
\[\pi_2[\text{ParentOf} \circ \text{ParentOf}] \equiv_{\text{path}} \pi_2[\text{ParentOf} \bowtie \text{ParentOf}]\]
The main result

Relation algebra

Semi-join algebra

\[ \leq_{\pi_1} \leq_{\pi_2} \leq_{\text{path}} \]

Intersection and difference

Consider \((E \circ E) \cap E\):

We can allow \(\cap\) and \(\neg\) on basic expressions (edges, \(\emptyset\), \(\text{id}\), \(\text{di}\)).
The main result

Relation algebra

Semi-join algebra

Intersection and difference

- Consider \((E \circ E) \cap E\):

\[ \mathcal{G}_{3,3}: \]

\[ \mathcal{G}_{4}: \]

- We can allow \(\cap\) and \(\neg\) on basic expressions (edges, \(\emptyset\), \(\text{id}\), \(\text{di}\)).
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 $\tau_{\circ_1}(e; \varepsilon) \equiv_{\pi_1} e \bowtie \varepsilon$ and $\tau_{\circ_2}(e; \varepsilon) \equiv_{\pi_2} \varepsilon \bowtie e$

Example

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

Rewriting $e$ using $\tau(e)$:

$$
\tau(e) = \tau_{\pi_2}(\pi_1[((\text{W} \circ \text{W}^\sim) \cap \text{F}) \circ \text{E}]) \bowtie \tau(\text{S})
= \pi_1[\tau_{\pi_1}(((\text{W} \circ \text{W}^\sim) \cap \text{F}) \circ \text{E})] \bowtie \text{S}
= \pi_1[\tau_{\circ_1}((\text{W} \circ \text{W}^\sim) \cap \text{F}; \text{E})] \bowtie \text{S}
= \pi_1[\tau(\text{W} \circ \text{W}^\sim) \cap \tau(\text{F})) \bowtie \text{E} \bowtie \text{S}]
= \pi_1[((\tau(\text{W}) \circ \tau(\text{W}^\sim)) \cap \text{F}) \bowtie \text{E} \bowtie \text{S}]
= \pi_1[((\text{W} \circ \text{W}^\sim) \cap \text{F}) \bowtie \text{E}] \bowtie \text{S}.
$$
Query rewriting and optimization

- Cost of each operator $X$.
- Input size of each operator $X$.
- Number of evaluation steps $X$. 

Relation algebra

Semi-join algebra

$\text{FO}[2]$

$\text{FO}[3]$

$\emptyset$

$id$

$\cup$

$\circ$

$\dashv$

$\pi$

$\cap$

$-^

$fp$

$\emptyset$

$\cup$

$\smallfrown$

$\dashv$
Query rewriting and optimization

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

Cost of each operator

Relation algebra

<table>
<thead>
<tr>
<th>Operator</th>
<th>FO[3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>✓</td>
</tr>
<tr>
<td>id</td>
<td>✓</td>
</tr>
<tr>
<td>∪</td>
<td>✓</td>
</tr>
<tr>
<td>○</td>
<td>✓</td>
</tr>
<tr>
<td>¬</td>
<td>✓</td>
</tr>
<tr>
<td>π</td>
<td>✓</td>
</tr>
<tr>
<td>¬π</td>
<td>✓</td>
</tr>
<tr>
<td>di</td>
<td>✓</td>
</tr>
<tr>
<td>∩</td>
<td>✓</td>
</tr>
<tr>
<td>¬</td>
<td>✓</td>
</tr>
</tbody>
</table>

Semi-join algebra

<table>
<thead>
<tr>
<th>Operator</th>
<th>FO[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>✓</td>
</tr>
<tr>
<td>Ø</td>
<td>✓</td>
</tr>
<tr>
<td>id</td>
<td>✓</td>
</tr>
<tr>
<td>∪</td>
<td>✓</td>
</tr>
<tr>
<td>×</td>
<td>✗</td>
</tr>
<tr>
<td>¬</td>
<td>✓</td>
</tr>
<tr>
<td>π</td>
<td>✓</td>
</tr>
<tr>
<td>¬π</td>
<td>✓</td>
</tr>
<tr>
<td>di</td>
<td>✓</td>
</tr>
<tr>
<td>∩</td>
<td>✓</td>
</tr>
<tr>
<td>¬</td>
<td>✓</td>
</tr>
</tbody>
</table>
Query rewriting and optimization

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

Input size of each operator

\[ A \circ B \equiv_{\pi_1} A \Join B \]

- \( A \circ B \) yields \( \{(m, n)\} \).
- \( A \Join B \) 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 X.

Input size of each operator

\[ A \circ B \equiv_{\pi_1} A \Join B \]

- \( A \circ B \) yields \( \{(m, n)\} \).
- \( A \Join B \) 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 ~.

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 \( \tau(e) = \pi_1[\ell \heartsuit (\ell \heartsuit (\ell \heartsuit \ell)))] \).

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
  \[
  (\text{FriendOf} \circ \text{WorksWith}) \cap \text{id} \equiv_{\text{path}} \sigma_{=} (\text{FriendOf} \circ \text{WorksWith}).
  \]

- Replace di by inequality-selections
  \[
  (\text{FriendOf} \circ \text{FriendOf}) \cap \text{di} \equiv_{\text{path}} \sigma_{\neq} (\text{FriendOf} \circ \text{FriendOf}).
  \]

- Replace \( \overline{\pi} \) by anti-semi-joins
  \[
  \text{FriendOf} \circ \overline{\pi}_1[\text{ParentOf}] \equiv_{\text{path}} \text{FriendOf} \bowtie \text{ParentOf}.
  \]
Further optimization: filter steps in practical queries

Find friend suggestions for Alice:

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

   \[ \text{FriendSuggestions} = (\text{FriendOf} \circ \text{FriendOf}) - (\text{FriendOf} \cup \text{id}). \]

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):

   \[ \text{FriendSuggestions} = (\text{FriendOf} \circ \text{FriendOf}) - (\text{FriendOf} \cup \text{id}). \]

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

Incorporate filter-step in the relation algebra

\[
\pi_2[(\langle Alice \rangle \bowtie \text{FriendOf}) \bowtie (\text{FriendOf} \not\bowtie \pi_2[(\langle Alice \rangle \bowtie \text{FriendOf}) \cup \langle Alice \rangle])]
\]
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.