Associativity Rules for Native XML Databases*

Andreas M. Weiner, Christian Mathis, and Theo Härder
Databases and Information Systems Group
Department of Computer Science
University of Kaiserslautern
D-67653 Kaiserslautern, Germany
{weiner, mathis, haerder}@informatik.uni-kl.de

March 10, 2008

1 Introduction

An associativity rule empowers a relational plan generator to perform movements in the search space of semantically equivalent queries by changing the join order. However, in the world of XML query languages, a single associativity rule is not sufficient, due to the dualism of content and structure. Instead, we will need a rule for reordering content-based joins and a set of associativity rules for structural joins that take combinations of several axes as well as early duplicate elimination and sorting into account.

Every tuple operator (TO) that forms the root node of a tree-structured query graph has to perform duplicate elimination and sorting, independent of its operator type. On the other hand, TOs that have incoming and outgoing edges potentially need to perform duplicate elimination. Fortunately, not every join operator needs additional duplicate elimination operations. For example, a full-join TO will not create any duplicates, independent of the structural predicate it evaluates. On the other hand, a semi-join TO can create duplicates on its output.

We can partition binary semi-join operators into two different equivalence classes depending on the emergence of duplicates: (1) semi-join operators where only tuples of one incoming tuple sequence can contain duplicates after join evaluation (join operators that evaluate `parent/child` or `previous-/following-sibling` axes), and (2) semi-join operators where both incoming tuple sequences can contain duplicates after join evaluation (join operators that evaluate `ancestor/descendant` or `previous/following` axes).

Let a denote the left join partner, b denote the right join partner of a binary structural semi-join j that evaluates the `child` axis. If j only produces a

*Appendix to Weiner et al.[WMH08]

tuples satisfying the structural predicate, then duplicate elimination has to be performed, because every node can have multiple child nodes. In contrast, if j only delivers b tuples to consuming operators, then duplicate elimination is not needed, because every node has at most one parent node. If j would evaluate a `descendant` axis, then duplicate elimination could be necessary in both cases, because every node can have multiple descendants and multiple ancestors.

As mentioned before, to provide a complete set of associativity rules, all combinations of axes have to be considered. Additionally, different output nodes need to be taken into account. A node is called an *output node* if its tuple sequence contributes to the query result or is processed in a subsequent TO.

Figure 1 shows the associativity rule for one output node and two adjacent semi-join operators that evaluate the `descendant` axis[1]. To support a more fine-granular treatment of sorting and duplicate elimination, we replace a call to the `ddo` function, which only eliminates duplicates, by `D`.

We assume that the output of each join operator is implicitly sorted by the node that is used by a subsequent TO or that contributes to the final result. On the left hand side of Figure 1, a structural full-join is performed between tuples of TO A and B which needs no additional sorting or duplicate elimination. The following semi-join operator requires duplicate elimination and sorting for two reasons: (1) it has only incoming edges, (2) each tuple of the incoming tuple sequence can have multiple descendant c nodes. On the right side, a structural join is performed first between TO B and TO C. Because this structural relationship is evaluated using a semi-join, we need additional duplicate elimination, because every b node can have multiple c descendants. The following semi-join operator requires duplicate elimination for the same reason, but it needs no additional sorting, because of our implicit sorting assumption.

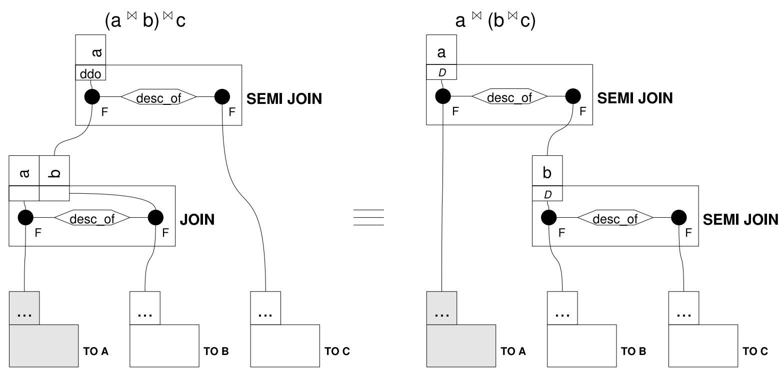


Figure 1: Associativity rule for two `descendant` axes and one output node

[1]This query graph corresponds to the following XPath expression: `a[.//b//c]`.

2 Associativity Rules

This section shows the associativity rules for binary structural join operators. To allow for semantics-preserving transformations, necessary duplicate elimination operations have to be considered. Table 1 shows under which circumstances duplicate elimination (D) is needed.

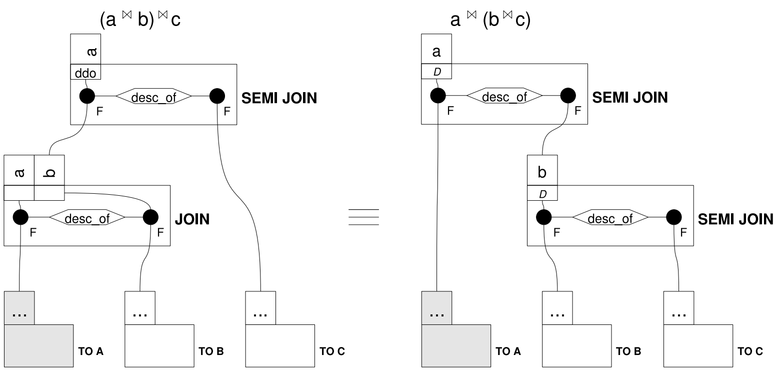
Table 1: Duplicate elimination depending on the output node

Output node	`child_of`, `following_sibling_of`	`parent_of`, `previous_sibling_of`	`anc_of`, `desc_of`, `following_of`, `previous_of`
A	D	–	D
B	–	D	D

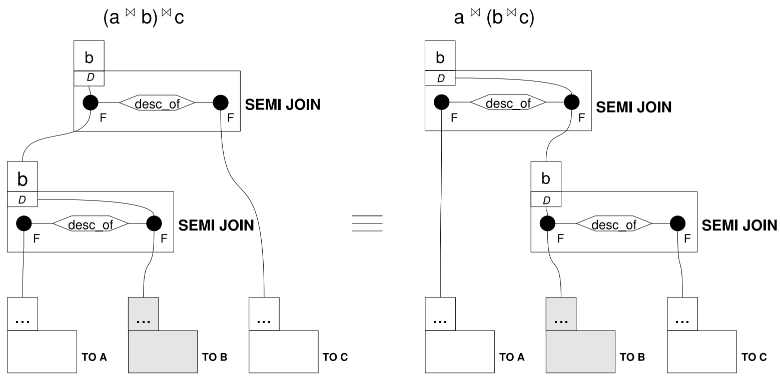
2.1 Only `descendant` axes

This section shows the associativity rules for two join operators which only evaluate `descendant axis`. These rules also hold for a combination of two join operators having structural predicates x, y with $x, y \in \{$`desc_of`, `anc_of`, `following_of`, `previous_of`$\}$ and $x = y$.

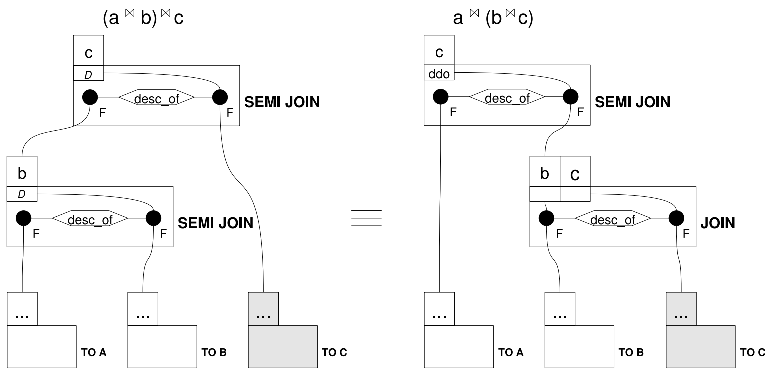
2.1.1 Output node a



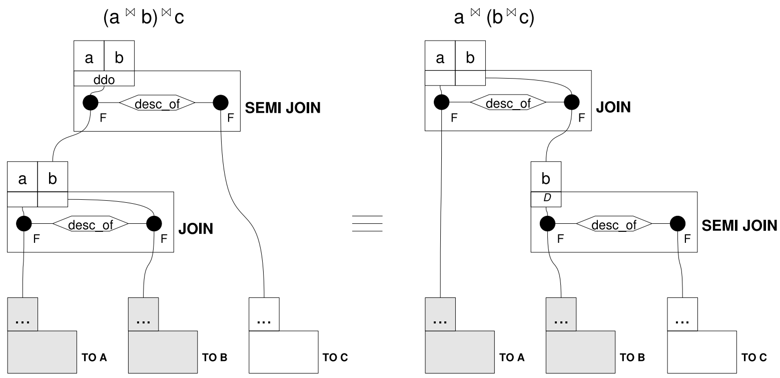
2.1.2 Output node b



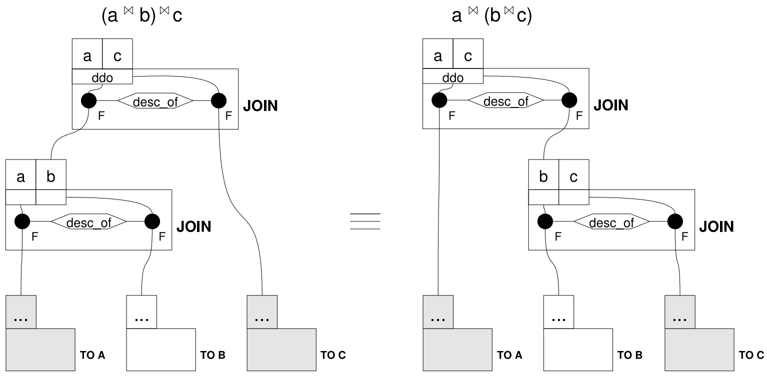
2.1.3 Output node c



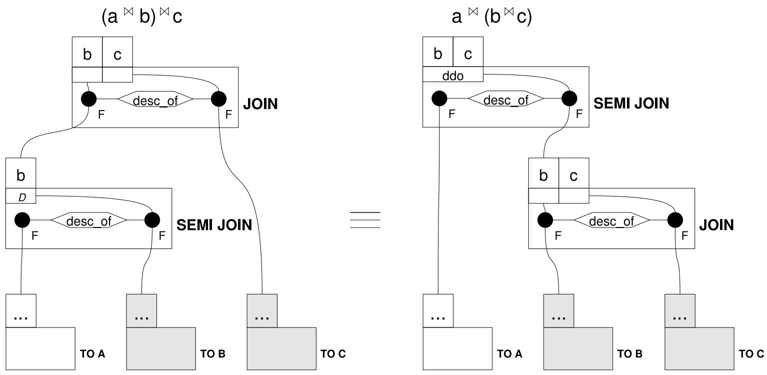
2.1.4 Output nodes a and b



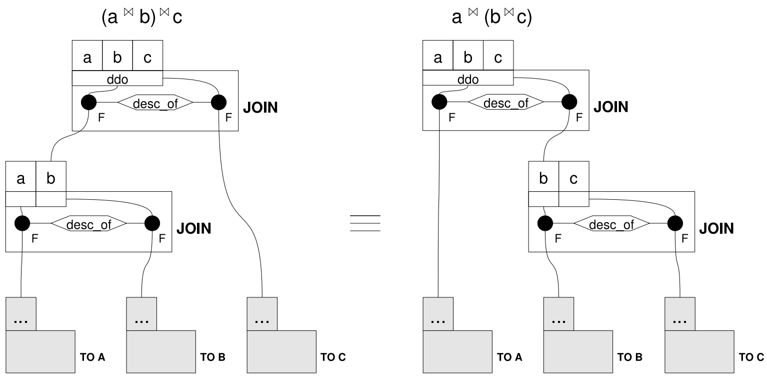
2.1.5 Output nodes a and c



2.1.6 Output nodes b and c



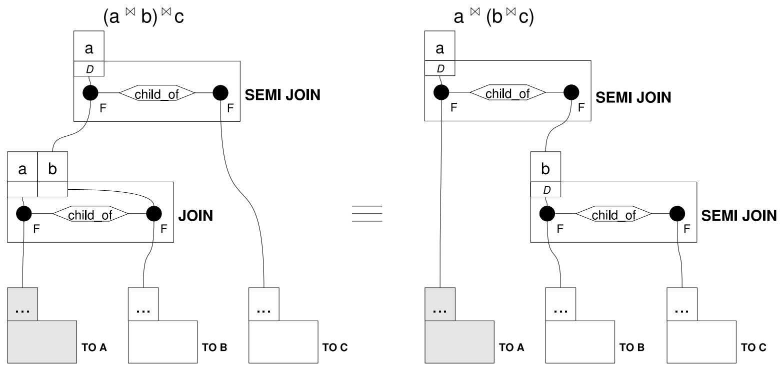
2.1.7 Output nodes a, b, and c



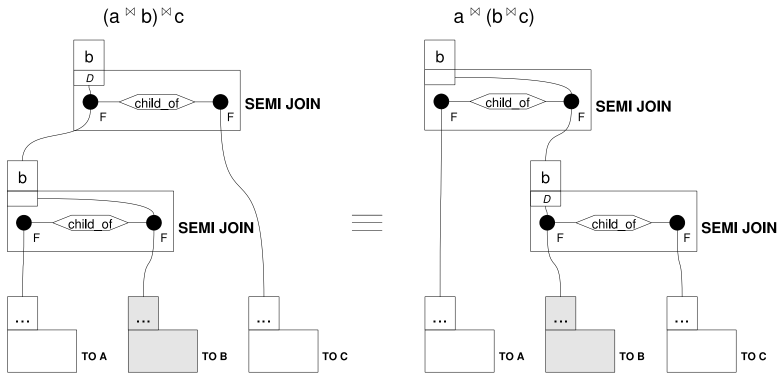
2.2 Only `child` axes

This section shows the associativity rules for two join operators which only evaluate `child` axes. These rules also hold for two join operators where x, y are structural predicates with $x, y \in \{$`child_of`, `following_sibling_of`$\}$ and $x = y$.

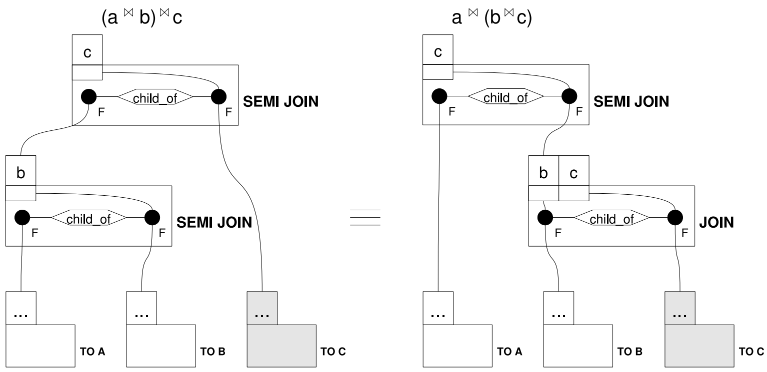
2.2.1 Output node a



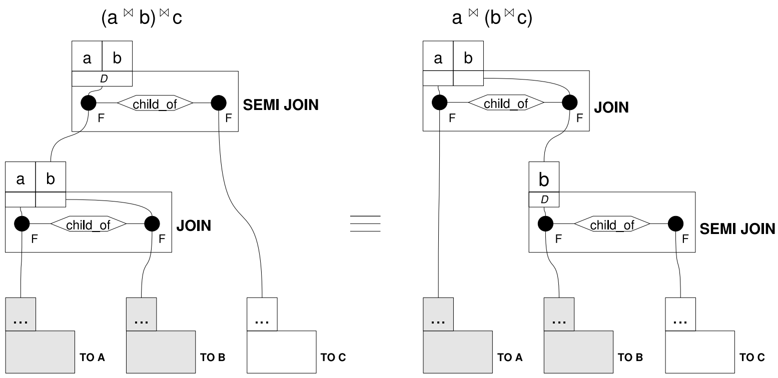
2.2.2 Output node b



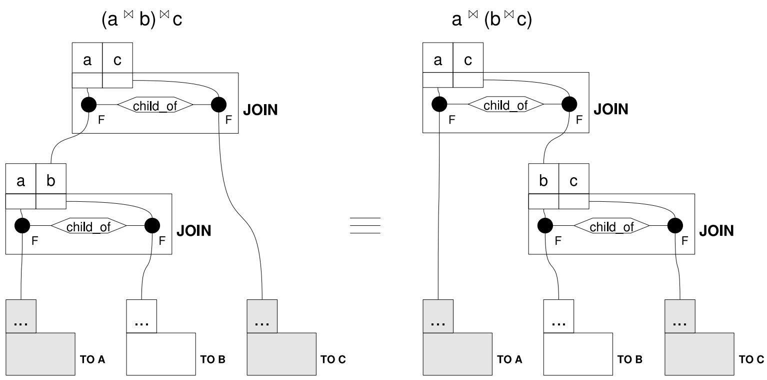
2.2.3 Output node c



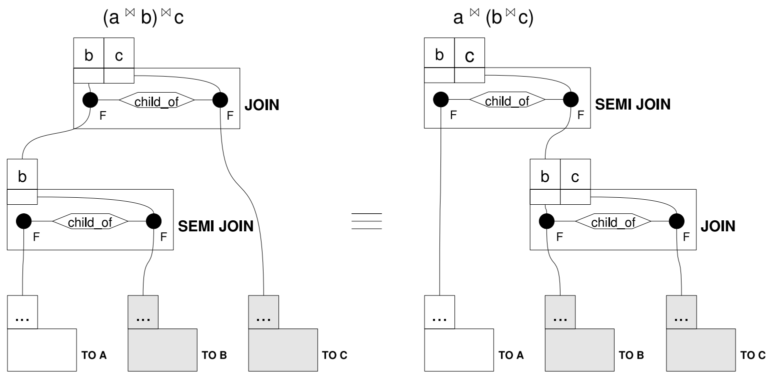
2.2.4 Output nodes a and b



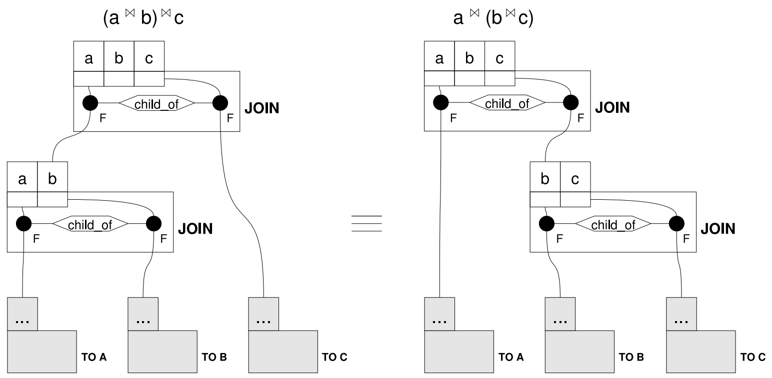
2.2.5 Output nodes a and c



2.2.6 Output nodes b and c



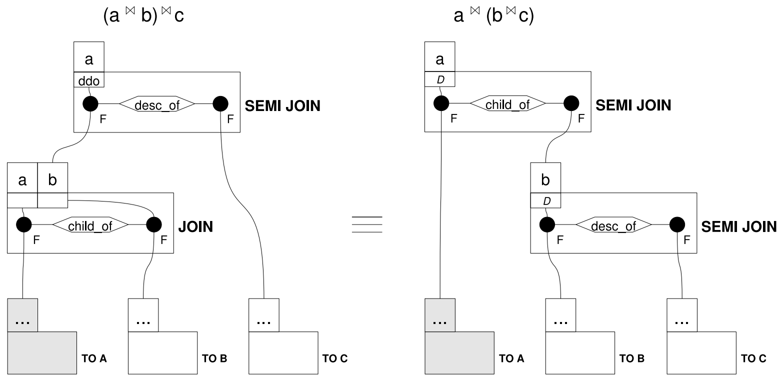
2.2.7 Output nodes a, b, and c



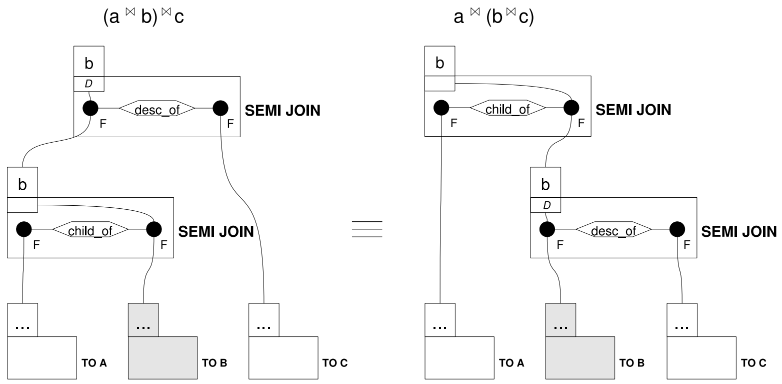
2.3 `Descendant` and `child` axes

This section shows the associativity rules for one join operator which evaluates a `descendant axis` and one operator which calculates a `child` axis. These rules also hold for a combination of two join operators having structural predicates x, y where $x \in \{\texttt{desc\_of}, \texttt{anc\_of}, \texttt{following\_of}, \texttt{previous\_of}\}$ and $y \in \{\texttt{child\_of}, \texttt{following\_sibling\_of}\}$

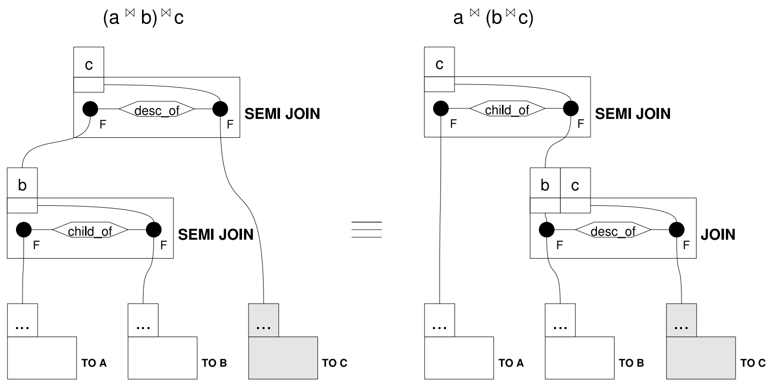
2.3.1 Output node a



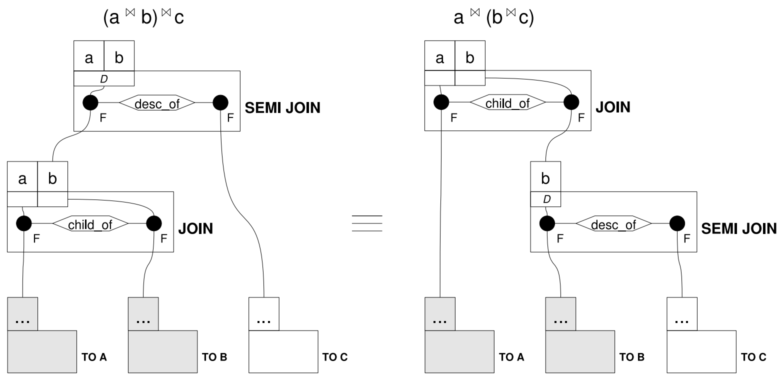
2.3.2 Output node b



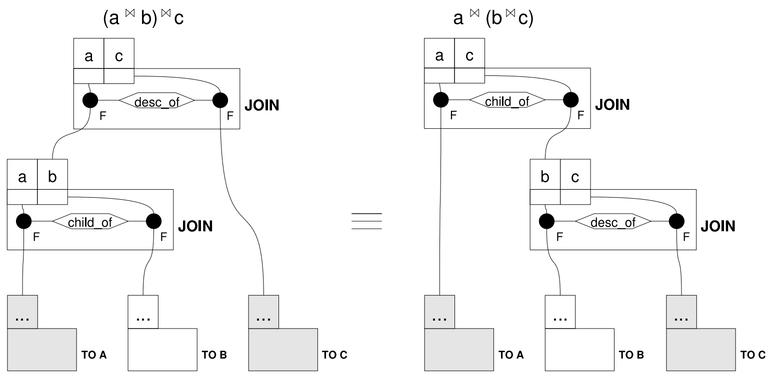
2.3.3 Output node c



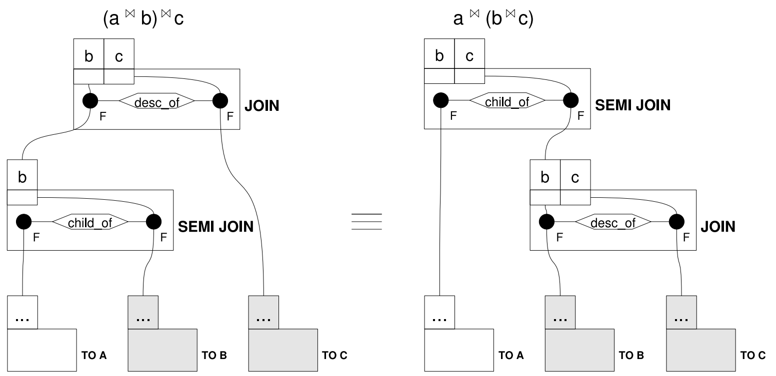
2.3.4 Output nodes a and b



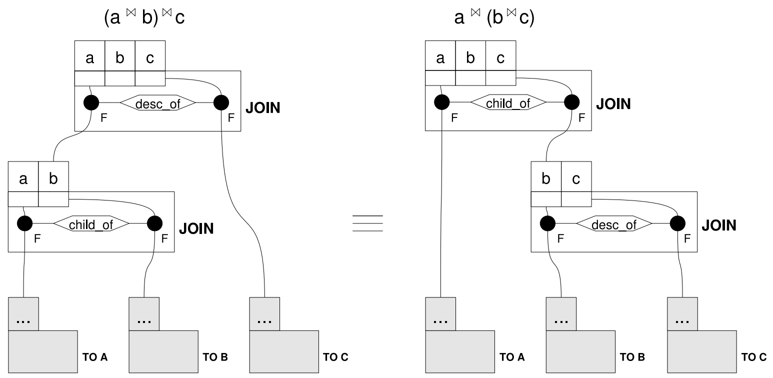
2.3.5 Output nodes a and c



2.3.6 Output nodes b and c



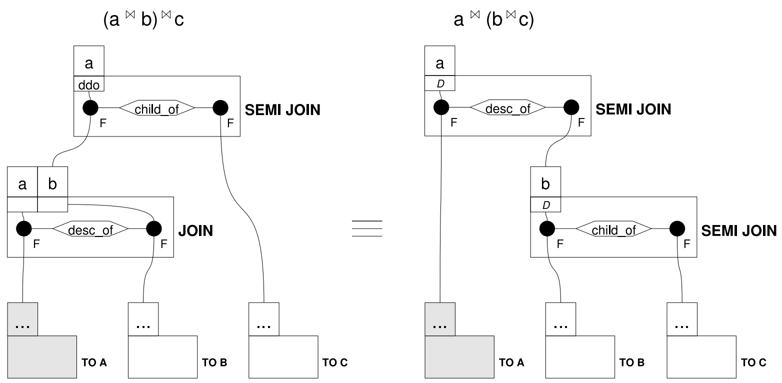
2.3.7 Output nodes a, b, and c



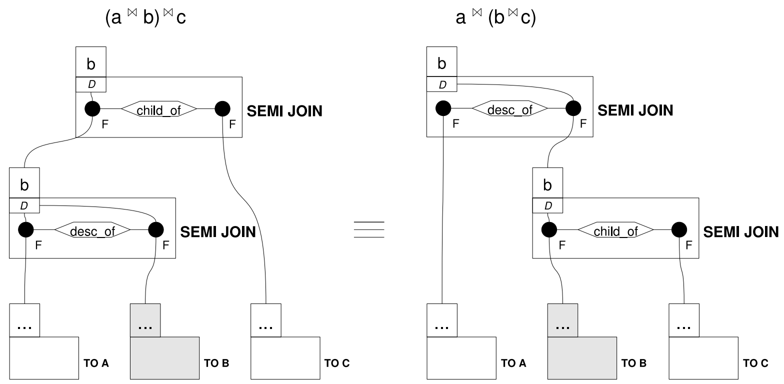
2.4 `Child` and `descendant` axes

This section shows the associativity rules for one join operator which evaluates a **`child axis`** and one operator which calculates a **`descendant`** axis. These rules also hold for a combination of two join operators having structural predicates x, y where $x \in \{\texttt{child\_of}, \texttt{following\_sibling\_of}\}$ and $y \in \{\texttt{desc\_of}, \texttt{anc\_of}, \texttt{following\_of}, \texttt{previous\_of}\}$.

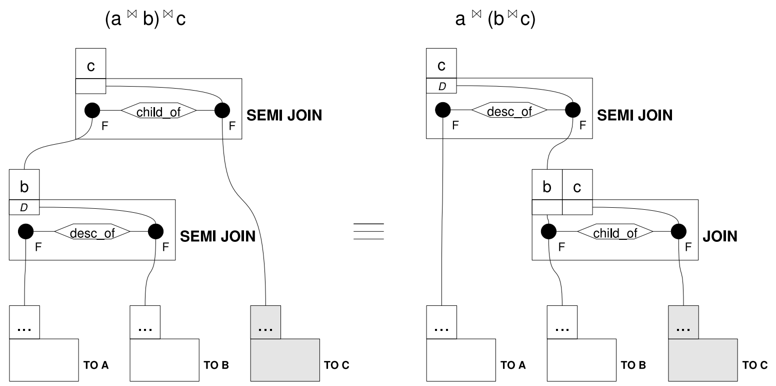
2.4.1 Output node a



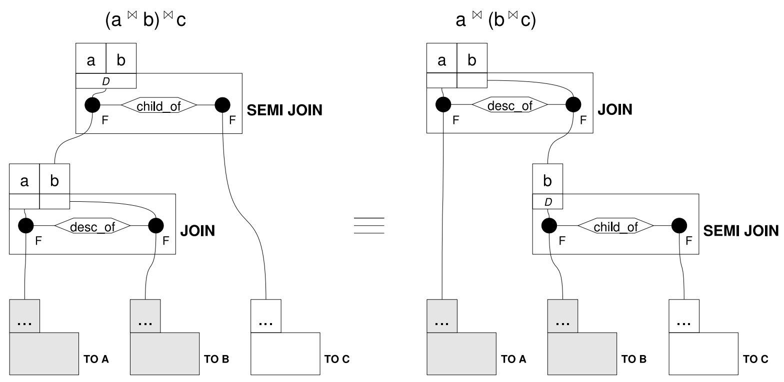
2.4.2 Output node b



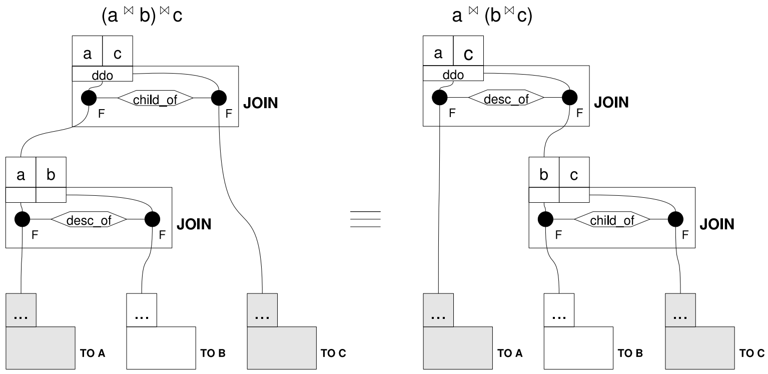
2.4.3 Output node c



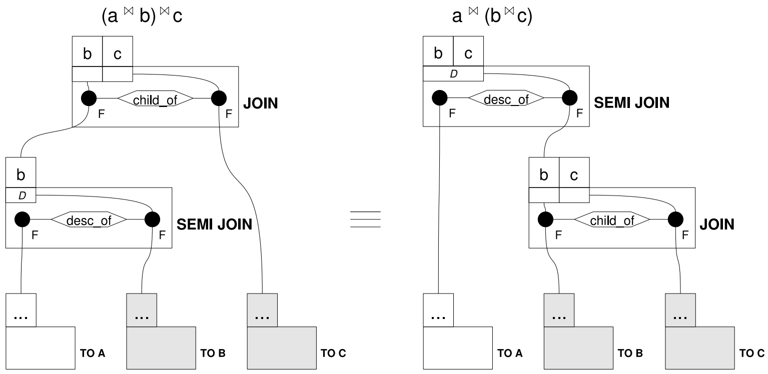
2.4.4 Output nodes a and b



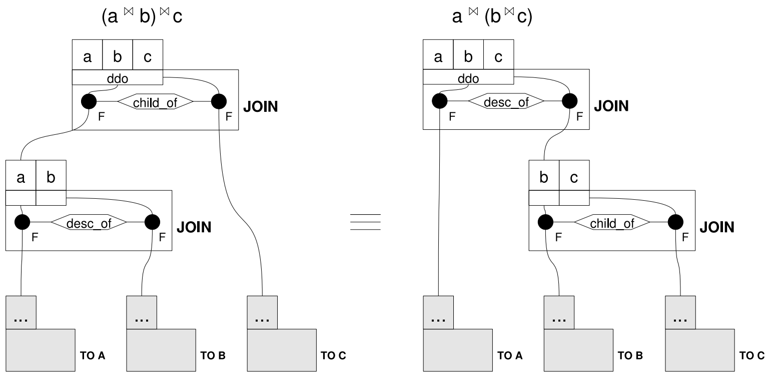
2.4.5 Output nodes a and c



2.4.6 Output nodes b and c



2.4.7 Output nodes a, b, and c



References

[WMH08] Andreas M. Weiner, Christian Mathis, and Theo Härder. Rules for Query Rewrite in Native XML Databases. In *Proceedings of the EDBT Workshops, Third International Workshop on Database Technologies for Handling XML Information on the Web (DataX 2008), March 25, 2008, Nantes, France*, 2008.