Relative Expressive Power of Downward Fragments of
Navigational Query Languages on Trees and Chains

Jelle Hellings¹  Marc Gyssens¹  Yuqing Wu²  Dirk Van Gucht³
Jan Van den Bussche¹  Stijn Vansummeren⁴  George H. L. Fletcher⁵

¹Hasselt University and Transnational University of Limburg, Belgium
²Pomona College, California, USA
³Indiana University, Indiana, USA
⁴Université Libre de Bruxelles, Belgium
⁵Eindhoven University of Technology, the Netherlands
{jelle.hellings, marc.gyssens, jan.vandenbussche}@uhasselt.be
melanie.wu@pomona.edu, vgucht@cs.indiana.edu
stijn.vansummeren@ulb.ac.be, g.h.l.fletcher@tue.nl

Abstract
Motivated by the continuing interest in the tree data model, we study the expressive power of downward fragments of navigational query languages on trees. The basic navigational query language we consider expresses queries by building binary relations from the edge relations and the identity relation, using composition and union. We study the effects on the expressive power when we add transitive closure, projections, coprojections, intersection, and difference. We study expressiveness at the level of boolean queries and path queries, on labeled and unlabeled trees, and on labeled and unlabeled chains. In all these cases, we are able to present the complete Hasse diagram of relative expressiveness. In particular, we were able to decide, for each fragment of the navigational query languages that we study, whether it is closed under difference and intersection when applied on trees.

Categories and Subject Descriptors H.2.3 [Database Management]: Languages—Query languages; F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages—Classes defined by grammars or automata

General Terms Theory

Keywords Tree Queries, Expressive Power, Automata

1. Introduction
Many relations between data can be described in a hierarchical way, including taxonomies such as the taxonomy of species studied by biologists, corporate hierarchies, and file and directory structures. A logical step is to represent these data using a tree-based data model. It is therefore not surprising that tree-based data models were among the first used in commercial database applications, the prime example being the hierarchical data model that is used since the 1960s [25]. Since the 1970s, other data models, such as the relational data model [6], almost completely replaced the hierarchical data model. Interest in tree-based data models revived in the 1990s by the introduction of the XML data model [4], which allowed for unstructured and semi-structured tree data, and, more recently, by JSON, as used by several NoSQL products [7].

Observe that tree-based data models are special cases of graph-based data models. In practice, query languages for trees and graphs usually rely on navigating the structure to find the data of interest. Examples of this focus on navigation can be found in XPath [2, 5, 19, 23], SPARQL [16, 21], and the regular path queries (RPQs) [1]. The core navigational power of these query languages can be captured by various fragments of the calculus of relations, popularized by Tarski, extended with transitive closure [13, 22]. In the form of the navigational query languages of Fletcher et al. [8], the relative expressive power of these fragments have been studied in full detail on graph-structured data [9–11]. Much less is known for the more restrictive tree data model, however: most of the results on graphs do not directly apply to trees. There are expressiveness results for several XPath fragments [3, 12, 15, 19, 20, 23, 24, 26] in the context of XML, but these results do not provide a complete picture of the relative expressive power of the various fragments of the navigational query languages we consider here. As a first step towards this aim, we study the expressive power of the downward fragments: these are the navigational query languages that only allow downward navigations in the tree via parent-child relations.

The smallest fragment of the downward navigational query languages we consider can express queries by building binary relations from the edge relations and the identity relation (id), using composition (◦) and union (∪). We study the effect on the expressive power of adding transitive closure (+); projections (π)—which can be used to express conditions similar to the node-expressions in XPath [19] and the branching operator in nested RPQs [1]; coprojections (π)—which can be used to express negated conditions; intersection (∩); and difference (\). We study expressiveness for both path queries, which evaluate to a set of node pairs, and boolean queries, which evaluate to true or false. We do not only consider labeled trees, but also unlabeled trees and labeled and unlabeled chains, the reason being that most query languages are easier to an-
alyze on these simpler structures and inexpressiveness results obtained on them can then be bootstrapped to the more general case.

For all the cases we consider, we are able to present the complete Hasse diagram of relative expressiveness; these Hasse diagrams are summarized in Figure 1. In several cases, we were able to argue that pairs of downward fragments of the navigational query languages that are not equivalent in expressive power when used to query graphs, are already not equivalent in expressive power on the simplest of graphs: labeled or unlabeled chains. Hence, for these languages, we actually strengthen the results of Fletcher et al. [8].

In the cases where graphs and trees yield different expressiveness results, we were able to prove collapse results. In particular, we were able to decide, for each fragment of the navigational query languages that we study, whether it is closed under difference and intersection when applied on trees; adding intersection to a downward fragment of the navigational query languages does not change the expressive power, and adding difference only adds expressive power when \( \pi \) is present and \( \overline{\pi} \) is not present, in which case difference only adds the ability to express \( \overline{\pi} \). To prove these closure results, we develop a novel technique based on finite automata [17], which we adapt to a setting with conditions. We use these condition automata to represent and manipulate navigational queries, with the goal to replace \( \cap \) and \( \backslash \) operations. We also use these condition automata to show that, in the boolean case, \( \pi \) never adds expressive power when querying labeled chains. Finally, using homomorphism-based techniques, we show that, in the boolean case on unlabeled trees and unlabeled chains, only fragments with the non-monotone operator \( \overline{\pi} \) can express queries that are not equivalent to queries of the form the height of the tree is at least \( k \).

Our study of the relative expressivity power of the downward fragments of the navigational query languages on trees also has practical ramifications. If, for example, two language fragments are equivalent, then this leads to a choice in query language design: choosing, on the one hand, a smaller set of operators that, due to its simplicity, is easier to implement and optimize, even when dealing with big data in a distributed setting or when using specialized hardware, or, on the other hand, a bigger set of operators that allows for easier querying by the end users. Indeed, if one is only interested in boolean queries on unlabeled trees, then RPOs are much harder to evaluate than queries of the form the height of the tree is at least \( k \), although our results indicate that these query languages are, in this case, equivalent. Moreover, all our collapse results are constructive: we present ways to rewrite queries using operators such as \( \cap \) and \( \backslash \) into queries that do not rely on these operators. Hence, our results can be used as a starting point for automatic query rewriting and optimization techniques that, depending on the hardware, the data size, and the data type, choose an appropriate query evaluation approach.

**Organization** In Section 2, we present the basic notions used throughout this paper. In Section 3, we present all relevant results on the relative expressivity power of the downward navigational query languages, resulting in the Hasse diagrams of relative expressiveness visualized in Figure 1. In Section 4, we discuss related work. In Section 5, we summarize our findings and propose directions for future work.

### 2. Preliminaries

A graph is a pair \( \mathcal{G} = (\mathcal{N}, \mathcal{E}) \), with \( \mathcal{N} \) a finite set of nodes and \( \mathcal{E} \) a set of binary edge relations. Each edge relation \( \ell \in \mathcal{E} \) is a subset of \( \mathcal{N} \times \mathcal{N} \) and is interpreted as those edges labeled with \( \ell \). A graph is unlabeled if \( |\mathcal{E}| = 1 \). On unlabeled graphs, we use \( E \) to refer to the single edge relation. A tree \( T = (\mathcal{N}, \mathcal{E}) \) is an acyclic graph in which exactly one node, the root, has no incoming edges, and all other nodes have exactly one incoming edge. In an edge \((m,n)\), node \( m \) is the parent of node \( n \) and node \( n \) is a child of node \( m \). A chain is a tree in which all nodes have at most one child. A path in a graph \( \mathcal{G} = (\mathcal{N}, \mathcal{E}) \) is a sequence \( n_1 \ell_1 n_2 \ldots \ell_{i-1} n_i \), with \( n_1, \ldots, n_i \in \mathcal{N} \), \( \ell_1, \ldots, \ell_{i-1} \in \mathcal{E} \), and, for all \( 1 \leq j < i \), \((n_j, n_{j+1}) \in \ell_j \).

**Definition 1** (Navigational expressions syntax). The navigational expressions over graphs are defined by the grammar

\[
e \in \{\emptyset, \text{id}, \ell, e \circ e, e \cup e, e^\pi \mid \{e\} \} \quad | \quad [e]^- \mid [e]^+ \mid \pi_1[e] \mid \pi_2[e] \mid \overline{\pi_1}[e] \mid \overline{\pi_2}[e] \mid e \cap e \mid e \backslash e.
\]

We also use the following shorthand notation

\[
[e]^+ \equiv [e] \cup \text{id}.
\]

**Definition 2** (Navigational expressions semantics). Let \( \mathcal{G} = (\mathcal{N}, \mathcal{E}) \) be a graph and let \( e \) be a navigational expression. We write \( e(\mathcal{G}) \) to denote the evaluation of navigational expression \( e \) on graph \( \mathcal{G} \). The semantics of navigational expressions is defined as follows:

\[
\emptyset(\mathcal{G}) = \emptyset;
\]

\[
\text{id}(\mathcal{G}) = \{(m, m) \mid m \in \mathcal{N}\};
\]

\[
\ell(\mathcal{G}) = \{m, n \mid \ell(m, n) \in \mathcal{E}\};
\]

\[
\ell(\mathcal{G}) = \emptyset \quad (for \ ell an edge-label with \ \ell \notin \mathcal{E});
\]

\[
e_1 \circ e_2(\mathcal{G}) = e_1(\mathcal{G}) \circ e_2(\mathcal{G});
\]

\[
e_1 \cup e_2(\mathcal{G}) = e_1(\mathcal{G}) \cup e_2(\mathcal{G});
\]

\[
[e]^-(\mathcal{G}) = e(\mathcal{G})^\pi;
\]

\[
[e]^+(\mathcal{G}) = e(\mathcal{G})^\pi;
\]

\[
\pi_1[e](\mathcal{G}) = \{(m, m) \mid \exists n (m, n) \in e(\mathcal{G})\};
\]

\[
\pi_2[e](\mathcal{G}) = \{(m, m) \mid \exists n (m, n) \in e(\mathcal{G})\};
\]

\[
\overline{\pi_1}[e](\mathcal{G}) = \{(m, m) \mid \neg \exists n (m, n) \in e(\mathcal{G})\};
\]

\[
\overline{\pi_2}[e](\mathcal{G}) = \{(m, m) \mid \neg \exists n (m, n) \in e(\mathcal{G})\};
\]

\[
e_1 \cap e_2(\mathcal{G}) = e_1(\mathcal{G}) \cap e_2(\mathcal{G});
\]

\[
e_1 \cup e_2(\mathcal{G}) = e_1(\mathcal{G}) \cup e_2(\mathcal{G}).
\]

In the above, \( R_1 \circ R_2 \), for binary relations \( R_1, R_2 \subseteq \mathcal{N} \times \mathcal{N} \), is defined by \( R_1 \circ R_2 \equiv \{(m, n) \mid \exists z ((m, z) \in R_1) \land ((z, n) \in R_2)\}\), \( R^{-1} \), for a binary relation \( R \subseteq \mathcal{N} \times \mathcal{N} \), is defined by \( R^{-1} \equiv \{(m, n) \mid (m, n) \in R\}\), and \( R^+ \), for a binary relation \( R \subseteq \mathcal{N} \times \mathcal{N} \), is defined by \( R^+ = \bigcup_{k \in \mathbb{N}} R^k \), in which we define

\[
R^k = \begin{cases} R & \text{if } k = 1; \\ R \circ R^{k-1} & \text{if } k > 1. \end{cases}
\]

In the sequel, we shall also use this last notation for navigational expressions.

In this work, we only study downward navigational expressions. Thus we do not consider diversity (di) and the converse operation (−) in this paper.

**Definition 3** (Query semantics). Let \( e \) be a navigational expression and let \( \mathcal{G} = (\mathcal{N}, \mathcal{E}) \) be a graph. Using path query semantics, \( e \) on \( \mathcal{G} \) evaluates to \( e(\mathcal{G}) \), and using boolean query semantics, \( e \) on \( \mathcal{G} \) evaluates to the truth value of \( e(\mathcal{G}) \neq \emptyset \).

**Example 1.** Consider the class-structure of a program described by relations subclass and method. In this setting, the query \( \text{subclass}^+ \) returns the relation between classes and their descendant classes, the query \( \pi_1[\text{method}] \) returns all classes that do not define own methods, and the query \( \pi_1[\text{method}] \cap \pi_1[\text{subclass}^+ \circ \text{method}] \)
returns all classes that define methods, while having no descendants that also define methods.

We study the relative expressive power, both for path queries and boolean queries, of the various classes of the navigational expressions obtained by allowing or disallowing certain parts of the grammar of Definition 1. We usually refer to the obtained expressions obtained by allowing or disallowing certain parts of the grammar of Definition 1.

<table>
<thead>
<tr>
<th>Chains</th>
<th>Boolean queries</th>
<th>Trees</th>
<th>Path queries</th>
<th>Chains and Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N(\pi))</td>
<td>(N(+, \pi))</td>
<td>(N(\pi, \pi))</td>
<td>(N(\pi))</td>
<td>(N(\pi, +))</td>
</tr>
<tr>
<td>(N(\pi, l))</td>
<td>(N(+, \pi, l))</td>
<td>(N(\pi, +, l))</td>
<td>(N(\pi, +, \pi))</td>
<td>(N(\pi, +, \pi, l))</td>
</tr>
</tbody>
</table>

![Figure 1](image1.png)

Figure 1. The full Hasse diagrams describing the relations between the expressive power of the various fragments of \(N(+, \pi, \pi, \\setminus, \cap)\). An edge \(A \rightarrow B\) indicates \(A \preceq B\) and \(B \not\preceq A\).

graphs to more general graphs (following the arrows), while equivalence results carry over from general graphs to more specialized graphs (following the arrows backwards).

The subsumption results for the navigational query languages on graphs (both labeled and unlabeled) have already been studied in full detail [8–11]. We summarize the results that are relevant to us in this paper.

First, observe the following identities:

\[
\begin{align*}
\pi_1[e] & \equiv \pi_1[\pi_1[e]]; \\
\pi_2[e] & \equiv \pi_2[\pi_2[e]]; \\
\pi_1[e] & \equiv \text{id} \setminus \pi_1[e]; \\
\pi_2[e] & \equiv \text{id} \setminus \pi_2[e]; \\
e_1 \cap e_2 & \equiv e_1 \setminus (e_1 \setminus e_2).
\end{align*}
\]

Let \(\triangledown \subseteq \{+, \pi, \pi, \cap, \\setminus\}\). We define \(\triangledown\) to be the superset of \(\triangledown\) obtained by adding all operators that can be expressed indirectly in \(N(\triangledown)\) by using the above identities.

**Proposition 1** (Fletcher et al. [8–11]). Let \(\triangledown_1, \triangledown_2 \subseteq \{+, \pi, \pi, \cap, \\setminus\}\). On labeled graphs, we have \(N(\triangledown_1) \subseteq_0 N(\triangledown_2)\) and \(N(\triangledown_1) \subseteq_0 N(\triangledown_2)\) if and only if \(\triangledown_1 \subseteq \triangledown_2\). On unlabeled graphs, we have \(N(\triangledown_1) \subseteq_0 N(\triangledown_2)\) if and only if \(\triangledown_1 \subseteq \triangledown_2\) and we have \(N(\triangledown_1) \subseteq_0 N(\triangledown_2)\) if and only if \(\triangledown_1 \subseteq \triangledown_2\) or \(\triangledown_1 \subseteq \{\pi\}\) and \(\triangledown_2 = \triangledown_1 \cup \{+\}\).
3. Expressive Power

In this Section, we present all relevant results on the relative expressive power of the downward navigational query languages, resulting in the Hasse diagrams of relative expressiveness visualized in Figure 1. We divide our results in four categories, based on the techniques used to prove them. Section 3.1 is dedicated to showing that $\pi$ is a primitive operator for path queries. Section 3.2 provides all the results obtained using homomorphisms. These include a major collapse result for boolean queries on unlabeled trees and unlabeled chains. Section 3.3 introduces condition automata and uses condition automata to obtain closure results for difference and intersection. Furthermore, condition automata are used to show that $\pi$ does not add expressive power for boolean queries on labeled trees. Section 3.4 uses the relation between the navigational query languages and first-order logic to prove some expressiveness results involving $+$. Combined, all these results prove the following:

**Theorem 1.** Let $\mathfrak{g}_1, \mathfrak{g}_2 \subseteq \left\{+, \pi, \cap, \setminus, \setminus \right\}$. We have $N(\mathfrak{g}_1) \subseteq N(\mathfrak{g}_2)$, respectively $N(\mathfrak{g}_1) \subseteq N(\mathfrak{g}_2)$, on unlabeled chains, respectively labeled chains, unlabeled trees, or labeled trees if and only if there exists a directed path from $\mathfrak{g}_1$ to $\mathfrak{g}_2$ in the corresponding Hasse diagram of Figure 1.

3.1 Identity-based Results

In this Section, we show that $\pi$ is a primitive operator for path queries, meaning that adding $\pi$ always increases expressive power (unless $\pi$ is already added, in which case $\pi$ can be expressed by straightforward rewriting).

**Lemma 1.** Let $C = (N, E)$ be an unlabeled chain, let $\mathfrak{g} \subseteq \left\{+, \land, \setminus \right\}$, and let $e$ be a navigational expression in $N(\mathfrak{g})$. We have $e(C) \cap id(C) = \emptyset$ or $e(C) \cap id(C) = id(C)$.

**Proposition 2.** Let $\mathfrak{g} \subseteq \left\{+, \setminus, \land \right\}$. On unlabeled chains we have $N(\pi) \notin N(\mathfrak{g})$.

**Proof.** Let $C = (N, E)$ be a chain with $2 \leq |N|$. We have $\emptyset \subseteq \pi_1[|E|](C) \subseteq \pi_1[|E|] \cap id(C) \subseteq id(C)$. By Lemma 1, no expression in $N(\mathfrak{g})$ can be path-equivalent with $\pi_1[|E|]$.

3.2 Homomorphism-based Results

Homomorphisms are often used to prove properties of query languages, including expressiveness results [8]. We use homomorphisms to show several results, including a major collapse result for boolean queries on unlabeled trees and unlabeled chains: we show that only fragments with the non-monotone operator $\pi$ can express queries that are not equivalent to queries of the form the height of the tree is at least $k$.

**Definition 4 (Homomorphism).** Let $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$ be graphs. We say that a mapping $h : N_1 \to N_2$ is a homomorphism from $G_1$ to $G_2$ if, for every pair of nodes $m, n \in N_1$ and every edge-label $\ell$, we have that $(m, n) \in \ell$ implies $(h(m), h(n)) \in \ell$.

It is straightforward to prove that the navigational query languages without the non-monotone operators $\pi$ and $\setminus$ are closed under homomorphisms.

**Lemma 2.** Let $\mathfrak{g} \subseteq \left\{+, \pi, \cap, \setminus \right\}$. The language $N(\mathfrak{g})$ is closed under homomorphisms: for every navigational expression $e$ in $N(\mathfrak{g})$, every pair of graphs $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$, and every homomorphism $h$ of $G_1$ to $G_2$, we have that $(m, n) \in e(G_1)$ implies $(h(m), h(n)) \in e(G_2)$.

$\pi$ is a navigational expression $e$ is monotone if, for every graph $G$ and every graph $G'$ obtained from adding nodes or edges to $G$, we have $e(G) \subseteq e(G')$.

Let $T = (N, E)$ be a tree. If $\tau \in N$ is the root of $T$, then the depth of node $n \in N$ is the length of the directed path, counted in the number of nodes on the path, from $\tau$ to $n$. The depth of tree $T$, denoted by depth$(T)$, is the maximum depth of any node $m \in N$. Hence, if $T$ is a chain, then depth$(T) = |N|$.

**Lemma 3.** Let $C_1 = (N_1, E_1)$ and $C_2 = (N_2, E_2)$ be unlabeled chains. If $|N_1| \leq |N_2|$, then there exists a homomorphism of $C_1$ to $C_2$. Let $T = (N_T, E_T)$ be an unlabeled tree and let $C = (N_C, E_C)$ be an unlabeled chain. If $|N_C| \geq$ depth$(T)$, then there exists a homomorphism of $T$ to $C$, and if $|N_C| \leq$ depth$(T)$, then there exists a homomorphism of $C$ to $T$.

We use these homomorphisms between unlabeled chains and unlabeled trees to show that only fragments that can express the non-monotone operator $\pi$, can express boolean queries on unlabeled trees that are different from queries of the form the height of the tree is at least $k$.

**Theorem 2.** Let $\mathfrak{g} \subseteq \left\{+, \pi, \cap \right\}$. On unlabeled trees we have $N(\mathfrak{g}) \subseteq N(\pi)$.

**Proof.** Let $T = (N, E)$ be a tree and let $C' = (N', E')$ be a chain with depth$(T) = |N'|$. By Lemma 3, there exists a homomorphism $h : N \to N'$ of $T$ to $C'$ and a homomorphism $h' : N' \to N$ of $C'$ to $T$. Hence, by Lemma 2, we have, for every navigational expression $e'$ in $N(\pi)$, $e(T) \neq \emptyset$ if and only if $e(C') \neq \emptyset$. As a consequence, we can conclude that no navigational expression in $N(\mathfrak{g})$ can distinguish between trees and chains of equal depth.

Let $C = (N, E)$ and $C' = (N', E')$ be chains such that $|N| < |N'|$. By Lemma 3, there exists a homomorphism $h : N \to N'$ of $C$ to $C'$. Hence, by Lemma 2, we have, for every navigational expression $e'$ in $N(\mathfrak{g})$, $e(C') \neq \emptyset$ if and only if $e(C') \neq \emptyset$. As a consequence, we can conclude that when a navigational expression holds on a chain, then it also holds on every chain of greater depth.

Let $e$ be a navigational expression in $N(\mathfrak{g})$. We choose $C = (N, E)$ to be the chain with minimum depth such that $e(C) \neq \emptyset$. If no such chain exists, then, by the two properties shown above, $e$ is boolean-equivalent to $\emptyset$. Since $C$ is also the chain with minimum depth such that $e^k(C) \neq \emptyset$, with $k = |N| - 1$, we may conclude, by the two properties shown above, that $e^k$ and $e$ are boolean-equivalent.

From a closer examination of the previous proof, the following result readily follows.

**Corollary 1.** Let $\mathfrak{g} \subseteq \left\{+, \pi, \cap \right\}$. On unlabeled trees $N(\mathfrak{g})$ can only express queries that are boolean-equivalent to either $\emptyset$ or navigational expressions of the form $e^k$.

**Corollary 2.** Let $\mathfrak{g} \subseteq \left\{+, \pi, \cap \right\}$. On unlabeled chains we have $N(\pi) \notin N(\mathfrak{g})$.

**Proof (sketch).** Let $e$ be the expression $\pi_2[|E|] \circ E \circ \pi_1[|E|]$. If a chain $C$ has depth two, then we have $e(C) \neq \emptyset$. For all chains $C'$ with a depth other than two, we have $e(C') = \emptyset$. In Corollary 1 we already showed that every expression in $N(\mathfrak{g})$ is boolean-equivalent to $\emptyset$ or $e^k$, which are both boolean-inequivalent to $e$.

Besides the above results, we use homomorphisms to show that $N(\pi)$ and $N(+)$ cannot properly distinguish between labeled trees and labeled chains. This result is then used to show that, on labeled tree, $N(\pi)$ and $N(+)$ cannot express all boolean queries expressible by $N(\pi)$.

**Lemma 4.** Let $\mathfrak{g} \subseteq \left\{+ \right\}$. Let $e$ be a navigational expression in $N(\mathfrak{g})$. If $e$ does not utilize the operator $\emptyset$, then there exists a labeled chain $C$ such that $e(C) \neq \emptyset$. 
Proposition 3. Let $\mathfrak{g} \subseteq \{+\}$. On labeled trees we have $\mathcal{N}(\pi) \subseteq \mathcal{N}(\mathfrak{g})$.

Proof. Let $\ell_1$ and $\ell_2$ be two edge-labels and let $e = \pi_1[\ell_1] \circ \pi_2[\ell_2]$ be a navigational expression in $\mathcal{N}(\pi)$. On labeled trees, this expression evaluates to true only if a node has two distinct outgoing edges labeled with $\ell_1$ and $\ell_2$, respectively. Hence, for all chains $C = (N, E)$, we have $e(C) = \emptyset$. As $e$ never evaluates to true on chains, we use Lemma 4 to conclude that no navigational expression in $\mathcal{N}(\mathfrak{g})$ is boolean-equivalent to $e$.

3.3 Automaton-based Results

Observe that $\mathcal{N}(+)$ and the regular path queries [1] are equivalent: queries in these query languages select pairs of nodes $m,n$ such that there is a directed path from $m$ to $n$ whose labeling satisfies some regular expression. In the case of trees, there is a unique path from $m$ to $n$ which yields a strong relation between $\mathcal{N}(+)$ and regular expressions. As a consequence, we can adapt results from automata theory [17] in a straightforward way to prove that $\mathcal{N}(+ \cap \mathfrak{g}) \subseteq \mathcal{N}(\mathfrak{g})$.

Example 2. We can, for example, rewrite the navigational expression $[\ell_1] + \{\ell_2\}$ path-equivalent navigational expression that do not use $\mathfrak{g}$ or $\{\}$:

$[\ell_1] + \{\ell_2\} \equiv [\ell_1]^+ \cap \{\ell_2\}^+ \cup \{\ell_1\} \circ \{[\ell_2]^2\}$

We extend these automata-based techniques to the languages $\mathcal{N}(\mathfrak{g})$ with $\mathfrak{g} \subseteq \{+, \pi, \tau\}$ by introducing conditions on automaton states. We use these extended automata to prove that $\mathcal{N}(\mathfrak{g})$ is closed under $\cap$ and $\mathcal{N}(\mathfrak{g})$ is closed under $\\}$.

Definition 5 (Condition). A navigational expression $e$ is a condition if, for every labeled tree $T$, $e(T) \subseteq \id(T)$.

Here, we only consider conditions of the form $\emptyset$, $\id$, $\pi_1[e']$, $\pi_2[e']$, or $\pi_3[e']$, with $e'$ a navigational expression.

Definition 6 (Condition automaton). A condition automaton is a 7-tuple $A = (S, \Sigma, C, I, F, \delta, \gamma)$, where $S$ is a set of states, $\Sigma$ a set of transition labels, $C$ a set of condition expressions, $I \subseteq S$ a set of initial states, $F \subseteq S$ a set of final states, $\delta \subseteq S \times (\Sigma \cup \{\id\}) \times S$ the transition relation, and $\gamma \subseteq S \times C$ the state-condition relation. We denote $\gamma(q) = \{c \in C \mid q \in \gamma\}$.

Let $\mathfrak{g} \subseteq \{+, \pi, \tau\}$. We say that $A$ is $\mathfrak{g}$-free if every condition in $C$ is a navigational expression in $\mathcal{N}(\mathfrak{g}) \setminus \\}$, and we say that $A$ is log-free if the transition relation $\delta$ of $A$ is acyclic, and we say that $A$ is id-transition-free if $\delta \subseteq S \times \Sigma \times S$.

Example 3. Consider the condition automaton $A = (S, \Sigma, C, I, F, \delta, \gamma)$ with

$S = \{q_1, q_2, q_3, q_4\}$;

$\Sigma = \{\ell_1, \ell_2, \ell_3\}$;

$C = \{\id, \pi_1[\ell_2^2], \pi_2[\ell_3^2]\}$;

$I = \{q_1, q_4\}$;

$F = \{q_3, q_4\}$;

$\delta = \{(q_1, \ell_1, q_2), (q_1, \ell_3, q_4), (q_2, \ell_2, q_2), (q_2, \ell_2, q_3)\}$;

and $\gamma = \{(q_1, \id), (q_3, q_2, \pi_1[\ell_2^2]), (q_2, q_3, \pi_2[\ell_3^2])\}$.

This has been visualized in Figure 3. Using this visualization, it is easy to verify that the condition automaton is not loop-free, is $(\pi, +)$-free, and is id-transition-free.

Definition 7 (Semantics of condition automata). Let $G = (N, E)$ be a graph and let $A = (S, C, I, F, \delta, \gamma)$ be a condition automaton.
Table 2. Basic building blocks used by the translation from navigational expressions to condition automata. In the table, ℓ is an edge-label.

<table>
<thead>
<tr>
<th>ℓ</th>
<th>Condition automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, \emptyset, \emptyset) )</td>
</tr>
<tr>
<td>id</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, {v, id, w}, \emptyset) )</td>
</tr>
<tr>
<td>( e )</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, {v, e, w}, \emptyset) )</td>
</tr>
<tr>
<td>( \pi_1[e'] )</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, {v, \emptyset, \emptyset}) )</td>
</tr>
<tr>
<td>( \pi_2[e'] )</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, \emptyset, {v, \emptyset, \emptyset}) )</td>
</tr>
<tr>
<td>( \pi_1'[e'] )</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, \emptyset, {v, \emptyset, \emptyset}) )</td>
</tr>
<tr>
<td>( \pi_2'[e'] )</td>
<td>( A = ({v, w}, \Sigma, \emptyset, {v}, \emptyset, {v, \emptyset, \emptyset}) )</td>
</tr>
</tbody>
</table>

Proposition 5. Let \( \mathcal{X} \subseteq \{+, \pi, \pi\} \). The class of condition automata specified for \( N(\mathcal{X}) \) in Table 1 is path-equivalent with \( N(\mathcal{X}) \).

Proof (sketch). Let \( \Sigma \) be the set of all relevant edge-labels. Let \( e \) be a navigational expression in \( N(\mathcal{X}) \). We translate \( e \) to a condition automaton using structural induction. The base cases are described in Table 2. The inductive cases are expressions of the form \( e = e_1 \circ e_2 \), \( e = e_1 \cup e_2 \), or \( e = [e_1] \) with \( e_1 \) and \( e_2 \) navigational sub-expressions. For these cases, we use the constructions needed to prove Proposition 4.

Conversely, let \( A = (S, \Sigma, C, I, F, \delta, \gamma) \) be a condition automaton. Let \( v, w \notin S \) be two distinct fresh states. Let \( A' = (S \cup \{\ell, v, w\}, \Sigma, C, \{v\}, \{w\}, \delta \cup \delta_{v,w}, \gamma) \) with \( \delta_{v,w} = \{(v, id, q) | q \in F\} \cup \{(q, id, w) | q \in F\} \). The condition automaton that is path-equivalent to \( A \) and having only a single initial state and a single final state. We translate \( A' \) into a navigational expression using Algorithm 1.

Algorithm 1 From condition automaton to navigational expression

1. We mark each state \( q \in S \cup \{v, w\} : M[q] := \text{true} \).
2. We construct navigational expressions \( e_{q,r} \) between state \( q \in S \cup \{v, w\} \) and \( r \in S \cup \{v, w\} \) and initialize

   \[ e_{q,r} := \bigcup_{(q, r, c) \in \delta_{v, w}} (\bullet q) \circ c \circ (\bullet r), \]

   with \( e_{q,r} := \emptyset \) if there are no transitions between \( q \) and \( r \).
3. While \( \exists(q \in S) \land (M[q] = \text{true}) \) do
4. Choose \( q \) with \( (q \in S) \land (M[q] = \text{true}) \)
5. for \( p_1, p_2 \in S \cup \{v, w\} \) with \( q \notin \{p_1, p_2\} \) do
6. \( e_{p_1, p_2} := e_{p_1, p_2} \cup e_{p_1, q} \circ [e_{q, q}] \circ e_{p_2, q} \)
7. If applicable, remove \( \emptyset \) from \( e_{p_1, p_2} \) or reduce \( e_{p_1, p_2} \) to \( \emptyset \)
8. end for
9. Unmark state \( q : M[q] := \text{false} \)
10. end while
11. return \( e_{v, w} \)

The following invariants hold for Algorithm 1:

1. Every expression \( e_{q_1, q_2} \), with \( q_1, q_2 \in S \cup \{v, w\} \), is a navigational expression in \( N(\mathcal{X}) \).
2. If \( (m, n) \in e_{q_1, q_2}(T) \), with \( q_1, q_2 \in S \cup \{v, w\} \), then there exists a run \( \gamma(q_1, m)^{q_1} \ldots (q_2, n) \) of \( A' \) on \( T \) that performs at least one transition.
3. If \( (q_1, n_1) \ldots (q_l, n_l) \) is a run of \( A' \) on \( T \) with \( M[q_1] = \ldots = M[q_{l-1}] = \text{false} \) that performs at least one transition, then we have \((n_1, n_2) \in e_{q_1, q_1}(T)\).

As \( v \neq w \), \( v \) is the only initial state, and \( w \) is the only final state, each accepting run of \( A' \) performs at least one transition. Hence Invariants 2 and 3 imply that \( A' \) and the resulting navigational expression \( e_{v, w} \) are path-equivalent. Invariant 1 implies that the resulting navigational expression is, as required, in \( N(\mathcal{X}) \).

We will use condition automata to remove \( \cap \) and \( \setminus \). The standard approach to constructing the intersection of two finite automata is by making their cross-product. In a relatively straightforward manner, we can apply a similar cross-product construction to condition automata if they are id-transition free. Observe that the id-labeled transitions fulfill a similar role as empty-string-transitions in finite automata and, as such, can be removed, which we show next.

Definition 8 (Identity pair). Let \( \mathcal{A} \) be a condition automaton. The pair \((q, \{q, q_1, \ldots, q_l\})\) is an identity pair of \( \mathcal{A} \) if there exists a path \( q \delta q_1^{id} \ldots \delta q_l^{id} \) in \( \mathcal{A} \) with \( (q, q_1, \ldots, q_l) \).

Lemma 5. Let \( \mathcal{X} \subseteq \{+, \pi, \pi\} \) and let \( \mathcal{A} \) be an \( \mathcal{X} \)-free condition automaton. There exists an id-transition free, \( \mathcal{X} \)-free condition automaton \( \mathcal{A}' \) that is path-equivalent to \( \mathcal{A} \). The condition automaton \( \mathcal{A}' \) is loop-free whenever \( \mathcal{A} \) is loop-free.

Proof (sketch). Let \( \mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma) \) be an \( \mathcal{X} \)-free condition automaton. We construct \( \mathcal{A}' = (S', \Sigma, C, I, F, \delta', \gamma') \) with

\[
S' = \{(q, Q) | (q, Q) \in S \land (q \in I)\},
\]

\[
I' = \{(q, Q) | (q, Q) \in I' \land (q \in \Sigma)\},
\]

\[
F' = \{(q, Q) | (q, Q) \in F' \land (q \in \Sigma)\},
\]

\[
\delta' = \{(q, Q) \in S' \land (q, \ell, \gamma') \land (q, Q) \in F' \land (q \in \Sigma)\}
\]

Even if we can construct the cross-product of two condition automata, this does not directly imply that the cross-product is path-equivalent to the intersection of the condition automata. On automata representing navigational query languages, there exist situations in which the cross-product construction is not equivalent to the intersection of the corresponding regular languages:

Example 5. Let \( \ell' \cap \ell \) be a navigational expression. This expression evaluates to non-empty on the graph of Figure 4, left, but not on the graph of Figure 4, right. If we directly translate \( \ell' \) and \( \ell \) to finite automata and construct their cross-product, the resulting automaton would represent the empty language, which is obviously not equivalent to \( \ell' \cap \ell \) on the graph in Figure 4, left.

On trees, however, the situation of Example 5 cannot occur, as a directed path between two nodes in a tree is always unique. This observation is crucial in showing that the cross-product construction on condition automata works on trees. The lemma below formalizes this observation:

Lemma 6. Let \( A_1 \) and \( A_2 \) be id-transition free condition automata and let \( \mathcal{T} = (N, E) \) be a tree. If there exists a run \( (p_1, n_1)^{e_1} \ldots (p_l, n_{l+1}) \) of \( A_1 \) on \( \mathcal{T} \) and there exists a run \( (p_2, m_1)^{e_2} \ldots (p_{l'}, m_{l'+1}) \) of \( A_2 \) on \( \mathcal{T} \) with \( n_1 = m_1 \) and
Proposition 6. Let $\mathfrak{F} \in \{+, \pi, \tau\}$ and let $A_1$ and $A_2$ be $\mathfrak{F}$-free condition automata. There exists an $\mathfrak{F}$-free condition automaton $A_3$ such that, for every tree $T$, we have $A_3(T) \in A_1(T) \cap A_2(T)$. The condition automaton $A_3$ is loop-free whenever $A_1$ or $A_2$ is loop-free.

Proof (sketch). Let $A_1 = (S_1, \Sigma_1, C_1, I_1, F_1, \delta_1, \gamma_1)$ and $A_2 = (S_2, \Sigma_2, C_2, I_2, F_2, \delta_2, \gamma_2)$ be condition automata. By Lemma 5, we may assume that $A_1$ and $A_2$ are id-transitio- free. Without loss of generality, we may also assume that $S_1 \cap S_2 = \emptyset$. We construct $A_3 = (S_3, \Sigma_3, \Sigma_4, C_3, I_3, F_3, \delta_3, \gamma_3)$ where

$$\delta_3 = \{((p_1, q_1), \ell, (p_2, q_2)) \mid (p_1, \ell, p_2) \in \delta_1 \land (q_1, \ell, q_2) \in \delta_2\};$$

$$\gamma_3 = \{[(p, q), c] \mid c \in \gamma_1(p) \lor c \in \gamma_2(q)\}.$$  

Let $T = (N, E)$ be a tree and let $v, w \in N$ be a pair of nodes. Let $(p_1, m_1) \ell_1 \ldots \ell_q, (p_1, m_1) \ell_1 \ldots \ell_q$ be runs of $A_1$ on $T$ and $A_2$ on $T$, respectively, with $p_1 \in I_1, q_1 \in F_1, p_2 \in I_2, q_2 \in F_2$, in which $v = m_1 = m_3$ and $w = n_{i+1} = m_{i+1}$. Using Lemma 6, we conclude that these runs exist if and only if $i = i_1 = i_2$ and, for all $1 \leq j \leq i$, $l_j = \ell_j = \ell_j$. But then, there also exists a run $((p_1, p_2), v \ell_1 \ldots \ell_q)$ of $A_3$ on $T$ with $(p_1, p_2) \in I_1 \times I_2$ and $(q_1, q_2) \in F_1 \times F_2$. \hfill \Box

Usually, the difference of two free automata $A_1$ and $A_2$ is constructed by first constructing the complement of $A_2$, and then constructing the intersection of $A_1$ with the resulting automaton. We cannot use such a complement construction for condition automata: the complement of a downward binary relation (represented by a condition automaton) is not a downward binary relation. Observe, however, that it is not necessary to consider the full complement when constructing the difference of two condition automata. As the difference of two downward binary relations is itself a downward relation, we can restrict ourselves to the downward complement of a binary relation.

Let $T = (N, E)$ be a tree. We define the downward complement of a binary relation $R \subseteq N \times N$, denoted by $\overline{R}$, as

$$\overline{R}_i = \{(m, n) \mid (m, n) \notin R \land (m, n) \in [\ell]^+(T)\},$$

in which $\ell = \bigcup_{i \in E} \ell_i$. Indeed, if $A_1$ and $A_2$ are condition automata and $T$ is a tree, then we have $A_1(T) \setminus A_2(T) \equiv A_3(T) \cap A_2(T)$.

Hence, we only need to show that condition automata are closed under downward complement. The construction of the downward complement uses deterministic condition automata:

Definition 9 (Deterministic condition automaton). The condition automaton $A = (S, \Sigma, C, I, F, \delta, \gamma)$ is deterministic if it is id-transition free and if it satisfies the following condition: for every tree $T = (N, E)$ and for every path of nodes $m, n$ with $m$ an ancestor of $n$, there exists exactly one run $(q, m)\ell_1 \ldots \ell_q$ of $A$ on $T$ with $q \in \ell_1$.

Example 6. In Figure 5 we visualize a conditional automaton that is deterministic (assuming $\Sigma = \{\ell_1, \ell_2\}$). This deterministic condition automaton accepts node pairs $(m, n)$, $m \neq n$, if $m$ satisfies $\pi_2[\ell_2]^*$ and if there is a path from $m$ to $n$ whose labeling matches the regular expression $\ell_1[\ell_2]^* \ell_1$. It also accepts node pairs $(n, n)$ if $n$ does not satisfy $\pi_2[\ell_2]^*$.\footnote{If we make abstraction of the condition expressions, then Definition 9 reduces to the classical definition of a finite deterministic automaton.}
Algorithm 2 Translation to deterministic condition automaton

1: Let $S_D, I_D$, and $\mathcal{V}$ be empty sets of states.
2: Let $\delta_D$ be an empty transition relation.
3: for $V \subseteq \mathcal{C}$ do
4: \begin{align*}
Q := \{q \mid q \in I \land (q, \ell, p) \in \delta\}
\end{align*}
5: Add new state $(Q, V)$ to $S_D, I_D$, and new
6: end for
7: while $\mathcal{V} \neq \emptyset$ do
8: Take and remove state $(Q, V)$ from $\mathcal{V}$
9: for $\ell \in \Sigma$ do
10: $P := \{p \mid \exists q \in Q \land (q, \ell, p) \in \delta\}$
11: for $W \subseteq \mathcal{C}$ do
12: $P' := \{p \mid (p, W) \in \delta\}$
13: if $(P', W) \notin S_D$ then
14: Add new state $(P', W)$ to $S_D$ and new
15: end if
16: Add new transition $(Q, V, \ell, (P', W))$ to $\delta_D$
17: end for
18: end for
19: end while

Proposition 7. Let $\triangledown \subseteq \{+, \pi, \pi\}$ and let $A$ be an $\triangledown$-free condition automaton. There exists a condition automaton $A'$ such that, for every $T$, we have $A(T) = A'(T)$. The condition automaton $A'$ is $\triangledown$-free if $+$ $\notin \triangledown$ and $(\pi, \pi)$-free if $\pi, \pi \notin \triangledown$.

Proof. Let $A' = (A', \triangledown')$ be the deterministic condition automaton equivalent to $A$. We construct $A' = (A', \triangledown')$ such that, for every $T$, we have $A(T) = A'(T)$. The condition automaton $A'$ is $\triangledown$-free if $+$ $\notin \triangledown$ and $(\pi, \pi)$-free if $\pi, \pi \notin \triangledown$, and loop-free whenever $A_1$ is loop-free.

Corollary 3. Let $A_1$ and $A_2$ be condition automata. There exists a condition automaton $A_3$ such that, for every $T$, we have $A_3(T) = A_1(T) \cup A_2(T)$. The condition automaton $A_3$ is $(\triangledown)$-free if $+$ $\notin \triangledown$. $(\pi, \pi)$-free if $\pi, \pi \notin \triangledown$, and loop-free whenever $A_1$ is loop-free.

Proposition 6 and Corollary 3 only remove intersection and difference at the highest level: these results ignore the expressions inside conditions. To fully remove intersection and difference, we use an induction argument on the depth of conditions.

Let e be an expression in $\mathcal{N}(\pi, \pi, \pi, \pi, \pi, \pi)$. We define the condition-depth of $e$, denoted by $cdepth(e)$, as

\[
cdepth(e) = \begin{cases} 0 & \text{if } e \in \{0, \emptyset\}; \\ 0 & \text{if } e = \ell, \text{ with } \ell \text{ an edge-label}; \\ cdepth(e') & \text{if } e = [e']; \\ cdepth(e') + 1 & \text{if } e \in \{\pi_1[e'], \pi_2[e'], \pi_1[e'], \pi_2[e']\}; \\ \max(\text{cdepth}(e_1), \text{cdepth}(e_2)) & \text{if } e \in \{e_1 \cap e_2, e_1 \cup e_2, e_1 \cap e_2\}. \end{cases}
\]

Theorem 3. Let $\triangledown \subseteq \{+, \pi, \pi, \pi, \pi, \pi\}$. On labeled trees we have $\mathcal{N}(\triangledown) \subseteq \mathcal{N}(\triangledown \setminus \{\pi, \pi, \pi, \pi\})$.

Proof (sketch). We use induction on both the condition-depth of expressions and on the length of expressions (of a specific condition-depth). Thereto, we use Proposition 5 to translate navigational sub-expressions $e_1$ and $e_2$, we use Proposition 4, Proposition 6, and Corollary 3 to construct condition automata representing navigational sub-expressions $[e_1]^+$, $e_1 \circ e_2, e_1 \cup e_2, e_1 \cap e_2$, and $e_1 \pi e_2$. Finally, we use Proposition 5 to translate the resulting condition automaton back to a navigational expression in $\mathcal{N}(\triangledown \setminus \{\pi, \pi, \pi, \pi\})$. \hfill \square

Observe that Theorem 3 does not strictly depend on the graph being a tree: indirectly, Theorem 3 depends on Lemma 6, which hold for all graphs in which each pair of nodes is connected by at most one directed path. Hence, the results of Theorem 3 can be generalized to, for example, forests.

The concept of condition automata to represent and manipulate navigational expressions can also be used to simplify boolean queries. We can, for example, use condition automata to remove $\pi$-conditions from any expression in $\mathcal{N}(\pi)$ or in $\mathcal{N}(\pi)$. We define the condition-depth of a $\{\pi\}$-free condition automaton $A = (S, \Sigma, C, I, F, \delta, \gamma)$, denoted by $\text{cdepth}(A)$, as $\text{cdepth}(A) = \max\{\text{cdepth}(e) \mid e \in C\}$. We define the condition-weight of $A$, denoted by $\text{cweight}(A)$, as

\[
\text{cweight}(A) = \left|\{c \mid c \in C \land \text{cdepth}(A) = \text{cdepth}(A)\}\right|.
\]

Lemma 8. Let $A$ be a $\{\pi\}$-free and id-translation free condition automaton. If $\text{cdepth}(A) > 0$, then there exists a $\{\pi\}$-free and id-translation free condition automaton $A_c$ such that

1. for every labeled chain $C$, we have $A_c(C) = \emptyset$ if and only if $A(C) = \emptyset$; and
2. $\text{cdepth}(A) > \text{cdepth}(A_c)$ or $\text{cdepth}(A) = \text{cdepth}(A_c) \wedge \text{cweight}(A) > \text{cweight}(A_c)$.

The condition automaton $A_c$ is loop-free and $\{\pi\}$-free whenever $A$ is loop-free and $\{\pi\}$-free.
δ₁₈ = \{(p, P ∪ P'), (q, Q) \mid (p, P ∪ P') ∈ S_e ∧ P' ⊆ F' ∧ (q, Q) ∈ S_e ∧ (q = q') \land (p = p') \land q = q' \land (P, q, Q) ∈ δ_{P\langle S'\rangle}\};

δ₁₉ = \{(p, P ∪ P'), (q, Q \cup \{q'\}) \mid (p, P ∪ P') ∈ S_e ∧ (q, Q) ∈ S_e ∧ (p, q, Q) ∈ S_e ∧ q ∈ S_e ∧ (q = q') \land (P, q, Q) ∈ δ_{P\langle S'\rangle}\};

δ₉ = δ₁₈ ∪ δ₁₉;

γ₉ = \{(q, Q, c') \mid (q, Q) ∈ S_e \land (c' ∈ γ(q) \lor (\exists q') q = Q ∧ c' ∈ γ'(q'))\}.

For c = π₂[c'], the construction is analogous.

Lemma 8 only removes a single π-condition. To fully remove π-conditions, we repeat these removal steps until no π-conditions are left. This leads to the following result:

**Theorem 4.** Let \(\emptyset \subseteq \{+\}. On labeled chains we have \(N(\emptyset) \preceq_b N(\emptyset \setminus \{\pi\})\).

**Proof.** We use Proposition 5 to translate a navigational expression to a condition automaton \(A\), then we repeatedly apply Lemma 8 to remove conditions, and, finally, we use Proposition 5 to translate back to a navigational expression in \(N(\emptyset \setminus \{\pi\})\). Observe that only a finite number of condition removal steps on \(A\) can be made, as Lemma 8 guarantees that either \(cdepth(A)\) strictly decreases or else \(cdepth(A)\) does not change and \(cweight(A)\) strictly decreases.

We observed that Theorem 3 does not strictly depend on the graph being a tree. A similar observation holds for Theorem 4: for boolean queries, we can remove a π-condition whenever the condition checks a part of the graph that does not branch. This is the case for π₂-conditions on trees, as trees do not have branching in the direction from a node to its ancestors. For π₁, this observation does not hold, as is illustrated by the proof of Proposition 3.

**Proposition 8.** Let \(\emptyset \subseteq \{+, \pi\}. On labeled trees we have \(N(\emptyset \cup \{\pi\}) \preceq_b N(\emptyset)\), but \(N(\emptyset \cup \{\pi_1\}) \npreceq_b N(\emptyset)\).

### 4. Related Work

Tree query languages have been widely studied, especially in the setting of the XML data model using XPath-like query languages. For an overview, we refer to Benedikt et al. [2]. Due to the large body of work on querying of tree-based data models, we only point to related work that studies similar expressiveness problems.

Benedikt et al. [3] study the expressive power of the XPath fragments with and without the parent axis, with and without ancestor and descendant axes, and with and without qualifiers (which are π₁-conditions). Furthermore, they study closure properties of these XPath fragments under intersection and complement. As such, the work by Benedikt et al. answers similar expressiveness questions as our work does. The Core XPath fragments studied by Benedikt et al. do, however, not include non-monotone operators such as π and γ and allow only for a very restricted form of transitive closure, required to define the ancestor and descendant axes. Hence, queries such as \([F_1 \circ F_2]^+\) and \(f_1 \circ [f_2 \circ f_3]^+\), used in Proposition 9, are not expressible in these XPath fragments.

When accounting for the difference between the node-labeled tree model used by Benedikt et al. [3] and the edge-labeled tree model used here, and when restricting ourselves to the downward fragments as studied here, we see that all relevant XPath fragments of Benedikt et al., fragment \(X_{\langle\rangle}\) and its fragments, are strictly less expressive than the navigational query language \(N(\{\pi, \pi_1\})\). Furthermore, we observe that \(X_{\langle\rangle}\) is path-equivalent to \(N(\emptyset)\) and that \(X_{\langle\rangle}\) is path-equivalent to \(N(\{\pi_1\})\). As such, our work extends some of the results of Benedikt et al. to languages that have a more general form of transitive closure.

Conditional XPath, Regular XPath, and Regular XPath	extsuperscript{**} [19, 20, 23, 24] are studied with respect to a sibling-ordered node-labeled tree data model. The choice of a sibling-ordered tree data model makes these studies incomparable with our work: on sibling-ordered trees, Conditional XPath is equivalent with FO[3], and FO[3] is equivalent to general first-order logic [19]. This result does not extend to our tree data model: on our tree data model, FO[3] cannot express simple first-order counting queries such as

\[
\exists n_3 \exists c_0 \exists c_1 \exists c_2 \exists c_3 \exists c_4 \exists e(n, c_1) \land \exists e(n, c_2) \land \exists e(n, c_3) \land \exists e(n, c_4) \land (c_1 \neq c_2) \land (c_1 \neq c_3) \land (c_1 \neq c_4) \land (c_2 \neq c_3) \land (c_2 \neq c_4) \land (c_3 \neq c_4),
\]

which is true on all trees that have a node with at least four distinct children. Although \(N(\{\pi, \pi_1, \pi_\land, \pi_\lor\})\) is not a fragment of FO[3], due to the inclusion of the transitive closure operator, a straightforward brute-force argument shows that not even \(N(\{\pi, \pi_1, \pi_\land, \pi_\lor\})\) can express these kinds of counting queries. With an ordered sibling axis, as present in the sibling-ordered tree data model, the above counting query is boolean-equivalent to sibling \(\circ\) sibling \(\circ\) sibling

Due to these differences in the tree data models used, the closure properties under intersection and complementation for Conditional XPath and Regular XPath	extsuperscript{**} cannot readily be translated to closure properties for the navigational query languages we study. Moreover, even if the closure properties for Conditional XPath and Regular XPath	extsuperscript{**} could be translated to our setting, these results would only cover a single fragment.

Lastly, the XPath algebra of Gyssens et al. [12, 15], when restricted to the downward fragment, corresponds to the navigational query language \(N(\{\pi, \pi_1, \pi_\land, \pi_\lor\})\), and the positive algebra of Wu et al. [26], when restricted to the downward fragment, corresponds to the navigational query language \(N(\{\pi, \pi_1\})\). Although these studies include some expressiveness results, these results are dependent on the availability of a parent-axis (or a converse operator), and, thus, these results are not relevant for our study.
On graphs, the navigational query languages (both labeled an unlabeled) have already been studied in full detail [8–11]. There has also been some work on the expressive power of variations of the regular path queries and nested regular path queries [1], which are equivalent to fragments of the navigational query languages. These results are all provided on general graphs. We were able to strengthen several known separation results, by showing that they already hold on very simple graphs, namely on chains and trees.

5. Conclusions and Directions for Future Work

This paper studies the expressive power of the downward navigational query languages on trees and chains, both in the labeled and in the unlabeled case. We were able to present the complete Hasse diagrams of relative expressiveness, visualized in Figure 1. In particular, our results show, for each fragment of the navigational query languages that we study, whether it is closed under difference and intersection when applied on trees. These results are shown using the concept of condition automata to represent and manipulate navigational expressions. We also used condition automata to prove that, on labeled chains, projections do not provide additional expressive power for boolean queries.

The next step in this line of research is to explore common non-downward operators, starting with including the converse of the edge relation (which provides, among other things, the parent axis of XPath and, in combination with transitive closure, provides the ancestor axis) and node inequality via the diversity operator. Particularly challenging are the interactions with \( \land \) and \( \lor \). We conjecture, for example, that \( N(\land, di) \neq N(\land, \pi, \land, di) \), but this conjecture is still wide open, even on unlabeled chains.

Another direction is the study of languages with only one of the projections (or one of the coprojections) as Proposition 8 shows that in some cases adding only \( \pi_1 \) or only \( \pi_2 \) may affect the expressive power. Indeed, various XPath fragments and the nested RPOQs only provide operators similar to \( \pi_1 \). Another interesting avenue of research is to explore the relation between the navigational expressions (and FO[3]) on restricted relational structures and FO[2], the language of first-order logic formulae using at most two variables [14]. Our results for \( N(\land, \pi, \land) \) on unlabeled trees already hint at a significant overlap of FO[3] and FO[2] for boolean queries: the query \( E^\land \) can easily be expressed by FO[2] algebras with semi-joins via \( E \times E \times \cdots \times E \). A last avenue of research we wish to mention is to consider other semantics for query-equivalence and other tree data models, such as the root equivalence of Benedikt et al. [3] and the ordered-sibling tree data model.

References


