Relative expressive power of downward fragments of navigational query languages on trees and chains

Peer-reviewed author version

HELLINGS, Jelle; GYSSENS, Marc; Wu, Yuqing; Van Gucht, Dirk; VAN DEN BUSSCHE, Jan; VANSUMMEREN, Stijn & Fletcher, George H. L. (2015) Relative expressive power of downward fragments of navigational query languages on trees and chains. In: Cheney, James; Neumann, Thomas (Ed.). Proceedings of the 15th Symposium on Database Programming Languages, p. 59-68.

DOI: 10.1145/2815072.2815081
Handle: http://hdl.handle.net/1942/21032

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

Jelle Hellings[1] Marc Gyssens[1] Yuqing Wu[2] Dirk Van Gucht[3]
Jan Van den Bussche[1] Stijn Vansummeren[4] George H. L. Fletcher[5]

[1]Hasselt University and Transnational University of Limburg, Belgium
[2]Pomona College, California, USA
[3]Indiana University, Indiana, USA
[4]Université Libre de Bruxelles, Belgium
[5]Eindhoven 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 ($\circ$) and union ($\cup$). 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 ($\overline{\pi}$)—which can be used to express negated conditions; intersection ($\cap$); and difference ($\setminus$). 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 π 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 $\setminus$ operations. We also use these condition automata to show that, in the boolean case, π 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 expressive 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 RPQs 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 $\setminus$ 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 expressive 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 $\mathcal{G} = (\mathbf{N}, \mathbf{E})$, with $\mathbf{N}$ a finite set of nodes and $\mathbf{E}$ a set of binary edge relations. Each edge relation $\ell \in \mathbf{E}$ is a subset of $\mathbf{N} \times \mathbf{N}$ and is interpreted as those edges labeled with ℓ. A graph is *unlabeled* if $|\mathbf{E}| = 1$. On unlabeled graphs, we use $\mathcal{E}$ to refer to the single edge relation. A *tree* $\mathcal{T} = (\mathbf{N}, \mathbf{E})$ 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 $\mathcal{G} = (\mathbf{N}, \mathbf{E})$ is a sequence $n_1 \ell_1 n_2 \dots \ell_{i-1} n_i$ with $n_1, \dots, n_i \in \mathbf{N}$, $\ell_1, \dots, \ell_{i-1} \in \mathbf{E}$, 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

$$e := \emptyset \mid \mathrm{id} \mid \mathrm{di} \mid \ell \text{ (for } \ell \text{ an edge-label)} \mid e \circ e \mid e \cup e \mid [e]^- \mid [e]^+ \mid \pi_1[e] \mid \pi_2[e] \mid \overline{\pi}_1[e] \mid \overline{\pi}_2[e] \mid e \cap e \mid e \setminus e.$$

We also use the following shorthand notation

$$[e]^* \equiv [e]^+ \cup \mathrm{id}.$$

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

$$\begin{aligned}
\emptyset\langle\mathcal{G}\rangle &= \emptyset;\\
\mathrm{id}\langle\mathcal{G}\rangle &= \{(m,m) \mid m \in \mathbf{N}\};\\
\mathrm{di}\langle\mathcal{G}\rangle &= \{(m,n) \mid m,n \in \mathbf{N} \wedge m \neq n\};\\
\ell\langle\mathcal{G}\rangle &= \ell \quad (\text{for } \ell \text{ an edge-label with } \ell \in \mathbf{E});\\
\ell\langle\mathcal{G}\rangle &= \emptyset \quad (\text{for } \ell \text{ an edge-label with } \ell \notin \mathbf{E});\\
e_1 \circ e_2\langle\mathcal{G}\rangle &= e_1\langle\mathcal{G}\rangle \circ e_2\langle\mathcal{G}\rangle;\\
e_1 \cup e_2\langle\mathcal{G}\rangle &= e_1\langle\mathcal{G}\rangle \cup e_2\langle\mathcal{G}\rangle;\\
[e]^-\langle\mathcal{G}\rangle &= [e\langle\mathcal{G}\rangle]^-;\\
[e]^+\langle\mathcal{G}\rangle &= [e\langle\mathcal{G}\rangle]^+;\\
\pi_1[e]\langle\mathcal{G}\rangle &= \{(m,m) \mid \exists n\ (m,n) \in e\langle\mathcal{G}\rangle\};\\
\pi_2[e]\langle\mathcal{G}\rangle &= \{(m,m) \mid \exists n\ (n,m) \in e\langle\mathcal{G}\rangle\};\\
\overline{\pi}_1[e]\langle\mathcal{G}\rangle &= \{(m,m) \mid (m \in \mathbf{N}) \wedge (\neg\exists n\ (m,n) \in e\langle\mathcal{G}\rangle)\};\\
\overline{\pi}_2[e]\langle\mathcal{G}\rangle &= \{(m,m) \mid (m \in \mathbf{N}) \wedge (\neg\exists n\ (n,m) \in e\langle\mathcal{G}\rangle)\};\\
e_1 \cap e_2\langle\mathcal{G}\rangle &= e_1\langle\mathcal{G}\rangle \cap e_2\langle\mathcal{G}\rangle;\\
e_1 \setminus e_2\langle\mathcal{G}\rangle &= e_1\langle\mathcal{G}\rangle \setminus e_2\langle\mathcal{G}\rangle.
\end{aligned}$$

In the above, $\mathrm{R}_1 \circ \mathrm{R}_2$, for binary relations $\mathrm{R}_1, \mathrm{R}_2 \subseteq \mathbf{N} \times \mathbf{N}$, is defined by $\mathrm{R}_1 \circ \mathrm{R}_2 = \{(m,n) \mid \exists z\ ((m,z) \in \mathrm{R}_1) \wedge ((z,n) \in \mathrm{R}_2)\}$, $[\mathrm{R}]^-$, for a binary relation $\mathrm{R} \subseteq \mathbf{N} \times \mathbf{N}$, is defined by $[\mathrm{R}]^- = \{(m,n) \mid (n,m) \in \mathrm{R}\}$, and $[\mathrm{R}]^+$, for a binary relation $\mathrm{R} \subseteq \mathbf{N} \times \mathbf{N}$, is defined by $[\mathrm{R}]^+ = \bigcup_{1 \leq k} \mathrm{R}^k$, in which we define

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

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 $\mathcal{G} = (\mathbf{N}, \mathbf{E})$ be a graph. Using *path query semantics*, e on $\mathcal{G}$ evaluates to $e\langle\mathcal{G}\rangle$, and using *boolean query semantics*, e on $\mathcal{G}$ evaluates to the truth value of $e\langle\mathcal{G}\rangle \neq \emptyset$.

Example 1. Consider the class-structure of a program described by relations *subclass* and *method*. In this setting, the query $[\mathit{subclass}]^+$ returns the relation between classes and their descendant classes, the query $\overline{\pi}_1[\mathit{method}]$ returns all classes that do not define own methods, and the query $\pi_1[\mathit{method}] \setminus \pi_1\big[[\mathit{subclass}]^+ \circ \mathit{method}\big]$

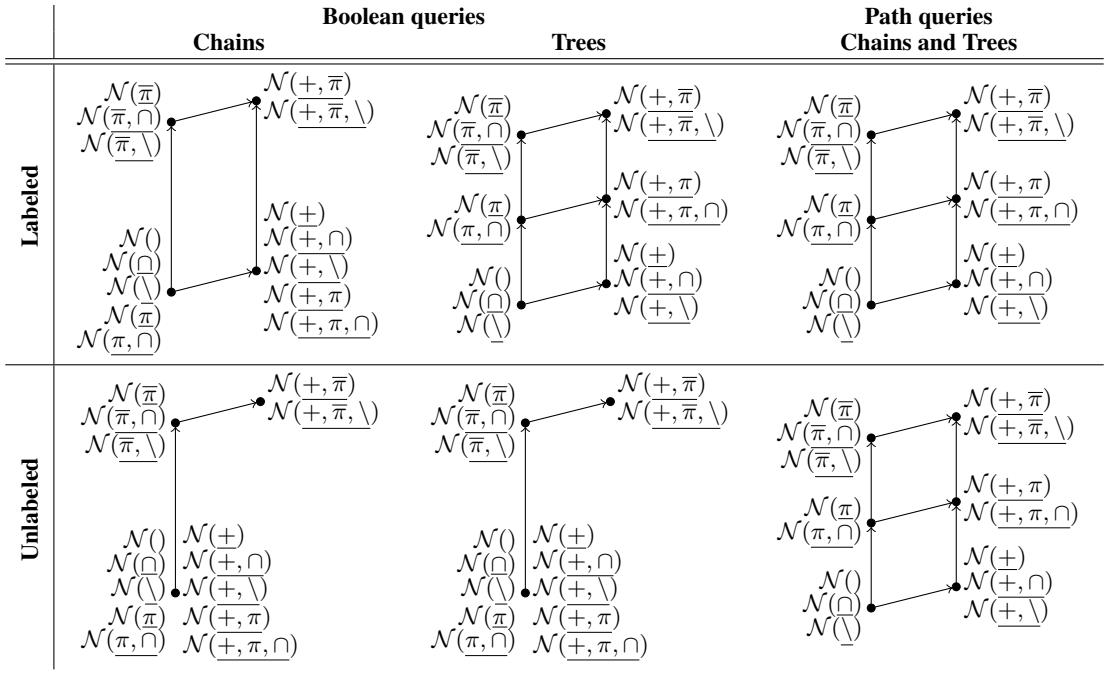

Figure 1. The full Hasse diagrams describing the relations between the expressive power of the various fragments of $\mathcal{N}(+, \overline{\pi}, \pi, \setminus, \cap)$. An edge $A \longrightarrow B$ indicates $A \preceq B$ and $B \not\preceq A$.

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 query languages as the navigational query languages. The most restricted language we study, denoted by $\mathcal{N}()$, is the language that allows the basic operators $\emptyset$, id, ℓ (for ℓ an edge-label), $\circ$, and $\cup$. If $\mathfrak{F} \subseteq \{+, \pi, \overline{\pi}, \cap, \setminus\}$, then $\mathcal{N}(\mathfrak{F})$ denotes the language that allows all basic operators and, additionally, the operators in $\mathfrak{F}$. Here, π represents both projection operators (π_1 and π_2), and $\overline{\pi}$ both coprojection operators ($\overline{\pi}_1$ and $\overline{\pi}_2$).

We say that queries e_1 and e_2 are *path-equivalent* if, for all graphs $\mathcal{G}$, we have $e_1\langle\mathcal{G}\rangle = e_2\langle\mathcal{G}\rangle$, and are *boolean-equivalent* if, for all graphs $\mathcal{G}$, we have $e_1\langle\mathcal{G}\rangle \neq \emptyset$ if and only if $e_2\langle\mathcal{G}\rangle \neq \emptyset$. If $\mathcal{N}_1$ and $\mathcal{N}_2$ are query languages, then $\mathcal{N}_2$ *path-subsumes* $\mathcal{N}_1$, denoted by $\mathcal{N}_1 \preceq_p \mathcal{N}_2$, if every query in $\mathcal{N}_1$ is path-equivalent to a query in $\mathcal{N}_2$. Likewise, $\mathcal{N}_2$ *boolean-subsumes* $\mathcal{N}_1$, denoted by $\mathcal{N}_1 \preceq_b \mathcal{N}_2$, if every query in $\mathcal{N}_1$ is boolean-equivalent to a query in $\mathcal{N}_2$. We say that two languages $\mathcal{N}_1$ and $\mathcal{N}_2$ are *path-equivalent* if $\mathcal{N}_1 \preceq_p \mathcal{N}_2$ and $\mathcal{N}_2 \preceq_p \mathcal{N}_1$, and *boolean-equivalent* if $\mathcal{N}_1 \preceq_b \mathcal{N}_2$ and $\mathcal{N}_2 \preceq_b \mathcal{N}_1$.

By restricting the set of graphs to labeled or unlabeled graphs, labeled or unlabeled trees, or labeled or unlabeled chains, we can also speak of path-subsumption and boolean-subsumption with respect to one of the more restricted classes of graphs. We can say, for example, that $\mathcal{N}_2$ boolean-subsumes $\mathcal{N}_1$ on unlabeled trees if, for every query e in $\mathcal{N}_1$, there exists a query e' in $\mathcal{N}_2$ such that, for all unlabeled trees $\mathcal{T}$, we have $e\langle\mathcal{T}\rangle \neq \emptyset$ if and only if $e'\langle\mathcal{T}\rangle \neq \emptyset$.

Observe that $\mathcal{N}(\mathfrak{F}_1) \preceq_p \mathcal{N}(\mathfrak{F}_2)$ implies $\mathcal{N}(\mathfrak{F}_1) \preceq_b \mathcal{N}(\mathfrak{F}_2)$, and, by contraposition, $\mathcal{N}(\mathfrak{F}_1) \not\preceq_b \mathcal{N}(\mathfrak{F}_2)$ implies $\mathcal{N}(\mathfrak{F}_1) \not\preceq_p \mathcal{N}(\mathfrak{F}_2)$ [8]. We also observe that we can carry over subsumption results from one type of graph to another, considering the relationships of Figure 2. Inequivalence results carry over from specialized

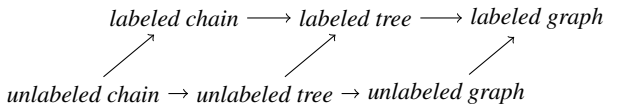

Figure 2. Relationship between the various types of graphs; arrows indicate is-a relations.

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 an 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:

$$\pi_1[e] \equiv \overline{\pi}_1[\overline{\pi}_1[e]];$$
$$\pi_2[e] \equiv \overline{\pi}_2[\overline{\pi}_2[e]];$$
$$\overline{\pi}_1[e] \equiv \mathrm{id} \setminus \pi_1[e];$$
$$\overline{\pi}_2[e] \equiv \mathrm{id} \setminus \pi_2[e];$$
$$e_1 \cap e_2 \equiv e_1 \setminus (e_1 \setminus e_2).$$

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

Proposition 1 (Fletcher et al. [8–11])**.** *Let $\mathfrak{F}_1, \mathfrak{F}_2 \subseteq \{+, \pi, \overline{\pi}, \cap, \setminus\}$. On labeled graphs, we have $\mathcal{N}(\mathfrak{F}_1) \preceq_p \mathcal{N}(\mathfrak{F}_2)$ and $\mathcal{N}(\mathfrak{F}_1) \preceq_b \mathcal{N}(\mathfrak{F}_2)$ if and only if $\mathfrak{F}_1 \subseteq \underline{\mathfrak{F}_2}$. On unlabeled graphs, we have $\mathcal{N}(\mathfrak{F}_1) \preceq_p \mathcal{N}(\mathfrak{F}_2)$ if and only if $\mathfrak{F}_1 \subseteq \underline{\mathfrak{F}_2}$, and we have $\mathcal{N}(\mathfrak{F}_1) \preceq_b \mathcal{N}(\mathfrak{F}_2)$ if and only if $\mathfrak{F}_1 \subseteq \underline{\mathfrak{F}_2}$ or $\mathfrak{F}_1 \subseteq \{\pi\}$ and $\mathfrak{F}_2 = \mathfrak{F}_1 \cup \{+\}$.*

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 π 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 π does not add expressive power for boolean queries on labeled chains. 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 $\mathfrak{F}_1, \mathfrak{F}_2 \subseteq \{+, \overline{\ }, \pi, \cap, \setminus\}$. We have $\mathcal{N}(\mathfrak{F}_1) \preceq_b \mathcal{N}(\mathfrak{F}_2)$, respectively $\mathcal{N}(\mathfrak{F}_1) \preceq_p \mathcal{N}(\mathfrak{F}_2)$, on unlabeled chains, respectively labeled chains, unlabeled trees, or labeled trees if and only if there exists a directed path from $\underline{\mathfrak{F}_1}$ to $\underline{\mathfrak{F}_2}$ in the corresponding Hasse diagram of Figure 1.*

3.1 Identity-based Results

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

Lemma 1. *Let $\mathcal{C} = (\mathbf{N}, \mathbf{E})$ be an unlabeled chain, let $\mathfrak{F} \subseteq \{+, \cap, \setminus\}$, and let e be a navigational expression in $\mathcal{N}(\mathfrak{F})$. We have $e\langle\mathcal{C}\rangle \cap \mathrm{id}\langle\mathcal{C}\rangle = \emptyset$ or $e\langle\mathcal{C}\rangle \cap \mathrm{id}\langle\mathcal{C}\rangle = \mathrm{id}\langle\mathcal{C}\rangle$.*

Proposition 2. *Let $\mathfrak{F} \subseteq \{+, \cap, \setminus\}$. On unlabeled chains we have $\mathcal{N}(\pi) \not\preceq_p \mathcal{N}(\mathfrak{F})$.*

Proof. Let $\mathcal{C} = (\mathbf{N}, \mathbf{E})$ be a chain with $2 \leq |\mathbf{N}|$. We have $\emptyset \subsetneq \pi_1[\mathcal{E}]\langle\mathcal{C}\rangle = \pi_1[\mathcal{E}] \cap \mathrm{id}\langle\mathcal{C}\rangle \subsetneq \mathrm{id}\langle\mathcal{C}\rangle$. By Lemma 1, no expression in $\mathcal{N}(\mathfrak{F})$ can be path-equivalent with $\pi_1[\mathcal{E}]$. □

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[1] operator $\overline{\pi}$ 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 $\mathcal{G}_1 = (\mathbf{N}_1, \mathbf{E}_1)$ and $\mathcal{G}_2 = (\mathbf{N}_2, \mathbf{E}_2)$ be graphs. We say that a mapping $h : \mathbf{N}_1 \to \mathbf{N}_2$ is a *homomorphism* from $\mathcal{G}_1$ to $\mathcal{G}_2$ if, for every pair of nodes $m, n \in \mathbf{N}_1$ and every edge-label ℓ, 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 $\overline{\pi}$ and $\setminus$ are closed under homomorphisms.

Lemma 2. *Let $\mathfrak{F} \subseteq \{+, \pi, \cap\}$. The language $\mathcal{N}(\mathfrak{F})$ is closed under homomorphisms: for every navigational expression e in $\mathcal{N}(\mathfrak{F})$, every pair of graphs $\mathcal{G}_1 = (\mathbf{N}_1, \mathbf{E}_1)$ and $\mathcal{G}_2 = (\mathbf{N}_2, \mathbf{E}_2)$, and every homomorphism h of $\mathcal{G}_1$ to $\mathcal{G}_2$, we have that $(m, n) \in e\langle\mathcal{G}_1\rangle$ implies $(h(m), h(n)) \in e\langle\mathcal{G}_2\rangle$.*

[1] A navigational expression e is monotone if, for every graph $\mathcal{G}$ and every graph $\mathcal{G}'$ obtained from adding nodes or edges to $\mathcal{G}$, we have $e\langle\mathcal{G}\rangle \subseteq e\langle\mathcal{G}'\rangle$.

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

Lemma 3. *Let $\mathcal{C}_1 = (\mathbf{N}_1, \mathbf{E}_1)$ and $\mathcal{C}_2 = (\mathbf{N}_2, \mathbf{E}_2)$ be unlabeled chains. If $|\mathbf{N}_1| \leq |\mathbf{N}_2|$, then there exists a homomorphism of $\mathcal{C}_1$ to $\mathcal{C}_2$. Let $\mathcal{T} = (\mathbf{N}_\mathcal{T}, \mathbf{E}_\mathcal{T})$ be an unlabeled tree and let $\mathcal{C} = (\mathbf{N}_\mathcal{C}, \mathbf{E}_\mathcal{C})$ be an unlabeled chain. If $|\mathbf{N}_\mathcal{C}| \geq \mathrm{depth}(\mathcal{T})$, then there exists a homomorphism of $\mathcal{T}$ to $\mathcal{C}$, and if $|\mathbf{N}_\mathcal{C}| \leq \mathrm{depth}(\mathcal{T})$, then there exists a homomorphism of $\mathcal{C}$ to $\mathcal{T}$.*

We use these homomorphisms between unlabeled chains and unlabeled trees to show that only fragments that can express the non-monotone operator $\overline{\pi}$, 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 $\mathfrak{F} \subseteq \{+, \pi, \cap\}$. On unlabeled trees we have $\mathcal{N}(\mathfrak{F}) \preceq_b \mathcal{N}()$.*

Proof. Let $\mathcal{T} = (\mathbf{N}, \mathbf{E})$ be a tree and let $\mathcal{C}' = (\mathbf{N}', \mathbf{E}')$ be a chain with $\mathrm{depth}(\mathcal{T}) = |\mathbf{N}'|$. By Lemma 3, there exists a homomorphism $h_1 : \mathbf{N} \to \mathbf{N}'$ of $\mathcal{T}$ to $\mathcal{C}'$ and a homomorphism $h_2 : \mathbf{N}' \to \mathbf{N}$ of $\mathcal{C}'$ to $\mathcal{T}$. Hence, by Lemma 2, we have, for every navigational expression e' in $\mathcal{N}(\mathfrak{F})$, $e\langle\mathcal{T}\rangle \neq \emptyset$ if and only if $e\langle\mathcal{C}'\rangle \neq \emptyset$. As a consequence, we can conclude that no navigational expression in $\mathcal{N}(\mathfrak{F})$ can distinguish between trees and chains of equal depth.

Let $\mathcal{C} = (\mathbf{N}, \mathbf{E})$ and $\mathcal{C}' = (\mathbf{N}', \mathbf{E}')$ be chains such that $|\mathbf{N}| < |\mathbf{N}'|$. By Lemma 3, there exists a homomorphism $h : \mathbf{N} \to \mathbf{N}'$ of $\mathcal{C}$ to $\mathcal{C}'$. Hence, by Lemma 2, we have, for every navigational expression e' in $\mathcal{N}(\mathfrak{F})$, $e\langle\mathcal{C}\rangle \neq \emptyset$ implies $e\langle\mathcal{C}'\rangle \neq \emptyset$. 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 $\mathcal{N}(\mathfrak{F})$. We choose $\mathcal{C} = (\mathbf{N}, \mathbf{E})$ to be the chain with minimum depth such that $e\langle\mathcal{C}\rangle \neq \emptyset$. If no such chain exists, then, by the two properties shown above, e is boolean-equivalent to $\emptyset$. Since $\mathcal{C}$ is also the chain with minimum depth such that $\mathcal{E}^k\langle\mathcal{C}\rangle \neq \emptyset$, with $k = |\mathbf{N}| - 1$, we may conclude, by the two properties shown above, that $\mathcal{E}^k$ and e are boolean-equivalent. □

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

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

Corollary 2. *Let $\mathfrak{F} \subseteq \{+, \pi, \cap\}$. On unlabeled chains we have $\mathcal{N}(\overline{\pi}) \not\preceq_b \mathcal{N}(\mathfrak{F})$.*

Proof (sketch). Let e be the expression $\overline{\pi}_2[\mathcal{E}] \circ \mathcal{E} \circ \overline{\pi}_1[\mathcal{E}]$. If a chain $\mathcal{C}$ has depth two, then we have $e\langle\mathcal{C}\rangle \neq \emptyset$. For all chains $\mathcal{C}'$ with a depth other than two, we have $e\langle\mathcal{C}'\rangle = \emptyset$. In Corollary 1 we already showed that every expression in $\mathcal{N}(\mathfrak{F})$ is boolean-equivalent to $\emptyset$ or $\mathcal{E}^k$, which are both boolean-inequivalent to e. □

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

Lemma 4. *Let $\mathfrak{F} \subseteq \{+\}$. Let e be a navigational expression in $\mathcal{N}(\mathfrak{F})$. If e does not utilize the operator $\emptyset$, then there exists a labeled chain $\mathcal{C}$ such that $e\langle\mathcal{C}\rangle \neq \emptyset$.*

Proposition 3. *Let $\mathfrak{F} \subseteq \{+\}$. On labeled trees we have $\mathcal{N}(\pi) \not\leq_b \mathcal{N}(\mathfrak{F})$.*

Proof. Let ℓ_1 and ℓ_2 be two edge-labels and let $e = \pi_1[\ell_1] \circ \pi_1[\ell_2]$ 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 ℓ_1 and ℓ_2, respectively. Hence, for all chains $\mathcal{C} = (\mathbf{N}, \mathbf{E})$, we have $e\langle\mathcal{C}\rangle = \emptyset$. As e never evaluates to `true` on chains, we use Lemma 4 to conclude that no navigational expression in $\mathcal{N}(\mathfrak{F})$ 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 relative straightforward way to prove that $\mathcal{N}(+, \cap, \setminus) \preceq_p \mathcal{N}(+)$.

Example 2. We can, for example, rewrite the navigational expressions $[\ell^3]^+ \cap [\ell^7]^+$ and $[\ell^3]^+ \setminus [\ell^7]^+$ to path-equivalent navigational expressions that do not use $\cap$ or $\setminus$:

$$[\ell^3]^+ \cap [\ell^7]^+ \equiv [\ell^{21}]^+;$$
$$[\ell^3]^+ \setminus [\ell^7]^+ \equiv (\ell^3 \cup \ell^6 \cup \ell^9 \cup \ell^{12} \cup \ell^{15} \cup \ell^{18}) \circ [\ell^{21}]^*.$$

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

Definition 5 (Condition)**.** A navigational expression e is a *condition* if, for every labeled tree $\mathcal{T}$, we have $e\langle\mathcal{T}\rangle \subseteq \mathrm{id}\langle\mathcal{T}\rangle$.

Here, we only consider conditions of the form $\emptyset$, id, $\pi_1[e']$, $\pi_2[e']$, $\overline{\pi}_1[e']$, or $\overline{\pi}_2[e']$, with e' a navigational expression.

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

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

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

$$S = \{q_1, q_2, q_3, q_4\};$$
$$\Sigma = \{\ell_1, \ell_2, \ell_3\};$$
$$C = \{\mathrm{id}, \pi_1[{\ell_1}^2], \pi_2[{\ell_2}^3]\};$$
$$I = \{q_1, q_4\};$$
$$F = \{q_3, q_4\};$$
$$\delta = \{(q_1, \ell_1, q_2), (q_1, \ell_3, q_4), (q_2, \ell_2, q_2), (q_2, \ell_2, q_3)\}; \text{ and}$$
$$\gamma = \{(q_1, \mathrm{id}), (q_2, \pi_1[{\ell_1}^2]), (q_2, \pi_2[{\ell_2}^3])\}.$$

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

Definition 7 (Semantics of condition automata)**.** Let $\mathcal{G} = (\mathbf{N}, \mathbf{E})$ be a graph and let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a condition automaton.

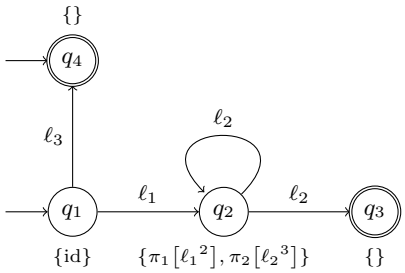

Figure 3. An example of a condition automaton.

Navigational language	Class of condition automata
$\mathcal{N}()$	$\{+, \pi, \overline{\pi}\}$-free and loop-free.
$\mathcal{N}(\pi)$	$\{+, \overline{\pi}\}$-free and loop-free.
$\mathcal{N}(\pi, \overline{\pi})$	$\{+\}$-free and loop-free.
$\mathcal{N}(+)$	$\{\pi, \overline{\pi}\}$-free.
$\mathcal{N}(+, \pi)$	$\{\overline{\pi}\}$-free.
$\mathcal{N}(+, \pi, \overline{\pi})$	no restrictions.

Table 1. Navigational languages and the corresponding class of condition automata.

We define the *condition expression* of state $q \in S$, denoted by $\bullet(q)$, as follows:

$$\bullet(q) = \begin{cases} \mathrm{id} & \text{if } \gamma(q) = \emptyset; \\ c_1 \circ \ldots \circ c_n & \text{if } \gamma(q) = \{c_1, \ldots, c_n\}. \end{cases}$$

Since each $c \in \gamma(q)$ is a condition, the particular ordering of $c_1, \ldots, c_n$ in the construction of $\bullet(q)$ does not matter: each ordering of the conditions will produce navigational expressions that are path-equivalent to each other. We say that a node $n \in \mathbf{N}$ *satisfies* state $q \in S$ if $(n, n) \in \bullet(q)\langle\mathcal{G}\rangle$.

A *run* of $\mathcal{A}$ on $\mathcal{G}$ is a sequence

$$(q_0, n_0)\ell_0(q_1, n_1)\ell_1 \ldots (q_{i-1}, n_{i-1})\ell_{i-1}(q_i, n_i),$$

where $q_0, \ldots, q_i \in S$, $n_0, \ldots, n_i \in \mathbf{N}$, $\ell_0, \ldots, \ell_i \in \Sigma \cup \{\mathrm{id}\}$, and the following conditions hold:

1. for all $0 \leq j \leq i$, n_j satisfies q_j;
2. for all $0 \leq j < i$, $(q_j, \ell_j, q_{j+1}) \in \delta$; and
3. for all $0 \leq j < i$, $(n_j, n_{j+1}) \in \ell_j\langle\mathcal{T}\rangle$.

We say that $\mathcal{A}$ *accepts* node pair $(m, n) \in \mathbf{N} \times \mathbf{N}$ if there exists a run $(q_0, m)\ell_0 \ldots (q_i, n)$ of $\mathcal{A}$ on $\mathcal{G}$ with $q_0 \in I$ and $q_i \in F$. We define the *evaluation* of $\mathcal{A}$ on $\mathcal{G}$, denoted by $\mathcal{A}\langle\!\langle\mathcal{G}\rangle\!\rangle$, as $\mathcal{A}\langle\!\langle\mathcal{G}\rangle\!\rangle = \{(m, n) \mid \mathcal{A} \text{ accepts } (m, n)\}$. Using *path query semantics*, $\mathcal{A}$ on $\mathcal{G}$ evaluates to $\mathcal{A}\langle\!\langle\mathcal{G}\rangle\!\rangle$, and using *boolean query semantics*, $\mathcal{A}$ on $\mathcal{G}$ evaluates to the truth value of $\mathcal{A}\langle\!\langle\mathcal{G}\rangle\!\rangle \neq \emptyset$.

Example 4. Consider the condition automaton of Example 3, shown in Figure 3. By carefully examining the automaton, one can conclude that it is path-equivalent to the navigational expression

$$\ell_1 \circ \pi_1[{\ell_1}^2] \circ \pi_2[{\ell_2}^3] \circ [\ell_2 \circ \pi_1[{\ell_1}^2] \circ \pi_2[{\ell_2}^3]]^* \circ \ell_2 \cup \ell_3 \cup \mathrm{id}.$$

Our first goal is to show the path-equivalence of $\mathcal{N}(\mathfrak{F})$, $\mathfrak{F} \subseteq \{+, \pi, \overline{\pi}\}$, with a restricted class of condition automata, as summarized in Table 1. To this end, we first adapt standard closure properties for finite automata under composition, union, and Kleene plus to the setting of condition automata:

Proposition 4. *Let $\mathfrak{F} \in \{+, \pi, \overline{\pi}\}$ and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be $\mathfrak{F}$-free condition automata. There exists $\mathfrak{F}$-free condition automata $\mathcal{A}_\circ$, $\mathcal{A}_\cup$, and $\mathcal{A}_+$ such that, for every tree $\mathcal{T}$, we have $\mathcal{A}_\circ\langle\!\langle\mathcal{T}\rangle\!\rangle =$*

e	Condition automaton
$\emptyset$	$\mathcal{A} = (\{v, w\}, \Sigma, \emptyset, \{v\}, \{w\}, \emptyset, \emptyset)$
id	$\mathcal{A} = (\{v, w\}, \Sigma, \emptyset, \{v\}, \{w\}, \{(v, \mathrm{id}, w)\}, \emptyset)$
ℓ	$\mathcal{A} = (\{v, w\}, \Sigma, \emptyset, \{v\}, \{w\}, \{(v, \ell, w)\}, \emptyset)$
$\pi_1[e']$	$\mathcal{A} = (\{v\}, \Sigma, \{\pi_1[e']\}, \{v\}, \{v\}, \emptyset, \{(v, \pi_1[e'])\})$
$\pi_2[e']$	$\mathcal{A} = (\{v\}, \Sigma, \{\pi_2[e']\}, \{v\}, \{v\}, \emptyset, \{(v, \pi_2[e'])\})$
$\overline{\pi}_1[e']$	$\mathcal{A} = (\{v\}, \Sigma, \{\overline{\pi}_1[e']\}, \{v\}, \{v\}, \emptyset, \{(v, \overline{\pi}_1[e'])\})$
$\overline{\pi}_2[e']$	$\mathcal{A} = (\{v\}, \Sigma, \{\overline{\pi}_2[e']\}, \{v\}, \{v\}, \emptyset, \{(v, \overline{\pi}_2[e'])\})$

Table 2. Basic building blocks used by the translation from navigational expressions to condition automata. In the table, ℓ is an edge-label.

$\mathcal{A}_1\langle\!\langle\mathcal{T}\rangle\!\rangle \circ \mathcal{A}_2\langle\!\langle\mathcal{T}\rangle\!\rangle$, $\mathcal{A}_\cup\langle\!\langle\mathcal{T}\rangle\!\rangle = \mathcal{A}_1\langle\!\langle\mathcal{T}\rangle\!\rangle \cup \mathcal{A}_2\langle\!\langle\mathcal{T}\rangle\!\rangle$, and $\mathcal{A}_+\langle\!\langle\mathcal{T}\rangle\!\rangle = [\mathcal{A}_1\langle\!\langle\mathcal{T}\rangle\!\rangle]^+$. The condition automata $\mathcal{A}_\circ$ and $\mathcal{A}_\cup$ are loop-free whenever $\mathcal{A}_1$ and $\mathcal{A}_2$ are loop-free.

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

Proof (sketch). Let Σ be the set of all relevant edge-labels. Let e be a navigational expression in $\mathcal{N}(\mathfrak{F})$. 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 $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a condition automaton. Let $v, w \notin S$ be two distinct fresh states. Let $\mathcal{A}' = (S \cup \{v, w\}, \Sigma, C, \{v\}, \{w\}, \delta \cup \delta_{v,w}, \gamma)$ with $\delta_{v,w} = \{(v, \mathrm{id}, q) \mid q \in I\} \cup \{(q, \mathrm{id}, w) \mid q \in F\}$ be a condition automaton that is path-equivalent to $\mathcal{A}$ and having only a single initial state and a single final state. We translate $\mathcal{A}'$ into a navigational expression using Algorithm 1.

Algorithm 1 From condition automaton to navigational expression

1: We mark each state $q \in S \cup \{v, w\}$: $M[q_i] :=$ `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

$$e_{q,r} := \bigcup_{(q,\ell,r) \in \delta \cup \delta_{v,w}} \bullet(q) \circ \ell \circ \bullet(r),$$

with $e_{q,r} := \emptyset$ if there are no transitions between q and r
3: **while** $\exists q\ (q \in S) \wedge (M[q] =$ `true`$)$ **do**
4: Choose q with $(q \in S) \wedge (M[q] =$ `true`$)$
5: **for** $p_1, p_2 \in S \cup \{v, w\}$ with $q \notin \{p_1, p_2\}$ **do**
6: $e_{p_1,p_2} := e_{p_1,p_2} \cup e_{p_1,q} \circ [e_{q,q}]^* \circ e_{q,p_2}$
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: $M[q] :=$ `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 $\mathcal{N}(\mathfrak{F})$.
2. If $(m, n) \in e_{q_1,q_2}\langle\mathcal{T}\rangle$, with $q_1, q_2 \in S \cup \{v, w\}$, then there exists a run $(q_1, m)\ell \ldots (q_2, n)$ of $\mathcal{A}'$ on $\mathcal{T}$ that performs at least one transition.
3. If $(q_1, n_1)\ell_1 \ldots (q_i, n_i)$ is a run of $\mathcal{A}'$ on $\mathcal{T}$ with $M[q_2] = \cdots = M[q_{i-1}] =$ `false` that performs at least one transition, then we have $(n_1, n_i) \in e_{q_1,q_i}\langle\mathcal{T}\rangle$.

Figure 4. Two simple graphs. Only on the graph to the left, the navigational expression $\ell^2 \cap \ell$ evaluates to `true`.

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

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 $(q, \{q, q_1, \ldots, q_i\})$ is an *identity pair* of $\mathcal{A}$ if there exists a path $q\mathrm{id}q'_1 \ldots \mathrm{id}q'_j$ in $\mathcal{A}$ with $\{q, q_1, \ldots, q_i\} = \{q, q'_1, \ldots, q'_j\}$.

Lemma 5. *Let $\mathfrak{F} \in \{+, \pi, \overline{\pi}\}$ and let $\mathcal{A}$ be an $\mathfrak{F}$-free condition automaton. There exists an* id*-transition free, $\mathfrak{F}$-free condition automaton $\mathcal{A}_{\not\mathrm{id}}$ that is path-equivalent to $\mathcal{A}$. The condition automaton $\mathcal{A}_{\not\mathrm{id}}$ is loop-free whenever $\mathcal{A}$ is loop-free.*

Proof (sketch). Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be an $\mathfrak{F}$-free condition automaton. We construct $\mathcal{A}_{\not\mathrm{id}} = (S_{\not\mathrm{id}}, \Sigma, C, I_{\not\mathrm{id}}, F_{\not\mathrm{id}}, \delta_{\not\mathrm{id}}, \gamma_{\not\mathrm{id}})$ with

$$S_{\not\mathrm{id}} = \{(q, Q) \mid (q, Q) \text{ is an identity pair of } \mathcal{A}\};$$
$$I_{\not\mathrm{id}} = \{(q, Q) \mid ((q, Q) \in S_{\not\mathrm{id}}) \wedge (q \in I)\};$$
$$F_{\not\mathrm{id}} = \{(q, Q) \mid ((q, Q) \in S_{\not\mathrm{id}}) \wedge (Q \cap F \neq \emptyset)\};$$
$$\delta_{\not\mathrm{id}} = \{((p, P), \ell, (q, Q)) \mid ((p, P) \in S_{\not\mathrm{id}}) \wedge (\ell \in \Sigma) \wedge ((q, Q) \in S_{\not\mathrm{id}}) \wedge (\exists p'\ (p' \in P) \wedge (p', \ell, q) \in \delta)\};$$
$$\gamma_{\not\mathrm{id}} = \{((q, Q), c) \mid ((q, Q) \in S_{\not\mathrm{id}}) \wedge (\exists q'\ (q' \in Q) \wedge (c \in \gamma(q')))\}.$$ □

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. For 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 $\ell^2 \cap \ell$ 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 ℓ^2 and ℓ to finite automata and construct their cross-product, the resulting automaton would represent the empty language, which is obviously not equivalent to $\ell^2 \cap \ell$ 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 $\mathcal{A}_1$ and $\mathcal{A}_2$ be* id*-transition free condition automata and let $\mathcal{T} = (\mathbf{N}, \mathbf{E})$ be a tree. If there exists a run $(p_1, n_1)\ell^1_1 \ldots \ell^1_{i_1}(q_1, n_{i_1+1})$ of $\mathcal{A}_1$ on $\mathcal{T}$ and there exists a run $(p_2, m_1)\ell^2_1 \ldots \ell^2_{i_2}(q_2, m_{i_2+1})$ of $\mathcal{A}_2$ on $\mathcal{T}$ with $n_1 = m_1$ and*

$n_{i_1+1} = m_{i_2+1}$, *then* $i_1 = i_2 = i$ *and, for all* $1 \leq j \leq i$, $\ell_j^1 = \ell_j^2$ *and* $n_j = m_j$.

This allows us to prove the following:

Proposition 6. *Let* $\mathfrak{F} \in \{+, \pi, \overline{\pi}\}$ *and let* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *be* $\mathfrak{F}$*-free condition automata. There exists an* $\mathfrak{F}$*-free condition automaton* $\mathcal{A}_\cap$ *such that, for every tree* $\mathcal{T}$*, we have* $\mathcal{A}_\cap\langle\!\langle\mathcal{T}\rangle\!\rangle = \mathcal{A}_1\langle\!\langle\mathcal{T}\rangle\!\rangle \cap \mathcal{A}_2\langle\!\langle\mathcal{T}\rangle\!\rangle$*. The condition automaton* $\mathcal{A}_\cap$ *is loop-free whenever* $\mathcal{A}_1$ *or* $\mathcal{A}_2$ *is loop-free.*

Proof (sketch). Let $\mathcal{A}_1 = (S_1, \Sigma_1, C_1, I_1, F_1, \delta_1, \gamma_1)$ and $\mathcal{A}_2 = (S_2, \Sigma_2, C_2, I_2, F_2, \delta_2, \gamma_2)$ be condition automata. By Lemma 5, we may assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ are id-transition free. Without loss of generality, we may also assume that $S_1 \cap S_2 = \emptyset$. We construct $\mathcal{A}_\cap = (S_1 \times S_2, \Sigma_1 \cup \Sigma_2, C_1 \cup C_2, I_1 \times I_2, F_1 \times F_2, \delta_\cap, \gamma_\cap)$ where

$$\delta_\cap = \{((p_1, q_1), \ell, (p_2, q_2)) \mid (p_1, \ell, p_2) \in \delta_1 \wedge (q_1, \ell, q_2) \in \delta_2\};$$
$$\gamma_\cap = \{((p, q), c) \mid c \in \gamma_1(p) \vee c \in \gamma_2(q)\}.$$

Let $\mathcal{T} = (\mathbf{N}, \mathbf{E})$ be a tree and let $v, w \in \mathbf{N}$ be a pair of nodes. Let $(p_1, n_1)\ell_1^1 \ldots \ell_{i_1}^1 (q_1, n_{i_1+1})$ and $(p_2, m_1)\ell_1^2 \ldots \ell_{i_2}^2 (q_2, m_{i_2+1})$ be runs of $\mathcal{A}_1$ on $\mathcal{T}$ and $\mathcal{A}_2$ on $\mathcal{T}$, respectively, with $p_1 \in I_1$, $q_1 \in F_1$, $p_2 \in I_2$, and $q_2 \in F_2$, in which $v = n_1 = m_1$ and $w = n_{i_1+1} = m_{i_2+1}$. Using Lemma 6, we conclude that these runs exist if and only if $i = i_1 = i_2$ and, for all $1 \leq j \leq i$, $l_j = \ell_j^1 = \ell_j^2$. But then, there also exists a run $((p_1, p_2), v)\ell_1 \ldots \ell_i((q_1, q_2), w)$ of $\mathcal{A}_\cap$ on $\mathcal{T}$ with $(p_1, p_2) \in I_1 \times I_2$ and $(q_1, q_2) \in F_1 \times F_2$. □

Usually, the difference of two finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ is constructed by first constructing the complement of $\mathcal{A}_2$, and then constructing the intersection of $\mathcal{A}_1$ 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 $\mathcal{T} = (\mathbf{N}, \mathbf{E})$ be a tree. We define the downward complement of a binary relation $\mathrm{R} \subseteq \mathbf{N} \times \mathbf{N}$, denoted by $\overline{\mathrm{R}}_\downarrow$, as

$$\overline{\mathrm{R}}_\downarrow = \{(m, n) \mid (m, n) \notin \mathrm{R} \wedge (m, n) \in [\epsilon]^*\langle\mathcal{T}\rangle\},$$

in which $\epsilon = \bigcup_{\ell \in \Sigma} \ell$. Indeed, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are condition automata and $\mathcal{T}$ is a tree, then we have $\mathcal{A}_1\langle\!\langle\mathcal{T}\rangle\!\rangle \setminus \mathcal{A}_2\langle\!\langle\mathcal{T}\rangle\!\rangle \equiv \mathcal{A}_1\langle\!\langle\mathcal{T}\rangle\!\rangle \cap \overline{\mathcal{A}_2\langle\!\langle\mathcal{T}\rangle\!\rangle}_\downarrow$. 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 $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ is *deterministic* if it is id-transition free and if it satisfies the following condition: for every tree $\mathcal{T} = (\mathbf{N}, \mathbf{E})$ and for every pair of nodes m, n with m an ancestor of n, there exists exactly one run $(q, m)\ell \ldots (p, n)$ of $\mathcal{A}$ on $\mathcal{T}$ with $q \in I$.[2]

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 $\pi_2[{\ell_1}^3]$ 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 $\pi_2[{\ell_1}^3]$.

[2] If we make abstraction of the condition expressions, then Definition 9 reduces to the classical definition of a finite deterministic automaton.

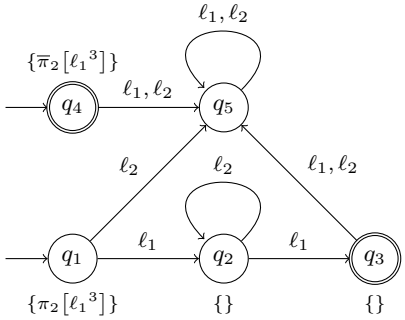

Figure 5. An example of a deterministic condition automaton.

In the construction of deterministic condition automata we shall use the *condition-complement* of a condition, defined as follows:

$$\mathrm{ccompl}(e) = \begin{cases} \emptyset & \text{if } e = \mathrm{id}; \\ \mathrm{id} & \text{if } e = \emptyset; \\ \overline{\pi}_1[e'] & \text{if } e = \pi_1[e']; \\ \overline{\pi}_2[e'] & \text{if } e = \pi_2[e']; \\ \pi_1[e'] & \text{if } e = \overline{\pi}_1[e']; \\ \pi_2[e'] & \text{if } e = \overline{\pi}_2[e']. \end{cases}$$

Observe that the condition complement of a projection expression is a coprojection expression, and vice-versa. We extend the definition of condition-complement to sets of conditions: if S is a set of conditions, then $\mathrm{ccompl}(S) = \{\mathrm{ccompl}(e) \mid e \in S\}$.

Lemma 7. *Let* $\mathfrak{F} \in \{+, \pi, \overline{\pi}\}$ *and let* $\mathcal{A}$ *be an* $\mathfrak{F}$*-free condition automaton. There exists a deterministic condition automaton* $\mathcal{A}_D$ *that is path-equivalent to* $\mathcal{A}$*. The condition automaton* $\mathcal{A}_D$ *is* $\{+\}$*-free if* $+ \notin \mathfrak{F}$ *and* $\{\pi, \overline{\pi}\}$*-free if* $\pi, \overline{\pi} \notin \mathfrak{F}$*.*

Proof (sketch). Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a condition automaton. By Lemma 5, we can assume that $\mathcal{A}$ is id-transition free. We construct $\mathcal{A}_D = (S_D, \Sigma, C \cup \mathrm{ccompl}(C), I_D, F_D, \delta_D, \gamma_D)$, where S_D, I_D, and δ_D are constructed by Algorithm 2, and

$$F_D = \{(Q, V) \mid (Q, V) \in S_D \wedge Q \cap F \neq \emptyset\};$$
$$\gamma_D = \{((Q, V), c) \mid (Q, V) \in S_D \wedge (c \in V \vee c \in \mathrm{ccompl}((C - V)))\}.$$

Both determinism of $\mathcal{A}_D$ and path-equivalence of $\mathcal{A}_D$ and $\mathcal{A}$ are guaranteed, as this construction satisfies the following properties:

1. There exists exactly one $V \subseteq C$ such that
$$(n, n) \in \bullet(V \cup \mathrm{ccompl}((C \setminus V)))\langle\mathcal{T}\rangle.$$
2. There exists exactly one state $(P, V) \in I_D$ such that m satisfies (P, V).
3. Let $(P, V) \in S_D$ be a state such that m satisfies (P, V). If there exists a directed path from m to n, then there exists exactly one run $((P, V), m)\ell \ldots ((Q, W), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$.
4. If there exists a run $(q_1, m)\ell \ldots (q_i, n)$ of $\mathcal{A}$ on $\mathcal{T}$ with $q_1 \in I$, then there exists a run $((Q_1, V_1), m)\ell \ldots ((Q_i, V_i), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$ with $(Q_1, V_1) \in I_D$, and, for all $1 \leq j \leq i$, $q_j \in Q_j$.
5. If there exists a run $((Q_1, V_1), m)\ell \ldots ((Q_i, V_i), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$ with $Q_i \neq \emptyset$, then, for all $q_i \in Q_i$, there exists a run $(q_1, m)\ell \ldots (q_i, n)$ of $\mathcal{A}$ on $\mathcal{T}$ with, for all $1 \leq j < i$, $q_j \in Q_j$. □

Algorithm 2 Translation to deterministic condition automaton

Let S_D, I_D, and *new* be empty sets of states
Let δ_D be an empty transition relation
for $V \subseteq C$ **do**
  $Q := \{q \mid q \in I \wedge \gamma(q) \subseteq V\}$
  Add new state (Q, V) to S_D, I_D, and *new*
end for
while *new* $\neq \emptyset$ **do**
  Take and remove state (Q, V) from *new*
  for $\ell \in \Sigma$ **do**
    $P := \{p \mid \exists q\ q \in Q \wedge (q, \ell, p) \in \delta\}$
    for $W \subseteq C$ **do**
      $P' := \{p \mid p \in P \wedge \gamma(p) \subseteq W\}$
      if $(P', W) \notin S_D$ **then**
        Add new state (P', W) to S_D and *new*
      end if
      Add new transition $((Q, V), \ell, (P', W))$ to δ_D
    end for
  end for
end while

Proposition 7. *Let $\mathfrak{F} \in \{+, \pi, \overline{\pi}\}$ and let $\mathcal{A}$ be an $\mathfrak{F}$-free condition automaton. There exists a condition automaton $\mathcal{A}'$ such that, for every tree $\mathcal{T}$, we have $\mathcal{A}'\langle\!\langle \mathcal{T} \rangle\!\rangle = \overline{\mathcal{A}\langle\!\langle \mathcal{T} \rangle\!\rangle}_{\downarrow}$. The condition automaton $\mathcal{A}'$ is $\{+\}$-free if $+ \notin \mathfrak{F}$ and $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \notin \mathfrak{F}$.*

Proof. Let $\mathcal{A}'' = (S'', \Sigma'', C'', I'', F'', \delta'', \gamma'')$ be the deterministic condition automaton equivalent to $\mathcal{A}$. We construct $\mathcal{A}' = (S'', \Sigma'', C'', I'', S'' \setminus F'', \delta'', \gamma'')$. □

Corollary 3. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be condition automata. There exists a condition automaton $\mathcal{A}_\setminus$ such that, for every tree $\mathcal{T}$, we have $\mathcal{A}_\setminus\langle\!\langle \mathcal{T} \rangle\!\rangle = \mathcal{A}_1\langle\!\langle \mathcal{T} \rangle\!\rangle \setminus \mathcal{A}_2\langle\!\langle \mathcal{T} \rangle\!\rangle$. The condition automaton $\mathcal{A}_\setminus$ is $\{+\}$-free if $+ \notin \mathfrak{F}$, $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \notin \mathfrak{F}$, and loop-free whenever $\mathcal{A}_1$ is loop-free.*

Proof. Since $\mathcal{A}_1\langle\!\langle \mathcal{T} \rangle\!\rangle \setminus \mathcal{A}_2\langle\!\langle \mathcal{T} \rangle\!\rangle = \mathcal{A}_1\langle\!\langle \mathcal{T} \rangle\!\rangle \cap \overline{\mathcal{A}_2\langle\!\langle \mathcal{T} \rangle\!\rangle}_{\downarrow}$, we can apply Proposition 7 and Proposition 6 to construct $\mathcal{A}_\setminus$. □

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 $\mathcal{N}(+, \pi, \overline{\pi}, \cap, \setminus)$. We define the *condition-depth* of e, denoted by $\mathrm{cdepth}(e)$, as

$$\mathrm{cdepth}(e) = \begin{cases} 0 & \text{if } e \in \{\emptyset, \mathrm{id}\}; \\ 0 & \text{if } e = \ell, \text{ with } \ell \text{ an edge-label}; \\ \mathrm{cdepth}(e') & \\ \qquad & \text{if } e = [e']^+; \\ \mathrm{cdepth}(e') + 1 & \\ \qquad & \text{if } e \in \{\pi_1[e'], \pi_2[e'], \overline{\pi}_1[e'], \overline{\pi}_2[e']\}; \\ \max(\mathrm{cdepth}(e_1), \mathrm{cdepth}(e_2)) & \\ \qquad & \text{if } e \in \{e_1 \circ e_2, e_1 \cup e_2, \\ \qquad & \qquad e_1 \cap e_2, e_1 \setminus e_2\}. \end{cases}$$

Theorem 3. *Let $\mathfrak{F} \subseteq \{+, \pi, \overline{\pi}, \cap, \setminus\}$. On labeled trees we have $\mathcal{N}(\mathfrak{F}) \preceq_p \mathcal{N}(\underline{\mathfrak{F}} \setminus \{\cap, \setminus\})$.*

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 in $\mathcal{N}(\mathfrak{F})$ to condition automata. Given two condition automata representing 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 $e_1 \setminus e_2$. Finally, we use Proposition 5 to translate the resulting condition automaton back to a navigational expression in $\mathcal{N}(\underline{\mathfrak{F}} \setminus \{\cap, \setminus\})$. □

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 π-conditions from any expression in $\mathcal{N}(+, \pi)$ or in $\mathcal{N}(\pi)$.

We define the *condition-depth* of a $\{\overline{\pi}\}$-free condition automaton $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$, denoted by $\mathrm{cdepth}(\mathcal{A})$, as $\mathrm{cdepth}(\mathcal{A}) = \max\{\mathrm{cdepth}(c) \mid c \in C\}$. We define the *condition-weight* of $\mathcal{A}$, denoted by $\mathrm{cweight}(\mathcal{A})$, as

$$\mathrm{cweight}(\mathcal{A}) = |\{c \mid c \in C \wedge \mathrm{cdepth}(c) = \mathrm{cdepth}(\mathcal{A})\}|.$$

Lemma 8. *Let $\mathcal{A}$ be a $\{\overline{\pi}\}$-free and* id*-transition free condition automaton. If $\mathrm{cdepth}(\mathcal{A}) > 0$, then there exists a $\{\overline{\pi}\}$-free and* id*-transition free condition automaton $\mathcal{A}_\pi$ such that*

1. *for every labeled chain $\mathcal{C}$, we have $\mathcal{A}\langle\!\langle \mathcal{C} \rangle\!\rangle = \emptyset$ if and only if $\mathcal{A}_\pi\langle\!\langle \mathcal{C} \rangle\!\rangle = \emptyset$; and*
2. *$\mathrm{cdepth}(\mathcal{A}) > \mathrm{cdepth}(\mathcal{A}_\pi)$ or $\mathrm{cdepth}(\mathcal{A}) = \mathrm{cdepth}(\mathcal{A}_\pi) \wedge \mathrm{cweight}(\mathcal{A}) > \mathrm{cweight}(\mathcal{A}_\pi)$.*

The condition automaton $\mathcal{A}_\pi$ is loop-free and $\{+\}$-free whenever $\mathcal{A}$ is loop-free and $\{+\}$-free.

Proof (sketch). Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a $\{\overline{\pi}\}$-free and id-transition free condition automaton. Choose a condition $c \in C$ with $\mathrm{cdepth}(\mathcal{A}) = \mathrm{cdepth}(c)$. Let $\mathcal{A}' = (S', \Sigma', C', I', F', \delta', \gamma')$ be a $\{\overline{\pi}\}$-free and id-transition free condition automaton equivalent to e'. If we construct $\mathcal{A}'$ in the canonical way, we have $\mathrm{cdepth}(\mathcal{A}') = \mathrm{cdepth}(e')$. Let $\varepsilon \notin S \cup S'$ be a fresh state. We define the power set of set S, denoted by $\mathcal{P}(S)$, as $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. In the following, we use the values 1 and 2 and the variable i, $i \in \{1, 2\}$, to indicate that definitions depend on the type i of the condition $c = \pi_i[e']$. We define $\mathcal{A}_\pi = (S_\pi, \Sigma_\pi, C_\pi, I_\pi, F_\pi, \delta_\pi, \gamma_\pi)$ for $c = \pi_1[e']$ as follows:

$$\begin{aligned} S_c &= \{q \mid c \in \gamma(q)\}; \\ S_{\neg c} &= S \setminus S_c; \\ S_{\neg 1} &= \{(q, Q) \mid q \in S_c \wedge Q \subseteq S' \wedge Q \cap I' = \emptyset\}; \\ S_\pi &= \left(S \times \mathcal{P}(S')\right) \setminus S_{\neg i} \cup \{\varepsilon\} \times \left(\mathcal{P}(S') \setminus \emptyset\right); \\ \Sigma_\pi &= \Sigma; \\ C_\pi &= C \setminus \{c\} \cup C'; \\ I_1 &= \{(q, \{q'\}) \mid q \in S_c \cap I \wedge q' \in I'\}; \\ I_\pi &= \{(q, \emptyset) \mid q \in S_{\neg c} \cap I\} \cup I_i; \\ F_1 &= \{(q, Q) \mid q \in S_{\neg c} \cap F \wedge \emptyset \subset Q \subseteq F'\} \\ &\quad \cup \{(q, Q) \mid q \in S_c \cap F \wedge \emptyset \subset Q \subseteq F' \cap I'\} \\ &\quad \cup \{(\varepsilon, Q) \mid \emptyset \subset Q \subseteq F'\}; \\ F_\pi &= \{(q, \emptyset) \mid q \in S_{\neg c} \cap F\} \cup F_i; \\ \delta_{\mathcal{P}(S')} &= \{(P, \ell, Q) \mid P \subseteq S' \wedge \ell \in \Sigma_\pi \wedge Q \subseteq S' \wedge \\ &\qquad \forall p\ p \notin P \vee (\exists q\ q \in Q \wedge (p, \ell, q) \in \delta') \wedge \\ &\qquad \forall q\ q \notin Q \vee (\exists p\ p \in P \wedge (p, \ell, q) \in \delta')\}; \end{aligned}$$

$$\delta_{1,\mathrm{b}} = \{((p, P \cup P'), \ell, (q, Q)) \mid (p, P \cup P') \in S_\pi \wedge P' \subseteq F' \wedge (q, Q) \in S_\pi \wedge ((p, \ell, q) \in \delta \vee ((p = \varepsilon \vee p \in F) \wedge q = \varepsilon)) \wedge (P, \ell, Q) \in \delta_{\mathcal{P}(S')}\};$$

$$\delta_{1,c} = \{((p, P \cup P'), \ell, (q, Q \cup \{q'\})) \mid (p, P \cup P') \in S_\pi \wedge (q, Q \cup \{q'\}) \in S_\pi \wedge (p, \ell, q) \in \delta \wedge (P, \ell, Q) \in \delta_{\mathcal{P}(S')} \wedge q \in S_c \wedge q' \in I' \wedge P' \subseteq F'\};$$

$$\delta_\pi = \delta_{i,\mathrm{b}} \cup \delta_{i,c};$$

$$\gamma_\pi = \{((q, Q), c') \mid (q, Q) \in S_\pi \wedge (c' \in \gamma(q) \vee (\exists q'\, q' \in Q \wedge c' \in \gamma'(q')))\}.$$

For $c = \pi_2[e']$, 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 $\mathfrak{F} \subseteq \{+, \pi\}$. On labeled chains we have $\mathcal{N}(\mathfrak{F}) \preceq_b \mathcal{N}(\mathfrak{F} \setminus \{\pi\})$.*

Proof. We use Proposition 5 to translate a navigational expression to a condition automaton $\mathcal{A}$, then we repeatedly apply Lemma 8 to remove conditions, and, finally, we use Proposition 5 to translate back to a navigational expression in $\mathcal{N}(\mathfrak{F} \setminus \{\pi\})$. Observe that only a finite number of condition removal steps on $\mathcal{A}$ can be made, as Lemma 8 guarantees that either $\mathrm{cdepth}(\mathcal{A})$ strictly decreases or else $\mathrm{cdepth}(\mathcal{A})$ does not change and $\mathrm{cweight}(\mathcal{A})$ 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 π_2-conditions on trees, as trees do not have branching in the direction from a node to its ancestors. For π_1, this observation does not hold, as is illustrated by the proof of Proposition 3.

Proposition 8. *Let $\mathfrak{F} \subseteq \{+\}$. On labeled trees we have $\mathcal{N}(\mathfrak{F} \cup \{\pi_2\}) \preceq_b \mathcal{N}(\mathfrak{F})$, but $\mathcal{N}(\mathfrak{F} \cup \{\pi_1\}) \not\preceq_b \mathcal{N}(\mathfrak{F})$.*

3.4 Results using First-order Logic

For all $\mathcal{N}(\mathfrak{F})$ with $\mathfrak{F} \subseteq \{\pi, \overline{\pi}, \cap, \setminus\}$, it is straightforward to show that every expression in $\mathcal{N}(\mathfrak{F})$ can be expressed by a first-order logic formula over the structure $(\mathbf{N}; \mathbf{E})$. Moreover, as $\mathcal{N}(\mathfrak{F})$ is essentially a fragment of the calculus of relations, every expression can be expressed in FO[3], the language of first-order logic formulae using at most three variables [13, 22]. Exploring this relationship yields the following two results involving $+$:

Proposition 9. *Let $\mathfrak{F} \subseteq \{\pi, \overline{\pi}, \cap, \setminus\}$. On unlabeled chains we have $\mathcal{N}(+) \not\preceq_p \mathcal{N}(\mathfrak{F})$, and on labeled chains we have $\mathcal{N}(+) \not\preceq_b \mathcal{N}(\mathfrak{F})$.*

Proof. Using well-known results on the expressive power of first-order logic [18], we conclude that no navigational expression in $\mathcal{N}(\mathfrak{F})$ is path-equivalent to $[\mathcal{E}]^+$ and that no navigational expression is boolean-equivalent to $\ell_1 \circ [\ell_2 \circ \ell_2]^+ \circ \ell_1$. □

Proposition 10. *Let $+ \notin \mathfrak{F}$. On unlabeled chains and trees we have $\mathcal{N}(\mathfrak{F} \cup \{+\}) \preceq_b \mathcal{N}(\mathfrak{F})$ if and only if $\overline{\pi} \notin \underline{\mathfrak{F}}$.*

Proof. For the cases with $\overline{\pi} \notin \underline{\mathfrak{F}}$, we refer to Theorem 2. If $\overline{\pi} \in \underline{\mathfrak{F}}$, then we conclude that no navigational expression in $\mathcal{N}(\mathfrak{F})$ is boolean-equivalent to $\overline{\pi}_2[\mathcal{E}] \circ [\mathcal{E} \circ \mathcal{E}]^+ \circ \overline{\pi}_1[\mathcal{E}]$, using well-known results on the expressive power of first-order logic [18]. □

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 π_1-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 $\overline{\pi}$ and $\setminus$ and allow only for a very restricted form of transitive closure, required to define the `ancestor` and `descendant` axis. Hence, queries such as $[\ell \circ \ell]^+$ and $\ell_1 \circ [\ell_2 \circ \ell_2]^+ \circ \ell_1$, 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 $\mathscr{X}_{r,[]}$ and its fragments, are strictly less expressive than the navigational query language $\mathcal{N}(+, \pi_1)$. Furthermore, we observe that $\mathscr{X}$ is path-equivalent to $\mathcal{N}()$ and that $\mathscr{X}_{[]}$ is path-equivalent to $\mathcal{N}(\pi_1)$. 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$^{\approx}$ [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

$$\exists n \exists c_1 \exists c_2 \exists c_3 \exists c_4\, \mathcal{E}(n, c_1) \wedge \mathcal{E}(n, c_2) \wedge \mathcal{E}(n, c_3) \wedge \mathcal{E}(n, c_4) \wedge (c_1 \neq c_2) \wedge (c_1 \neq c_3) \wedge (c_1 \neq c_4) \wedge (c_2 \neq c_3) \wedge (c_2 \neq c_4) \wedge (c_3 \neq c_4),$$

which is true on all trees that have a node with at least four distinct children. Although $\mathcal{N}(+, \pi, \overline{\pi}, \cap, \setminus)$ is not a fragment of FO[3], due to the inclusion of the transitive closure operator, a straightfroward brute-force argument shows that not even $\mathcal{N}(+, \pi, \overline{\pi}, \cap, \setminus)$ 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 $\texttt{sibling} \circ \texttt{sibling} \circ \texttt{sibling}$.

Due to these differences in the tree data models used, the closure properties under intersection and complementation for Conditional XPath and Regular XPath$^{\approx}$ 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$^{\approx}$ could be translated to our setting, then 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 $\mathcal{N}(\pi, \cap, \setminus)$, and the positive algebra of Wu et al. [26], when restricted to the downward fragment, corresponds to the navigational query language $\mathcal{N}(\pi, \cap)$. 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 $\setminus$ and di. We conjecture, for example, that $\mathcal{N}(+, \setminus, \mathrm{di}) \not\leq_b \mathcal{N}(+, \overline{\pi}, \cap, \mathrm{di})$, 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 π_1 or only π_2 may affect the expressive power. Indeed, various XPath fragments and the nested RPQs only provide operators similar to π_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 $\mathcal{N}(+, \pi, \cap)$ on unlabeled trees already hint at a significant overlap of FO[3] and FO[2] for boolean queries: the query $\mathcal{E}^k$ can easily be expressed by FO[2] algebras with semijoins via $\mathcal{E} \ltimes \cdots \ltimes \mathcal{E}$. 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

[1] P. Barceló. Querying graph databases. In *Proceedings of the 32nd Symposium on Principles of Database Systems*, PODS '13, pages 175–188. ACM, 2013.

[2] M. Benedikt and C. Koch. XPath leashed. *ACM Computing Surveys (CSUR)*, 41(1):3:1–3:54, 2009.

[3] M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. *Theoretical Computer Science*, 336(1):3–31, 2005.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible markup language (XML) 1.1 (second edition). W3C recommendation, W3C, 2006. `http://www.w3.org/TR/2006/REC-xml11-20060816`.

[5] J. Clark and S. DeRose. XML path language (XPath) version 1.0. W3C recommendation, W3C, 1999. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[6] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 1970.

[7] Ecma International. The JSON data interchange format, 1st edition / october 2013, 2013. `http://www.ecma-international.org/publications/standards/Ecma-404.htm`.

[8] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 197–207. ACM, 2011.

[9] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. The impact of transitive closure on the boolean expressiveness of navigational query languages on graphs. In *Foundations of Information and Knowledge Systems*, volume 7153 of *Lecture Notes in Computer Science*, pages 124–143. Springer Berlin Heidelberg, 2012.

[10] G. H. L. Fletcher, M. Gyssens, D. Leinders, D. Surinx, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. *Information Sciences*, 298:390–406, 2015.

[11] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. The impact of transitive closure on the expressiveness of navigational query languages on unlabeled graphs. *Annals of Mathematics and Artificial Intelligence*, 73(1-2):167–203, 2015.

[12] G. H. L. Fletcher, M. Gyssens, J. Paredaens, D. Van Gucht, and Y. Wu. Structural characterizations of the navigational expressiveness of relation algebras on a tree. *Journal of Computer and System Science*, to appear, 2015.

[13] S. Givant. The calculus of relations as a foundation for mathematics. *Journal of Automated Reasoning*, 37(4):277–322, 2006.

[14] E. Grädel and M. Otto. On logics with two variables. *Theoretical Computer Science*, 224(1–2):73–113, 1999.

[15] M. Gyssens, J. Paredaens, D. Van Gucht, and G. H. L. Fletcher. Structural characterizations of the semantics of XPath as navigation tool on a document. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 318–327. ACM, 2006.

[16] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321`.

[17] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3th edition*. Pearson, 2007.

[18] L. Libkin. *Elements of Finite Model Theory*. Springer Berlin Heidelberg, 2004.

[19] M. Marx. Conditional XPath. *ACM Transactions on Database Systems*, 30(4):929–959, 2005.

[20] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005.

[21] G. Schreiber and Y. Raimond. RDF 1.1 primer. W3C working group note, W3C, 2014. `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624`.

[22] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

[23] B. ten Cate. The expressivity of XPath with transitive closure. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 328–337. ACM, 2006.

[24] B. ten Cate and M. Marx. Navigational XPath: Calculus and algebra. *SIGMOD Record*, 36(2):19–26, 2007.

[25] D. C. Tsichritzis and F. H. Lochovsky. Hierarchical data-base management: A survey. *ACM Computing Surveys (CSUR)*, 8(1):105–123, 1976.

[26] Y. Wu, D. Van Gucht, M. Gyssens, and J. Paredaens. A study of a positive fragment of path queries: Expressiveness, normal form and minimization. *The Computer Journal*, 54(7):1091–1118, 2011.