Comparing the Expressiveness of Downward Fragments of the Relation Algebra with Transitive Closure on Trees[☆]

Jelle Hellings[a,b], Marc Gyssens[a,*], Yuqing Wu[c], Dirk Van Gucht[d], Jan Van den Bussche[a], Stijn Vansummeren[e], George H. L. Fletcher[f]

[a]*Hasselt University, Martelarenlaan 42, Hasselt, Belgium*
[b]*University of California, Davis, CA 95616-8562, USA*
[c]*Pomona College, 185 E 6th St., Claremont, CA 91711, USA*
[d]*Indiana University, 919 E 10th St, Bloomington, IN 47408, USA*
[e]*Université Libre de Bruxelles, Avenue Franklin Roosevelt 50, Brussels, Belgium*
[f]*Eindhoven University of Technology, De Groene Loper 5, Eindhoven, the Netherlands*

Abstract

Motivated by the continuing interest in the tree data model, we study the expressive power of downward navigational query languages on trees and chains. Basic navigational queries are built from the identity relation and edge relations using composition and union. We study the effects on relative expressiveness when we add transitive closure, projections, coprojections, intersection, and difference; this for Boolean queries and path queries on labeled and unlabeled structures. In all cases, we present the complete Hasse diagram. In particular, we establish, for each query language fragment that we study on trees, whether it is closed under difference and intersection.

Keywords: tree data model, relational calculus with transitive closure, downward query language fragments, path queries, Boolean queries, relative expressive power

1. Introduction

Many relations between data can be described intuitively in a hierarchical way, including 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. To illustrate this, consider the hierarchical class-structure of a program described by relations *subclass* and *method*, an example

[☆] This is a revised and extended version of the conference paper 'Relative Expressive Power of Downward Fragments of Navigational Query Languages on Trees and Chains', presented at the 15th International Symposium on Database Programming Languages, Pittsburgh, Pennsylvania, United States (DBPL 2015).

[*] Corresponding author, marc.gyssens@uhasselt.be

of which is visualized by the tree in Figure 1. Given the naturalness of trees to represent hierarchically structured data, it is not surprising that tree-based data models have been continuously studied since the 1960s in the form of the hierarchical data model [1], XML [2] and, more recently, JSON [3]. In addition, the tree data model is a specialization of the more general graph data model (e.g. [4–6]).

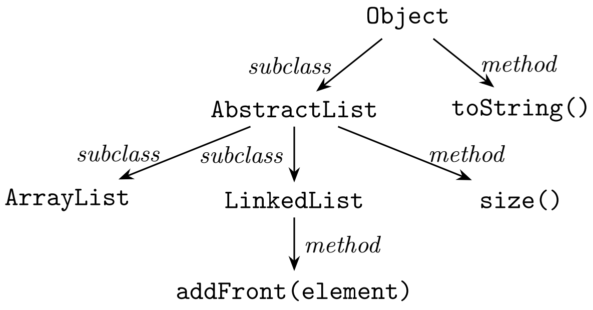

Figure 1: The hierarchical relations within typical list-classes in a Java-like object-oriented programming language.

Query languages for tree and graph data usually rely on navigating the tree structure to find data of interest. For tree data, this navigation is often in a top-down fashion, which we refer to as *downward navigation*. In the JSON data model, for example, most data retrieval is done by explicit top-down traversal of a data structure representation of the JSON data. Even in more declarative settings, such as within the PostgreSQL relational database system, the JSON query facilities primarily aim at downward navigation.[1] This focus on downward navigation is also found outside the setting of JSON data. As examples, we mention XPath [7–10] in which many queries rely on navigating the parent-child axis, the nested relational database models that use downward navigation via nesting and unnesting as an important tool to query the data (see, e.g., [11]), and graph query languages such as SPARQL [12, 13] and the regular path queries (RPQs) [14].

The core navigational power of query languages can be captured by fragments of the calculus of relations, popularized by Tarski, extended with transitive closure [15, 16]. In the form of the navigational query languages of Fletcher et al. [5], the relative expressive power of these fragments have been studied in full detail on graph-structured data [17–19]. On graph-structured data, Fletcher et al. showed that only language fragments that can express the same operators via straightforward rewriting rules have the same expressive power. Surprisingly, much less is known for the more restrictive tree data model, however. In

[1] For details on what PostgreSQL provides, we refer to `https://www.postgresql.org/docs/10/static/functions-json.html`. Observe that all basic arrow operators provided by PostgreSQL perform, in essence, downward navigation.

particular, the separation results on graphs of Fletcher et al. do not necessarily also apply to trees. Moreover, the expressiveness results for several XPath fragments [8, 9, 20–24] in the context of XML only provide an incomplete picture of the relative expressive power of the navigational query languages we consider here.

As a first step towards a more complete picture on querying tree data, we study the expressive power of downward navigation. To do so, we start with a basic query language in which queries are built using edge relations, the identity relation (id), composition ($\circ$), and union ($\cup$). We study the effect on the expressive power if we add projections (π), which can be used to express conditions similar to the node-expressions in XPath [8] and the branching operator in nested RPQs [14]; coprojections ($\overline{\pi}$), which can be used to express negated conditions; intersection ($\cap$); difference ($-$); and transitive closure (*). To illustrate the power of these operators, consider the following queries:

$$\begin{aligned}
e_1 &= \mathit{subclass} \circ (\mathit{subclass} \cup \mathit{method});\\
e_2 &= \pi_1[\mathit{method}] - \pi_1[\mathit{subclass} \circ [\mathit{subclass}]^* \circ \mathit{method}];\\
e_3 &= \overline{\pi}_1[\mathit{method}];\\
e_4 &= [\mathit{subclass} \circ \pi_1[\mathit{method}]]^*.
\end{aligned}$$

In these expressions, π_1 denote projection on the first column, while $\overline{\pi}_1$ denotes projection on the complement of the first column. On the tree data of Figure 1, e_1 returns classes and the subclasses and methods defined by their subclasses, e_2 returns all classes that define their own methods while excluding classes that have descendants that define their own methods, e_3 returns all classes that do not define their own methods, and, finally, e_4 returns classes and all their descendant classes that define their own methods.

For these fragments, we study relative expressiveness for both path queries, which evaluate to a set of node pairs, and Boolean queries, which evaluate to true or false. We consider not only labeled trees, but also unlabeled trees and labeled and unlabeled chains, the reason being that most query languages are easier to analyze 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 shown in Figure 2. Some of our results extend to the diversity relation (di) and to converse ($^\frown$), which both are non-downward.[2] For completeness, we have included these extended results. In several cases, we are 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. [5].

In the cases where graphs and trees yield different expressiveness results, we

[2] For a formal definition of diversity and converse, we refer to Definition 2.1.

are able to prove collapse results. In particular, we are able to establish, 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 never changes 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 technique based on finite automata [25], 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 $-$ operations. We also use these condition automata to show that, in the Boolean case, π never adds expressive power when querying labeled chains. With homomorphism-based techniques, we finally 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. On the one hand, one can choose 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. On the other hand, a bigger set of operators allows for easier query writing 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 $-$ 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 introduce the basic notions and terminology used throughout this paper. In Section 3, we present all basic results on the relative expressive power of the downward navigational query languages, as well as some generalizations of these. The results and their generalizations are visualized in the Hasse diagrams of relative expressiveness shown in Figure 2. Observe that these diagrams include collapses involving diversity and converse, which are non-downward. In Section 4, we introduce condition automata and use them to prove the redundancies involving intersection and difference shown in Figure 2. In Section 5, we discuss related work. In Section 6, we summarize our findings and propose directions for future work.

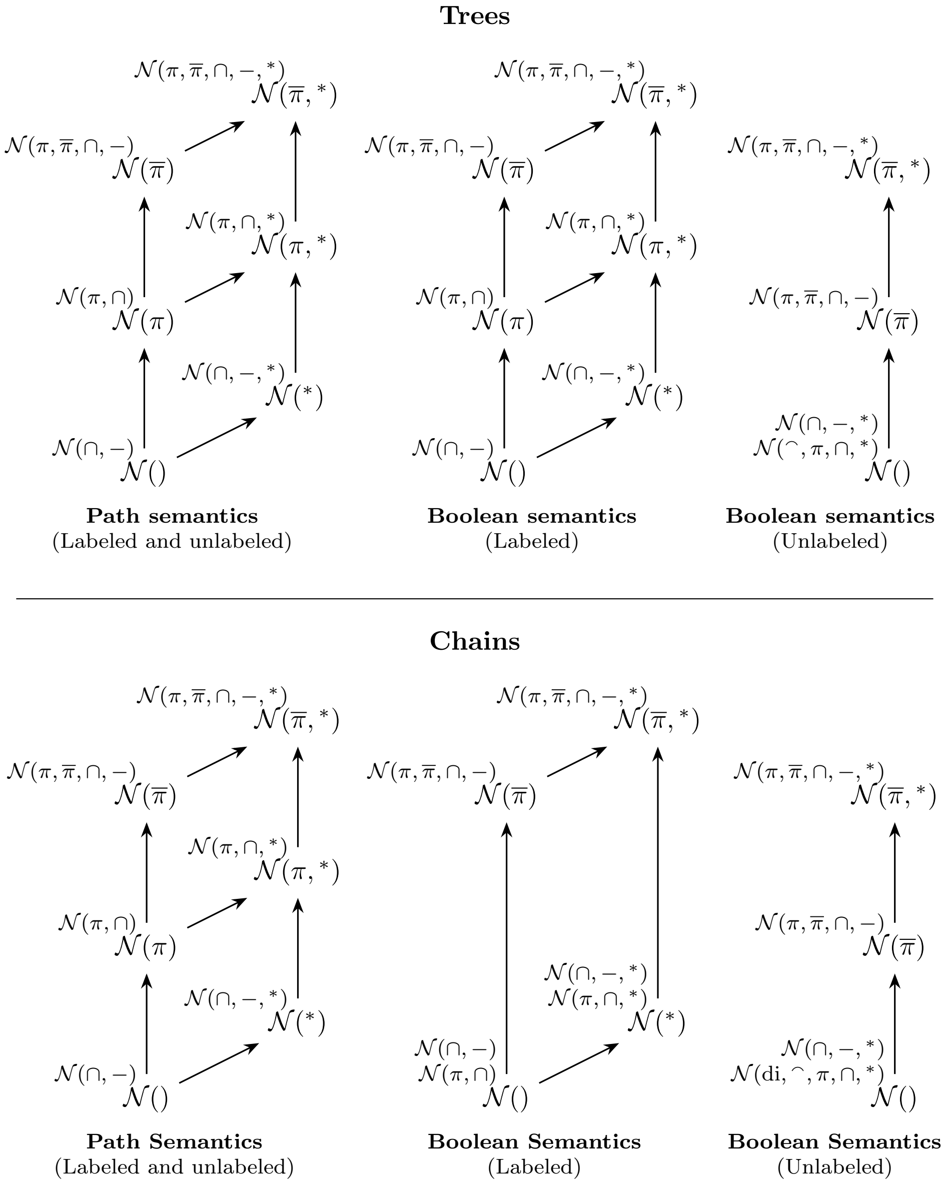
Trees
$\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$
$\mathcal{N}(\overline{\pi},^*)$
$\mathcal{N}(\pi,\overline{\pi},\cap,-)$
$\mathcal{N}(\overline{\pi})$
$\mathcal{N}(\pi,\cap,^*)$
$\mathcal{N}(\pi,^*)$
$\mathcal{N}(\pi,\cap)$
$\mathcal{N}(\pi)$
$\mathcal{N}(\cap,-,^*)$
$\mathcal{N}(^*)$
$\mathcal{N}(\cap,-)$
$\mathcal{N}()$
Path semantics
(Labeled and unlabeled)
$\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$
$\mathcal{N}(\overline{\pi},^*)$
$\mathcal{N}(\pi,\overline{\pi},\cap,-)$
$\mathcal{N}(\overline{\pi})$
$\mathcal{N}(\pi,\cap,^*)$
$\mathcal{N}(\pi,^*)$
$\mathcal{N}(\pi,\cap)$
$\mathcal{N}(\pi)$
$\mathcal{N}(\cap,-,^*)$
$\mathcal{N}(^*)$
$\mathcal{N}(\cap,-)$
$\mathcal{N}()$
Boolean semantics
(Labeled)
$\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$
$\mathcal{N}(\overline{\pi},^*)$
$\mathcal{N}(\pi,\overline{\pi},\cap,-)$
$\mathcal{N}(\overline{\pi})$
$\mathcal{N}(\cap,-,^*)$
$\mathcal{N}(^\frown,\pi,\cap,^*)$
$\mathcal{N}()$
Boolean semantics
(Unlabeled)
Chains
$\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$
$\mathcal{N}(\overline{\pi},^*)$
$\mathcal{N}(\pi,\overline{\pi},\cap,-)$
$\mathcal{N}(\overline{\pi})$
$\mathcal{N}(\pi,\cap,^*)$
$\mathcal{N}(\pi,^*)$
$\mathcal{N}(\pi,\cap)$
$\mathcal{N}(\pi)$
$\mathcal{N}(\cap,-,^*)$
$\mathcal{N}(^*)$
$\mathcal{N}(\cap,-)$
$\mathcal{N}()$
Path Semantics
(Labeled and unlabeled)
$\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$
$\mathcal{N}(\overline{\pi},^*)$
$\mathcal{N}(\pi,\overline{\pi},\cap,-)$
$\mathcal{N}(\overline{\pi})$
$\mathcal{N}(\cap,-,^*)$
$\mathcal{N}(\pi,\cap,^*)$
$\mathcal{N}(^*)$
$\mathcal{N}(\cap,-)$
$\mathcal{N}(\pi,\cap)$
$\mathcal{N}()$
Boolean Semantics
(Labeled)
$\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$
$\mathcal{N}(\overline{\pi},^*)$
$\mathcal{N}(\pi,\overline{\pi},\cap,-)$
$\mathcal{N}(\overline{\pi})$
$\mathcal{N}(\cap,-,^*)$
$\mathcal{N}(\mathrm{di},^\frown,\pi,\cap,^*)$
$\mathcal{N}()$
Boolean Semantics
(Unlabeled)

Figure 2: Hasse diagrams that visualize the relative *expressiveness* of fragments of $\mathcal{N}(\pi,\overline{\pi},\cap,-,^*)$, the downward navigational expressions. Each node represents a minimally sized fragment and the superscripts on the left-hand side represent all maximally sized fragments that are equivalent to the fragment represented by the node. Arrows represent strict subsumption relations. Notice that equivalence of $\mathcal{N}(\mathcal{F}_1)$ and $\mathcal{N}(\mathcal{F}_2)$ does not imply equivalence of $\mathcal{N}(\mathcal{F}_1 \cup \{o\})$ and $\mathcal{N}(\mathcal{F}_2 \cup \{o\})$. E.g. for path semantics on trees, $\mathcal{N}()$ and $\mathcal{N}(\cap,-)$ are equivalent, but $\mathcal{N}(\pi)$ and $\mathcal{N}(\pi,\cap,-)$ are not equivalent.

2. Preliminaries

We assume an infinitely enumerable set of labels $\mathbf{\Sigma}$. Let $\Sigma \subseteq \mathbf{\Sigma}$ be a finite set of labels, interpreted as edge labels. A *graph* over Σ is a triple $\mathcal{G} = (\mathcal{V}, \Sigma, \mathbf{E})$, with $\mathcal{V}$ a finite set of nodes and $\mathbf{E} : \Sigma \to 2^{\mathcal{V}\times\mathcal{V}}$ a function mapping edge labels to edge relations. A graph is *unlabeled* if $|\Sigma| = 1$. It is said that edge $(m, n) \in \bigcup_{\ell\in\Sigma} \mathbf{E}(\ell)$ is an *outgoing edge* of m and an *incoming edge* of n. A *path* of length k is a sequence $n_1\ell_1n_2 \cdots \ell_kn_{k+1}$ with $n_1, \dots, n_{k+1} \in \mathcal{V}$, $\ell_1, \dots, \ell_k \in \Sigma$, and, for all $1 \leq i \leq k$, $(n_i, n_{i+1}) \in \mathbf{E}(\ell_j)$. A path $n_1 \dots n_{k+1}$ forms a *cycle* if $n_1 = n_{k+1}$ and the path contains at least one edge. A graph is *acyclic* if there are no cycles. A *tree* $\mathcal{T} = (\mathcal{V}, \Sigma, \mathbf{E})$ is an acyclic graph in which exactly one node, the *root*, has no incoming edges, and each other node has exactly one incoming edge which belongs to exactly one edge relation. In other words, we do not allow multi-labeled edges between two nodes in a tree. If (m, n) is an edge in $\mathcal{T}$, then node m is the *parent* of node n and node n is a *child* of node m. A graph is a *forest* if it is the union of a set of disjoint trees. A *chain* is a tree in which all nodes have at most one child.

Definition 2.1. Let Σ be a finite set of labels. The *navigational expressions* over Σ are defined by the grammar

$$e := \emptyset \mid \mathrm{id} \mid \mathrm{di} \mid \ell \mid \ell^\frown \mid \pi_j[e] \mid \overline{\pi}_j[e] \mid [e \circ e] \mid [e \cup e] \mid [e \cap e] \mid [e - e] \mid [e]^*,$$

in which $\ell \in \Sigma$ and $j \in \{1, 2\}$. Now, let $\mathcal{G} = (\mathcal{V}, \Sigma, \mathbf{E})$ be a graph and let e be an expression. The semantics of evaluation is defined as follows:

$$\begin{aligned}
[\![\emptyset]\!]_\mathcal{G} &= \emptyset; && \text{(empty set)}\\
[\![\mathrm{id}]\!]_\mathcal{G} &= \{(m, m) \mid m \in \mathcal{V}\}; && \text{(identity)}\\
[\![\mathrm{di}]\!]_\mathcal{G} &= \{(m, n) \mid m, n \in \mathcal{V} \wedge m \neq n\}; && \text{(diversity)}\\
[\![\ell]\!]_\mathcal{G} &= \mathbf{E}\,(\ell) && \text{(edge label)}\\
[\![\ell^\frown]\!]_\mathcal{G} &= \{(n, m) \mid (m, n) \in [\![\ell]\!]_\mathcal{G}; && \text{(edge label converse)}\\
[\![\pi_j[e]]\!]_\mathcal{G} &= \{(m, m) \mid m \in [\![e]\!]_\mathcal{G}|_j\}; && \text{(projection)}\\
[\![\overline{\pi}_j[e]]\!]_\mathcal{G} &= [\![\mathrm{id}]\!]_\mathcal{G} - [\![\pi_j[e]]\!]_\mathcal{G}; && \text{(coprojection)}\\
[\![\,[e_1 \circ e_2]\,]\!]_\mathcal{G} &= [\![e_1]\!]_\mathcal{G} \circ [\![e_2]\!]_\mathcal{G}; && \text{(composition)}\\
[\![\,[e_1 \cup e_2]\,]\!]_\mathcal{G} &= [\![e_1]\!]_\mathcal{G} \cup [\![e_2]\!]_\mathcal{G}; && \text{(union)}\\
[\![\,[e_1 \cap e_2]\,]\!]_\mathcal{G} &= [\![e_1]\!]_\mathcal{G} \cap [\![e_2]\!]_\mathcal{G}; && \text{(intersection)}\\
[\![\,[e_1 - e_2]\,]\!]_\mathcal{G} &= [\![e_1]\!]_\mathcal{G} - [\![e_2]\!]_\mathcal{G}; && \text{(difference)}\\
[\![[e]^*]\!]_\mathcal{G} &= \textstyle\bigcup_{i\geq 0} [\![e^i]\!]_\mathcal{G}, && \text{(Kleene-star)}
\end{aligned}$$

with $e^0 = \mathrm{id}$ and $e^k = e \circ e^{k-1}$, and in which the composition $R \circ S$ of binary relations R and S is defined by $R \circ S = \{(m, n) \mid \exists z\ ((m, z) \in R \wedge (z, n) \in S)\}$. If an expression always evaluates to a set of identical pairs, as is the case for id and all projections and coprojections, then it is called a *node expression*.

We denote by $\mathcal{N}_\Sigma$ the language of navigational expressions over Σ and by $\mathcal{N}$ the language $\bigcup_{\Sigma\subseteq\mathbf{\Sigma}} \mathcal{N}_\Sigma$.

We sometimes use the shorthand $\mathrm{all} = (\mathrm{id} \cup \mathrm{di})$ and $[e]^{+} = e \circ [e]^{*}$. Also, we shall omit brackets whenever no ambiguity is possible (e.g., outer brackets or brackets in a sequence of multiple compositions).

We write $\mathcal{F} \subseteq \{\mathrm{di}, {}^{\frown}, \pi, \overline{\pi}, \cap, -, {}^{*}\}$ to denote a set of operators in which π represents both π_1 and π_2 and, likewise, $\overline{\pi}$ represents both $\overline{\pi}_1$ and $\overline{\pi}_2$. Given Σ, we denote by $\mathcal{N}_\Sigma(\mathcal{F})$ the fragment of $\mathcal{N}_\Sigma$ that only allows $\emptyset$, id, $\ell \in \Sigma$, $\circ$, $\cup$, and all operators in $\mathcal{F}$. Also, we denote by $\mathcal{N}(\mathcal{F})$ the fragment of $\mathcal{N}$ defined as $\bigcup_{\Sigma \subseteq \mathbf{\Sigma}} \mathcal{N}_\Sigma(\mathcal{F})$.

Navigational expressions without diversity or converse, the ones we focus on in this paper, can only inspect a tree downwards from ancestor to descendant. To study this in more detail, we define the notions of downward expression formally:

Definition 2.2. Let Σ be a finite set of labels. A navigational expression e over Σ is *downward* if, for all trees $\mathcal{T}$ over Σ, and for all nodes m and n in $\mathcal{V}$, $(m, n) \in [\![e]\!]_\mathcal{T}$ only if m is an ancestor of n (i.e., there is a directed path from m to n).[3]

Given Σ, we say that expressions e_1 and e_2 over Σ are *path-equivalent*, denoted by $e_1 \equiv_{\mathrm{path}} e_2$, if, for every graph $\mathcal{G}$ over Σ, $[\![e_1]\!]_\mathcal{G} = [\![e_2]\!]_\mathcal{G}$, and are *Boolean-equivalent*, denoted by $e_1 \equiv_{\mathrm{bool}} e_2$, if, for every graph $\mathcal{G}$ over Σ, $[\![e_1]\!]_\mathcal{G} = \emptyset$ if and only if $[\![e_2]\!]_\mathcal{G} = \emptyset$. Let $\mathit{sem} \in \{\mathrm{path}, \mathrm{bool}\}$. We say that the class of expressions $\mathcal{L}_1$ is *sem-subsumed* by the class of expressions $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq_{\mathit{sem}} \mathcal{L}_2$, if every expression in $\mathcal{L}_1$ is *sem*-equivalent to an expression in $\mathcal{L}_2$. We say that $\mathcal{L}_1$ and $\mathcal{L}_2$ are *sem-equivalent*, denoted by $\mathcal{L}_1 \equiv_{\mathit{sem}} \mathcal{L}_2$, if $\mathcal{L}_1 \preceq_{\mathit{sem}} \mathcal{L}_2$ and $\mathcal{L}_2 \preceq_{\mathit{sem}} \mathcal{L}_1$.

We may also consider path equivalence or Boolean equivalence with respect to trees or chains by only allowing trees or chains instead of general graphs in the above definitions. Similarly, we may consider path equivalence or Boolean equivalence of classes of expressions on labeled or unlabeled structures. The first case coincides with the standard definition above, while in the second case we restrict ourselves to the expressions that are defined over a set of labels Σ with $|\Sigma| = 1$ (and, hence, also to unlabeled graphs when evaluating their equivalence). By combining these two ways of specializing path equivalence or Boolean equivalence, we may speak, e.g., about path equivalence and Boolean equivalence with respect to labeled chains or unlabeled trees.

[3] We do not require m to be a strict ancestor of n: there is always a directed path of length 0 from a node to itself. Hence, m and n can be the same node.

Several operators can be expressed in terms of the other operators [18]:

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

in which $^{-1}$ denotes converse at the expression level,[4] and with $j \in \{1, 2\}$. Let $\mathcal{F} \subseteq \{\mathrm{di}, {}^\frown, \pi, \overline{\pi}, \cap, -, {}^*\}$. By $\underline{\mathcal{F}}$ we denote the set of operators obtained from $\mathcal{F}$ by adding operators to $\mathcal{F}$ that can be expressed using the operators already in $\mathcal{F}$ using the above rules.

We conclude these preliminaries with some established results that will be used throughout this work:

Proposition 2.3 (Fletcher et al. [18]). *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be query languages.*

1. *If $\mathcal{L}_1 \preceq_{\mathrm{path}} \mathcal{L}_2$, then $\mathcal{L}_1 \preceq_{\mathrm{bool}} \mathcal{L}_2$;*
2. *If $\mathcal{L}_1 \not\preceq_{\mathrm{bool}} \mathcal{L}_2$, then $\mathcal{L}_1 \not\preceq_{\mathrm{path}} \mathcal{L}_2$.*

Besides carrying over results between Boolean and path queries, we can also carry over results between types of graphs.

Proposition 2.4. *Let sem $\in \{\mathrm{path}, \mathrm{bool}\}$, let $\mathcal{L}_1$ and $\mathcal{L}_2$ be query languages, and let $\mathcal{C}_1$ and $\mathcal{C}_2$ be classes of graphs such that $\mathcal{C}_1$ is a subclass of $\mathcal{C}_2$.*

1. *If $\mathcal{L}_1 \preceq_{sem} \mathcal{L}_2$ on $\mathcal{C}_2$, then $\mathcal{L}_1 \preceq_{sem} \mathcal{L}_2$ on $\mathcal{C}_1$;*
2. *If $\mathcal{L}_1 \not\preceq_{sem} \mathcal{L}_2$ on $\mathcal{C}_1$, then $\mathcal{L}_1 \not\preceq_{sem} \mathcal{L}_2$ on $\mathcal{C}_2$.*

$$\begin{tikzcd}
 & \textit{labeled chain} \arrow[r] & \textit{labeled tree} \arrow[r] & \textit{labeled graph} \\
\textit{unlabeled chain} \arrow[r] \arrow[ur] & \textit{unlabeled tree} \arrow[r] \arrow[ur] & \textit{unlabeled graph} \arrow[ur] &
\end{tikzcd}$$

Figure 3: The various classes of graphs on which relationships in the expressive power of navigational query languages is studied. The arrows between classes define is-a relationships.

We often use Proposition 2.3, Proposition 2.4, and the subclass relations of Figure 3 implicitly to carry over results between different semantics and/or various classes of graphs.

[4] By pushing down converse through the expression to the level of labels, we can always express this more general notion of converse in terms of the edge label converse operator $^\frown$ of Definition 2.1.

3. Basic results on relative expressive power

In this section, we present all basic results on the relative expressive power of the downward navigational query languages. We divide our results into four categories, based on the techniques used to prove them. Section 3.1 provides all the results obtained directly using downward-based arguments. Section 3.2 provides all the results obtained using branching-based arguments. Section 3.3 provides all the results obtained using homomorphisms. These include a major collapse result for Boolean queries on unlabeled trees and unlabeled chains. Finally, Section 3.4 uses the relation between the navigational query languages and first-order logic to prove some expressiveness results involving transitive closure. Combined with the results of Section 4, where we show that intersection never adds expressive power while difference only adds limited expressive power in some cases, these results demonstrate

Theorem 3.1. *Let $\mathcal{F}_1, \mathcal{F}_2 \subseteq \{\pi, \overline{\pi}, \cap, -, ^*\}$. We have $\mathcal{N}(\mathcal{F}_1) \preceq_{\text{bool}} \mathcal{N}(\mathcal{F}_2)$, respectively $\mathcal{N}(\mathcal{F}_1) \preceq_{\text{path}} \mathcal{N}(\mathcal{F}_2)$, on unlabeled chains, respectively labeled chains, unlabeled trees, or labeled trees if and only if there exists a directed path from $\underline{\mathcal{F}_1}$ to $\underline{\mathcal{F}_2}$ in the corresponding Hasse diagram of Figure 2.*

We start with our results obtained from downward-based arguments.

3.1. Results using the notion of downward querying

A straightforward induction on the length of expressions yields the following:

Proposition 3.2. *Let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, \cap, -, ^*\}$. Every expression in $\mathcal{N}(\mathcal{F})$ is downward.*

The queries di and $\ell^\frown$ (with ℓ any edge label) are not downward. We derive

Corollary 3.3. *Let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, \cap, -, ^*\}$. Already on unlabeled chains, we have $\mathcal{N}(\text{di}) \not\preceq_{\text{path}} \mathcal{N}(\mathcal{F})$ and $\mathcal{N}(^\frown) \not\preceq_{\text{path}} \mathcal{N}(\mathcal{F})$.*

For downward languages that cannot express π, one can easily verify that these languages cannot be used to select individual nodes in a chain. Hence, we have the following:

Proposition 3.4. *Let $\mathcal{F} \subseteq \{^*\}$. Already on unlabeled chains, we have $\mathcal{N}(\pi) \not\preceq_{\text{path}} \mathcal{N}(\mathcal{F})$.*

Proof. Let $\Sigma = \{\ell\}$, and let $\mathcal{C} = (\mathcal{V}, \Sigma, \mathbf{E})$ be a chain with $|\mathcal{V}| = 3$. On $\mathcal{C}$, the expression $\pi_1[\ell \circ \ell]$ over Σ yields $\{(r, r)\}$, with r the root of the chain. It is straightforward to verify that no expression in $\mathcal{N}_\Sigma(^*)$ yields a single identical node-pair on $\mathcal{C}$. Hence, no expression in $\mathcal{N}_\Sigma(^*)$ is path-equivalent to $\pi_1[\ell \circ \ell]$. □

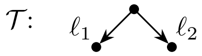

Figure 4: A tree with two labeled branches.

3.2. Detecting branches

The obvious way to detect whether or not a tree is a chain is by detecting whether some node has several children. This is particularly easy when these children are connected to their parent using distinct edge labels. Next, we show that the most basic language is not capable of detecting labeled branching.

Lemma 3.5. *Let Σ be a finite set of labels, and let e be an expression in $\mathcal{N}_\Sigma(^*)$. If there exists a tree $\mathcal{T}$ over Σ such that $[\![e]\!]_{\mathcal{T}} \neq \emptyset$, then there exists a chain $\mathcal{C}$ over Σ such that $[\![e]\!]_{\mathcal{C}} \neq \emptyset$.*

Proof. Let $\mathcal{T} = (\mathcal{V}, \Sigma, \mathbf{E})$ be a tree for which $[\![e]\!]_{\mathcal{T}} \neq \emptyset$. As $\mathcal{T}$ is given, there clearly exists a *-free expression e' over Σ for which $[\![e]\!]_{\mathcal{T}} = [\![e']\!]_{\mathcal{T}}$. We write e' as a union of $\cup$-free expressions. Since $[\![e]\!]_{\mathcal{T}} \neq \emptyset$, this union must contain a $\cup$-free expression e'' for which $[\![e'']\!]_{\mathcal{T}} \neq \emptyset$. If $e'' \equiv_{\mathrm{path}} \mathrm{id}$, then a chain $\mathcal{C}$ with a single node suffices. Else, e'' can be written as the composition of edge labels $\ell_1 \circ \ldots \circ \ell_k$. In this case, a chain $\mathcal{C}$ representing the path $n_1\, \ell_1\, n_2 \ldots n_i\, \ell_k\, n_{k+1}$ suffices. □

In contrast, most other languages are able to detect labeled branching, as shown next.

Proposition 3.6. *Let $\mathcal{F} \subseteq \{\mathrm{di}, ^\frown, \pi, \overline{\pi}, \cap, -, ^*\}$. If $\pi \in \underline{\mathcal{F}}$ or $^\frown \in \underline{\mathcal{F}}$, then, already on labeled trees, we have $\mathcal{N}(\mathcal{F}) \not\leq_{\mathrm{bool}} \mathcal{N}(^*)$.*

Proof. Let Σ be a finite set of labels, and let $\ell_1, \ell_2 \in \Sigma$. Consider the expressions $e_1 = \ell_1{}^\frown \circ \ell_2$ and $e_2 = \pi_1[\ell_1] \circ \pi_1[\ell_2]$ over Σ, which are Boolean-equivalent. Clearly, for any tree $\mathcal{T}'$ over Σ, we have $[\![e_i]\!]_{\mathcal{T}'} \neq \emptyset$, $i \in \{1, 2\}$, only if $\mathcal{T}'$ has a node with at least two children, one reachable via an edge labeled ℓ_1 and another via an edge labeled ℓ_2. Hence, for the tree $\mathcal{T}$ over Σ of Figure 4, we have $[\![e_i]\!]_{\mathcal{T}} \neq \emptyset$ and for all chains $\mathcal{C}'$ over Σ, we have $[\![e_i]\!]_{\mathcal{C}'} = \emptyset$. Let e be any expression in $\mathcal{N}_\Sigma(^*)$ with $[\![e]\!]_{\mathcal{T}} \neq \emptyset$. By Lemma 3.5, there exists a chain $\mathcal{C}$ over Σ with $[\![e]\!]_{\mathcal{C}} \neq \emptyset$. Hence, e is not Boolean-equivalent to e_i, and we conclude that $\mathcal{N}_\Sigma(^*)$ cannot express e_i. □

3.3. Results using homomorphisms

Only coprojection and difference provide a form of negation. As a consequence, all fragments of $\mathcal{N}$ that do not include these two operators are *monotone.*

Definition 3.7. Let Σ be a finite set of labels. An expression e over Σ is *monotone* if, for every graph $\mathcal{G}$ over Σ and every graph $\mathcal{G}'$ over Σ obtained by adding nodes and/or edges to $\mathcal{G}$, we have $[\![e]\!]_{\mathcal{G}} \subseteq [\![e]\!]_{\mathcal{G}'}$.

Monotonicity puts obvious limits on the expressive power of a query language: a monotone query can never put upper bounds on the size of a graph. We can, however, derive stronger results on the limitations of query languages by considering closure results under *homomorphisms*:

Definition 3.8. Let $\mathcal{L}$ be a class of expressions and F a class of functions. We say that $\mathcal{L}$ is *closed* under F if, for every pair of graphs $\mathcal{G}_1 = (\mathcal{V}_1, \Sigma, \mathbf{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \Sigma, \mathbf{E}_2)$ over the same set of labels Σ, every expression e over Σ in $\mathcal{L}$, and every function $f \in F$ mapping $\mathcal{V}_1$ into $\mathcal{V}_2$, we have that $(m, n) \in [\![e]\!]_{\mathcal{G}_1}$ implies $(f(m), f(n)) \in [\![e]\!]_{\mathcal{G}_2}$.

Such a mapping f is called a *homomorphism* from $\mathcal{G}_1$ to $\mathcal{G}_2$ if, for every pair of nodes $m, n \in \mathcal{V}_1$ and every edge label $\ell \in \Sigma$, we have that $(m, n) \in \mathbf{E}_1(\ell)$ implies $(h(m), h(n)) \in \mathbf{E}_2(\ell)$. It is called an *injective homomorphism* if, moreover, for all $m, n \in \mathcal{V}_1$, $n \neq m$ implies $h(n) \neq h(m)$.

Observe that diversity is a form of inequality. Hence, via a straightforward induction on the structure of expressions, we can show that the fragments of $\mathcal{N}$ that cannot express coprojection are closed under injective homomorphisms, whereas the languages that can express coprojections are not closed under injective homomorphisms. Additionally, we can show that the fragments of $\mathcal{N}$ that do not use diversity and cannot express coprojection are closed under arbitrary homomorphisms.

Lemma 3.9. *Let* $\mathcal{F} \subseteq \{\text{di}, ^\frown, \pi, \cap, ^*\}$.

1. $\mathcal{N}(\mathcal{F})$ *is closed under injective homomorphisms.*
2. *If* $\text{di} \notin \mathcal{F}$*, then* $\mathcal{N}(\mathcal{F})$ *is closed under homomorphisms.*

We use the above to show that the Boolean expressive power of the monotone query languages we consider is very limited.

Lemma 3.10. *Let* $\Sigma = \{\ell\}$*, and let* $\mathcal{F} \subseteq \{\text{di}, ^\frown, \pi, \cap, ^*\}$.

1. *Let e be an expression in $\mathcal{N}_\Sigma(\mathcal{F})$. If $e \not\equiv_{\text{path}} \emptyset$ on chains, then there exists $k \geq 0$ such that $e \equiv_{\text{bool}} \ell^k$ on chains.*
2. *Let e be an expression in $\mathcal{N}_\Sigma(\mathcal{F} - \{\text{di}\})$. If $e \not\equiv_{\text{path}} \emptyset$ on trees, then there exists $k \geq 0$ such that $e \equiv_{\text{bool}} \ell^k$ on trees.*

Proof. In the following, we write $\|m \to n\|_{\mathcal{T}}$, for node m an ancestor of node n in tree $\mathcal{T}$, to denote the distance between m and n. We write $\text{depth}(\mathcal{T})$ to denote the depth of tree $\mathcal{T}$, which is the maximum distance of the root node of $\mathcal{T}$ to any leaf node.

1. Let e be an expression in $\mathcal{N}_\Sigma(\mathcal{F})$, and let $\mathcal{C}$ be the chain over Σ with minimal depth for which $[\![e]\!]_{\mathcal{C}} \neq \emptyset$. Let $\mathcal{C}'$ be any chain over Σ with $\text{depth}(\mathcal{C}') \geq \text{depth}(\mathcal{C})$, and let r and r' be the root nodes of $\mathcal{C}$ and $\mathcal{C}'$, respectively. The function h that maps node z in $\mathcal{C}$ to the node z' in $\mathcal{C}'$ with $\|r \to z\|_{\mathcal{C}} = \|r' \to z'\|_{\mathcal{C}'}$ is an injective homomorphism. Hence, by Lemma 3.9, $[\![e]\!]_{\mathcal{C}} \neq \emptyset$ implies $[\![e]\!]_{\mathcal{C}'} \neq \emptyset$. Observe that $[\![\ell^{\text{depth}(\mathcal{C})}]\!]_{\mathcal{C}} \neq \emptyset$ and that $\mathcal{C}$ is the smallest chain for which this holds. Since $\text{depth}(\mathcal{C}') \geq \text{depth}(\mathcal{C})$, we also have $[\![\ell^{\text{depth}(\mathcal{C})}]\!]_{\mathcal{C}'} \neq \emptyset$.

2. Let e be an expression in $\mathcal{N}_\Sigma(\mathcal{F} - \{\text{di}\})$ and let $\mathcal{T}$ be an tree over Σ with $[\![e]\!]_\mathcal{T} \neq \emptyset$. Let $\mathcal{C}'$ be the chain over Σ with $\text{depth}(\mathcal{T}) = \text{depth}(\mathcal{C}')$ and let r and r' be the root nodes of $\mathcal{T}$ and $\mathcal{C}'$, respectively. The function h that maps every node z in $\mathcal{T}$ to the node z' in $\mathcal{C}'$ with $\|r \to z\|_\mathcal{T} = \|r' \to z'\|_{\mathcal{C}'}$ is a homomorphism. Hence, by Lemma 3.9, $[\![e]\!]_\mathcal{T} \neq \emptyset$ implies $[\![e]\!]_{\mathcal{C}'} \neq \emptyset$. Since e satisfies all conditions of Statement 1, there exists k, $0 \leq k \leq \text{depth}(\mathcal{T})$, for which $e \equiv_{\text{bool}} \ell^k$ on chains. Finally, we have $[\![\ell^k]\!]_{\mathcal{C}'} \neq \emptyset$ implies $[\![e]\!]_\mathcal{T} \neq \emptyset$ by Lemma 3.9, as one can construct a homomorphism from $\mathcal{C}'$ to the longest path in $\mathcal{T}$. □

Lemma 3.10 has significant consequences for the expressive power of Boolean queries in $\mathcal{N}(\text{di}, ^\frown, \pi, \cap, ^*)$ and $\mathcal{N}(^\frown, \pi, \cap, ^*)$.

Corollary 3.11.

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

Finally, we use Lemma 3.10 to conclude the following:

Theorem 3.12. *Already on unlabeled chains, we have* $\mathcal{N}(\overline{\pi}) \not\preceq_{\text{bool}} \mathcal{N}(\text{di}, ^\frown, \pi, \cap, ^*)$.

Proof. Let $\Sigma = \{\ell\}$. Consider the expression $e = \overline{\pi}_2[\ell] \circ \ell \circ \overline{\pi}_1[\ell]$ over Σ. Expression e evaluates to `true` on a chain over Σ if and only if this chain consists of exactly one edge. Hence, if $\mathcal{C}_1$ is the chain over Σ with $\text{depth}(\mathcal{C}_1) = 1$ and $\mathcal{C}_2$ is the chain over Σ with $\text{depth}(\mathcal{C}_2) = 2$, then $[\![e]\!]_{\mathcal{C}_1} \neq \emptyset$ and $[\![e]\!]_{\mathcal{C}_2} = \emptyset$. For every expression e' in $\mathcal{N}_\Sigma(\text{di}, ^\frown, \pi, \cap, ^*)$, however, $[\![e']\!]_{\mathcal{C}_1} \neq \emptyset$ implies $[\![e']\!]_{\mathcal{C}_2} \neq \emptyset$ by monotonicity. Hence, no expression in $\mathcal{N}_\Sigma(\text{di}, ^\frown, \pi, \cap, ^*)$ is Boolean-equivalent to e. □

3.4. Results using first-order logic

The navigational expressions without the Kleene-star ($\mathcal{N}(\text{di}, ^\frown, \pi, \overline{\pi}, \cap, -)$) are path-equivalent to Tarski's calculus of relations. It is well-known that Tarski's calculus of relations is path-equivalent to FO[3] [15, 16], the fragment of first-order logic in which formulae use at most three variables. There are several well-known bounds on the expressive power of such first-order queries [26]. In particular, the transitive closure of a binary relation cannot be expressed, even if the binary relation represents a chain. Hence,

Proposition 3.13. *Already on unlabeled chains, we have* $\mathcal{N}(^*) \not\preceq_{\text{path}} \mathcal{N}(\text{di}, ^\frown, \pi, \overline{\pi}, \cap, -)$.

With respect to Boolean queries on unlabeled trees and chains, Corollary 3.11 already showed that, in many cases, the Kleene-star does not add expressive power. Next, we exhibit the cases in which the Kleene-star does add expressive power to Boolean queries.

Proposition 3.14. *Let* $\mathcal{F} \subseteq \{\pi, \overline{\pi}, \cap, -\}$.

1. *Already on labeled chains, we have* $\mathcal{N}(^*) \not\preceq_{\text{bool}} \mathcal{N}(\mathcal{F})$.

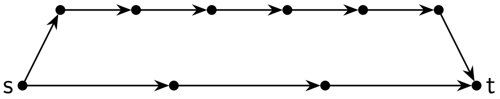

Figure 5: An acyclic directed graph, which is not a chain, a tree, or a forest. Observe that $\ell^3 \cap \ell^7$ returns the node pair (s, t).

2. *Already on unlabeled chains, we have* $\mathcal{N}(\overline{\pi}, {}^*) \not\preceq_{\text{bool}} \mathcal{N}(\mathcal{F})$.

Proof. Consider the expression $\ell_1 \circ [\ell_2 \circ \ell_2]^* \circ \ell_1$ over $\Sigma = \{\ell_1, \ell_2\}$. On a chain over Σ, this expression evaluates to `true` if and only if two there is an even-length path of ℓ_2-labeled edges between two subsequent ℓ_1-labeled edges. Next, consider the expression $\overline{\pi}_2[\ell] \circ [\ell \circ \ell]^* \circ \overline{\pi}_1[\ell]$ over $\Sigma = \{\ell\}$. On a chain over Σ, this expression evaluates to `true` if and only the chain has even length. The result now follows from the observation that the even-query is not first-order definable [26]. □

4. Redundancy of intersection and difference

In the following, we will prove that intersection and difference are redundant in downward navigational expressions. To do so, we will borrow concepts from the theory of finite automata.

Observe that queries in $\mathcal{N}({}^*)$ select pairs of nodes m, n such that there exists a directed path from m to n whose labeling satisfies some regular expression. In the case of trees, this directed path is unique, which yields a strong connection between $\mathcal{N}({}^*)$ and the closure results under intersection and difference for regular languages [25]. As a consequence, we can show, in a relatively straightforward way, that $\mathcal{N}(\cap, -, {}^*) \preceq_{\text{path}} \mathcal{N}({}^*)$.

Example 4.1. Let $\Sigma = \{\ell\}$. We can rewrite the expressions $[\ell^3]^+ \cap [\ell^7]^+$ and $[\ell^3]^+ - [\ell^7]^+$ over Σ to path-equivalent expressions that use neither intersection nor difference:

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

Notice that this rewriting does not work on arbitrary graphs. Indeed, on the graph $\mathcal{G}$ in Figure 5, we have $[\![[\ell^3]^+ \cap [\ell^7]^+]\!]_{\mathcal{G}} \neq \emptyset$, whereas $[\![[\ell^{21}]^+]\!]_{\mathcal{G}} = \emptyset$.

For regular expressions, the closure results under intersection and difference are usually obtained by first proving that regular expressions have the same expressive power as finite automata, and then proving that finite automata are closed under intersection and difference. We extend these automata-based techniques to the languages $\mathcal{N}(\mathcal{F})$ with $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$ by introducing *conditions* on automaton states. We use these extended automata to prove that, on trees, $\mathcal{N}(\mathcal{F})$ is closed under intersection and $\mathcal{N}(\underline{\mathcal{F}})$ is closed under difference.

First, in Section 4.1, we introduce condition automata. Then, in Section 4.2, we show that specific classes of condition automata describe specific fragments of the downward navigational expressions that we study. In Section 4.3, we develop techniques to make condition automata id-transition-free (which resembles removing empty-string-transitions from finite automata) and use these id-transition-free condition automata to show that, on labeled trees, condition automata are closed under intersection. In Section 4.4, we develop techniques to make condition automata deterministic (which resembles the translation from non-deterministic to deterministic finite automata) and use these deterministic condition automata to show that, on labeled trees, condition automata are closed under difference. In Section 4.5, we use these closure results to prove the cases in which either intersection or difference are redundant. Finally, in Section 4.6, we use condition automata to show the redundancy of projection in Boolean queries on chains.

4.1. Condition automata

The *conditions* we consider are node expressions of the form $\emptyset$, id, $\pi_1[e]$, $\pi_2[e]$, $\overline{\pi}_1[e]$, or $\overline{\pi}_2[e]$.

Definition 4.2. Let $\Sigma \subseteq \mathbf{\Sigma}$ be a finite set of labels, interpreted as transition labels. A *condition automaton* over Σ is a 7-tuple $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$, where S is a finite set of states, C a finite set of conditions over Σ, $I \subseteq S$ a set of initial states, $F \subseteq S$ a set of final states, $\delta \subseteq S \times (\Sigma \cup \{\text{id}\}) \times S$ the transition relation, and $\gamma \subseteq S \times C$ the state-condition relation. For a state $q \in S$, we denote $\gamma(q) = \{c \mid (q, c) \in \gamma\}$.

Let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$. We say that $\mathcal{A}$ is *$\mathcal{F}$-free* if every condition in C is an expression in $\mathcal{N}_\Sigma(\{\pi, \overline{\pi}, {}^*\} - \mathcal{F})$, we say that $\mathcal{A}$ is *acyclic* if the transition relation δ of $\mathcal{A}$ is acyclic (viewed as a labeled graph relation), and we say that $\mathcal{A}$ is id-*transition-free* if $\delta \subseteq S \times \Sigma \times S$.

Example 4.3. Consider the condition automaton $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ with

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

This automaton is visualized in Figure 6. We note that states q_3 and q_4 do not have any conditions associated with them, which is visualized via a labeling $\{\ \}$, representing an empty set of conditions. Using this visualization, it is easy to verify that the condition automaton is not acyclic (due to the ℓ_1-labeled self-loop), is $\{\overline{\pi}, {}^*\}$-free, and is id-transition-free.

Observe that condition automata are strongly related to finite automata, the main difference being that states in the automata have a set of conditions

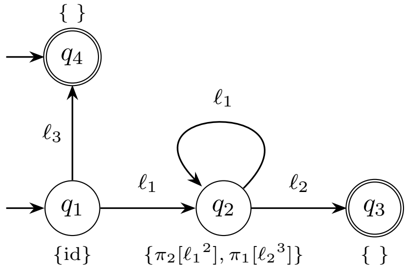

Figure 6: An example of a condition automaton.

associated to them. In the evaluation of condition automata on trees, this set of conditions determines in which tree nodes a state can hold, which we define next.

Definition 4.4. Let Σ be a finite set of labels, let $\mathcal{G} = (\mathcal{V}, \Sigma, \mathbf{E})$ be a graph over Σ, let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a condition automaton, and let $q \in S$. We define the *condition expression* $\mathsf{conditions}(q)$ of q as the node expression $\mathsf{conditions}(q) = e_1 \circ \ldots \circ e_k$ if $\gamma(q) = \{e_1, \ldots e_k\}$, $k \geq 1$, and $\mathsf{conditions}(q) = \mathrm{id}$ if $\gamma(q) = \emptyset$.[5]

We say that a node $n \in \mathcal{V}$ *satisfies* state $q \in S$ if $(n, n) \in [\![\mathsf{conditions}(q)]\!]_{\mathcal{G}}$. 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 $q_0, \ldots, q_i \in S$, $n_0, \ldots, n_i \in \mathcal{V}$, $\ell_0, \ldots, \ell_{i-1} \in \Sigma \cup \{\mathrm{id}\}$, 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 $0 \leq j < i$, $(n_j, n_{j+1}) \in [\![\ell_j]\!]_{\mathcal{G}}$.

We say that $\mathcal{A}$ *accepts* node pair $(m, n) \in \mathcal{V} \times \mathcal{V}$ 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 $[\![\mathcal{A}]\!]_{\mathcal{G}}$, as $[\![\mathcal{A}]\!]_{\mathcal{G}} = \{(m, n) \mid \mathcal{A} \text{ accepts } (m, n)\}$.

Example 4.5. Consider the condition automaton of Example 4.3, shown in Figure 6, and the tree $\mathcal{T}$ shown in Figure 7. Both are defined over $\Sigma = \{\ell_1, \ell_2, \ell_3\}$. For this combination of a condition automaton and a tree, we can construct several accepting runs. Examples are the run $(q_1, r)\, \ell_3\, (q_4, m)$, which semantically implies

$$(r, m) \in [\![\mathsf{conditions}(q_1) \circ \ell_3 \circ \mathsf{conditions}(q_4)]\!]_{\mathcal{T}} = [\![\mathrm{id} \circ \ell_3 \circ \mathrm{id}]\!]_{\mathcal{T}},$$

[5] Observe that if $\gamma(q) = \{e_1, \ldots, e_k\}$, ≥ 1, then $\mathsf{conditions}(q) \equiv_{\mathrm{path}} e_1 \cap \cdots \cap e_k$, i.e., all the conditions $\gamma(q)$ are enforced by $\mathsf{conditions}(q)$, which is what we intended. However, it is more convenient to work with compositions than intersections, hence our definition. If $\gamma(q) = \emptyset$, then no restrictive conditions hold on state q, explaining why we have set $\mathsf{conditions}(q) = \mathrm{id}$ in this case.

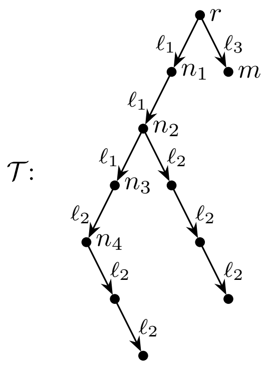

Figure 7: A labeled tree in which six distinct nodes are named.

Table 1: Fragments of the navigational expressions and corresponding classes of condition automata.

Navigational expressions	Class of condition automata
$\mathcal{N}()$	$\{\pi, \overline{\pi}, {}^*\}$-free and acyclic.
$\mathcal{N}(\pi)$	$\{\overline{\pi}, {}^*\}$-free and acyclic.
$\mathcal{N}(\pi, \overline{\pi})$	$\{{}^*\}$-free and acyclic.
$\mathcal{N}({}^*)$	$\{\pi, \overline{\pi}\}$-free.
$\mathcal{N}(\pi, {}^*)$	$\{\overline{\pi}\}$-free.
$\mathcal{N}(\pi, \overline{\pi}, {}^*)$	no restrictions.

and $(q_1, n_1)\ \ell_1\ (q_2, n_2)\ \ell_1\ (q_2, n_3)\ \ell_2\ (q_3, n_4)$, which semantically implies

$$(n_1, n_4) \in [\![\mathsf{conditions}(q_1) \circ \ell_1 \circ \mathsf{conditions}(q_2) \circ \ell_1 \circ \\ \mathsf{conditions}(q_2) \circ \ell_2 \circ \mathsf{conditions}(q_3)]\!]_{\mathcal{T}} = \\ [\![\mathrm{id} \circ \ell_1 \circ \pi_2[{\ell_1}^2] \circ \pi_1[{\ell_2}^3] \circ \ell_1 \circ \pi_2[{\ell_1}^2] \circ \pi_1[{\ell_2}^3] \circ \ell_2 \circ \mathrm{id}]\!]_{\mathcal{T}}.$$

4.2. Condition automata and downward queries

Given a finite set of labels Σ, we can evaluate a condition automaton over Σ on a graph over Σ to a set of node pairs of that graphs. Hence, we can extend the notions of path equivalence and Boolean equivalence of navigational expressions to equivalences of condition automata and navigational expressions. In a similar vein, both equivalence notions can be bootstrapped to fragments of the navigational expressions or classes of condition automata, possibly restricted to a subset of all graphs, e.g., labeled trees or unlabeled chains.

Our first goal is to show path equivalence on labeled trees of $\mathcal{N}(\mathcal{F})$, $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$, with the corresponding class of condition automata shown in Table 1.

Example 4.6. Consider the condition automaton of Example 4.3, shown in Figure 6. By carefully examining the automaton, one can conclude that, on labeled graphs, it is path-equivalent to the expression $(\ell_1 \circ \pi_2[{\ell_1}^2] \circ \pi_1[{\ell_2}^3] \circ [\ell_1 \circ \pi_2[{\ell_1}^2] \circ \pi_1[{\ell_2}^3]]^* \circ \ell_2) \cup \ell_3 \cup \mathrm{id}$ over $\Sigma = \{\ell_1, \ell_2, \ell_3\}$.

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

Expression	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	$\mathcal{A} = (\{v, w\}, \Sigma, \emptyset, \{v\}, \{w\}, \{(v, \ell, w)\}, \emptyset)$
$\pi_1[e]$	$\mathcal{A} = (\{v\}, \Sigma, \{\pi_1[e]\}, \{v\}, \{v\}, \emptyset, \{(v, \pi_1[e])\})$
$\pi_2[e]$	$\mathcal{A} = (\{v\}, \Sigma, \{\pi_2[e]\}, \{v\}, \{v\}, \emptyset, \{(v, \pi_2[e])\})$
$\overline{\pi}_1[e]$	$\mathcal{A} = (\{v\}, \Sigma, \{\overline{\pi}_1[e]\}, \{v\}, \{v\}, \emptyset, \{(v, \overline{\pi}_1[e])\})$
$\overline{\pi}_2[e]$	$\mathcal{A} = (\{v\}, \Sigma, \{\overline{\pi}_2[e]\}, \{v\}, \{v\}, \emptyset, \{(v, \overline{\pi}_2[e])\})$

To show the path equivalences proposed in Table 1, we first adapt standard closure properties for finite automata under composition, union, and Kleene-plus to the setting of condition automata:

Proposition 4.7. *Let Σ be a finite set of labels, let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$, and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be $\mathcal{F}$-free condition automata over Σ. There exists $\mathcal{F}$-free condition automata $\mathcal{A}_\circ$, $\mathcal{A}_\cup$, and $\mathcal{A}^+$ over Σ such that, for every graph $\mathcal{G}$ over Σ, $[\![\mathcal{A}_\circ]\!]_\mathcal{G} = [\![\mathcal{A}_1]\!]_\mathcal{G} \circ [\![\mathcal{A}_2]\!]_\mathcal{G}$, $[\![\mathcal{A}_\cup]\!]_\mathcal{G} = [\![\mathcal{A}_1]\!]_\mathcal{G} \cup [\![\mathcal{A}_2]\!]_\mathcal{G}$, and $[\![\mathcal{A}^+]\!]_\mathcal{G} = [[\![\mathcal{A}_1]\!]_\mathcal{G}]^+$. Moreover, $\mathcal{A}_\circ$ and $\mathcal{A}_\cup$ are acyclic whenever $\mathcal{A}_1$ and $\mathcal{A}_2$ are acyclic.*

Proof. Let $\mathcal{A}_1 = (S_1, \Sigma, C_1, I_1, F_1, \delta_1, \gamma_1)$ and $\mathcal{A}_2 = (S_2, \Sigma, C_2, I_2, F_2, \delta_2, \gamma_2)$. Without loss of generality, we may assume that $S_1 \cap S_2 = \emptyset$. We define $\mathcal{A}_\circ$, $\mathcal{A}_\cup$, and $\mathcal{A}^+$ as follows:

1. $\mathcal{A}_\circ = (S_1 \cup S_2, \Sigma, C_1 \cup C_2, I_1, F_2, \delta_1 \cup \delta_2 \cup \delta_\circ, \gamma_1 \cup \gamma_2)$, in which $\delta_\circ = \{(q_1, \mathrm{id}, q_2) \mid (q_1 \in F_1) \land (q_2 \in I_2)\}$.
2. $\mathcal{A}_\cup = (S_1 \cup S_2, \Sigma, C_1 \cup C_2, I_1 \cup I_2, F_1 \cup F_2, \delta_1 \cup \delta_2, \gamma_1 \cup \gamma_2)$.
3. $\mathcal{A}^+ = (S_1 \cup \{v, w\}, \Sigma, C_1, \{v\}, \{w\}, \delta_1 \cup \delta^+, \gamma_1)$, in which $v, w \notin S_1$ are two distinct fresh states and $\delta^+ = \{(v, \mathrm{id}, q) \mid q \in I_1\} \cup \{(q, \mathrm{id}, w) \mid q \in F_1\} \cup \{(w, \mathrm{id}, v)\}$.

It is straightforward to see that $\mathcal{A}_\circ$, $\mathcal{A}_\cup$, and $\mathcal{A}^+$ satisfy the requirements of the Proposition. □

We can translate navigational expressions to condition automata in a recursive manner. The base cases are the atomic and node expressions described in Table 2. The recursive cases are compositions, unions, and transitive closures, for which we use the closure results of Proposition 4.7 in a straightforward manner. We conclude

Proposition 4.8. *Let Σ be a finite set of labels, and let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$. On labeled graphs, every expression in $\mathcal{N}_\Sigma(\mathcal{F})$ is path-equivalent to some condition automaton over Σ in the class specified for $\mathcal{N}(\mathcal{F})$ in Table 1.*

Finally, to show the converse, we adapt the translation of finite automata to regular expressions to the setting of condition automata.

Proposition 4.9. *Let Σ be a finite set of labels, and let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, ^*\}$. On labeled graphs, every condition automaton over Σ in the class specified for $\mathcal{N}(\mathcal{F})$ in Table 1 is path-equivalent to some expression in $\mathcal{N}_\Sigma(\mathcal{F})$.*

Proof. In the usual translation from finite automata to regular expressions consists of three steps [25]. First, the finite automaton is translated to a *generalized transition graph* in which each transition is a regular expression. Then, the generalized transition graph is reduced by removing states and replacing them by more complicated transitions. Finally, the reduced generalized transition graph has a single regular expression between initial and final state, which is the result of the translation.

For our purpose, we only need to add to the above the ability to deal with conditions. Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a condition automaton. As a first step, we translate this condition automaton to a generalized transition graph in which each transition is an expression in $\mathcal{N}_\Sigma(\mathcal{F})$. We do so by translating each transition (p, ℓ, q) to the expression $\mathsf{conditions}(p) \circ \ell \circ \mathsf{conditions}(q)$. Next, we follow the standard procedure to reduce the generalized transition graph to an expression. To verify that the resulting expression is in $\mathcal{N}_\Sigma(\mathcal{F})$, we only need to observe that the standard procedure to translate a generalized transition graph to a regular expression only uses composition (concatenation), union, and Kleene-star. The Kleene-star is only introduced when the transition graph is cyclic. □

Notice that we not only proved path equivalence between classes of condition automata and classes of the expressions on general labeled graphs, but also provided constructive algorithms to translate between these classes.

4.3. Closure under intersection

In the following, we work towards showing that condition automata, when used as queries on *trees*, are not only closed under $\circ$, $\cup$, and *, as Proposition 4.7 shows, but also under $\cap$ and $-$. We then use this closure result to remove $\cap$ and $-$ from expressions that are used to query trees. The standard approach to constructing the intersection of two finite automata is by making their cross-product. In a fairly straightforward manner, we can apply a similar cross-product construction to condition automata, given that 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. To do so, we first introduce the notion of identity pairs.

Definition 4.10. Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$ be a condition automaton and $q \text{ id } q_1 \cdots \text{ id } q_j$ be a path in $\mathcal{A}$. We say that the pair $(q, \{q, q_1, \ldots, q_j\})$ is an *identity pair* of $\mathcal{A}$.

Example 4.11. In Figure 8 two condition automata are shown. Observe that the condition automaton on the *left* has id-transitions. The id-transition-free condition automaton on the *right* is obtained from it by applying the construction of Lemma 4.12, below. The main step in this process is obtaining the identity pairs,

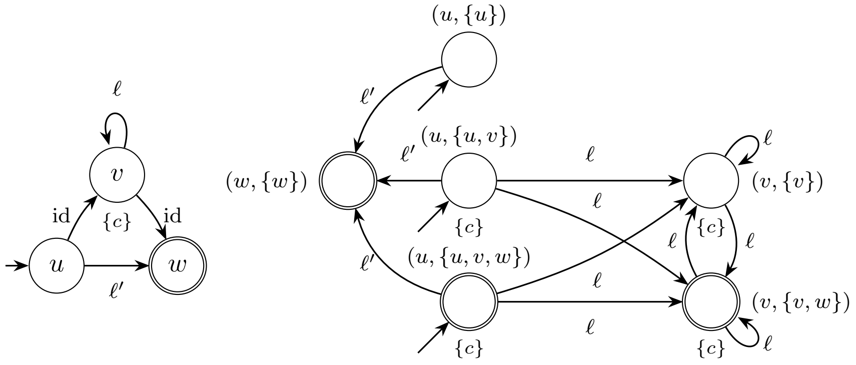

Figure 8: Two path-equivalent condition automata. Only the one on the *right* is id-transition-free.

as these are the states of the constructed condition automaton. The original condition automaton has the following paths consisting of identity-transitions only:

$$u, \quad v, \quad w, \quad u \,\mathrm{id}\, v, \quad v \,\mathrm{id}\, w, \quad u \,\mathrm{id}\, v \,\mathrm{id}\, w,$$

resulting in the identity pairs $(u,\{u\})$, $(v,\{v\})$, and $(w,\{w\})$, the identity pairs $(u,\{u,v\})$ and $(v,\{v,w\})$, and the identity pair $(u,\{u,v,w\})$, which are the states of the constructed condition automaton on the *right*.

Next, we formally prove that id-transitions can be eliminated by generalizing Example 4.11:

Lemma 4.12. *Let Σ be a finite set of labels, let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$, and let $\mathcal{A}$ be an $\mathcal{F}$-free condition automaton over Σ. On labeled graphs, there exists an* id*-transition-free and $\mathcal{F}$-free condition automaton $\mathcal{A}_{\text{noid}}$ over Σ that is path-equivalent to $\mathcal{A}$. Moreover, $\mathcal{A}_{\text{noid}}$ is acyclic whenever $\mathcal{A}$ is acyclic.*

Proof. Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$. Note that we cannot simply use finite automata techniques to remove id transitions in the same way as empty-word transitions can be removed [25], as we also have to keep track of any additional state conditions implied by id transitions. We construct $\mathcal{A}_{\text{noid}} = (S_{\text{noid}}, \Sigma, C, I_{\text{noid}}, F_{\text{noid}}, \delta_{\text{noid}}, \gamma_{\text{noid}})$ with

$$\begin{aligned}
S_{\text{noid}} &= \{(q,Q) \mid Q \subseteq S \text{ and } (q,Q) \text{ is an identity pair of } \mathcal{A}\};\\
I_{\text{noid}} &= \{(q,Q) \mid ((q,Q) \in S_{\text{noid}}) \land (q \in I)\};\\
F_{\text{noid}} &= \{(q,Q) \mid ((q,Q) \in S_{\text{noid}}) \land (Q \cap F \neq \emptyset)\};\\
\delta_{\text{noid}} &= \{((p,P),\ell,(q,Q)) \mid ((p,P) \in S_{\text{noid}}) \land (\ell \in \Sigma) \land{}\\
&\qquad ((q,Q) \in S_{\text{noid}}) \land (\exists p' \; (p' \in P) \land (p',\ell,q) \in \delta)\};\\
\gamma_{\text{noid}} &= \{((q,Q),c) \mid ((q,Q) \in S_{\text{noid}}) \land (\exists q' \; (q' \in Q) \land (c \in \gamma(q')))\}.
\end{aligned}$$

Let $\mathcal{G}$ be a graph over Σ. By a straightforward induction on the lengths of runs in $\mathcal{A}$ and $\mathcal{A}_{\text{noid}}$, one can prove that $[\![\mathcal{A}_{\text{noid}}]\!]_{\mathcal{G}} = [\![\mathcal{A}]\!]_{\mathcal{G}}$. □

We now proceed with showing that condition automata, when used to query trees as opposed to general labeled graphs, are closed under intersection. We already know from Example 4.1 that, on general graphs, standard automata-techniques cannot be adapted to obtain these closure results. On trees, however, the situation of Example 4.1 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 for querying trees. The lemma below formalizes this observation:

Lemma 4.13. *Let Σ be a finite set of labels, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be* id*-transition-free condition automata over Σ, and let $\mathcal{T}$ be a tree over Σ. If there exists a run $r_1 = (p_1, n_1)\,\ell^1_1 \cdots \ell^1_{i_1}\,(q_1, n_{i_1+1})$ of $\mathcal{A}_1$ on $\mathcal{T}$ and there exists a run $r_2 = (p_2, m_1)\,\ell^2_1 \cdots \ell^2_{i_2}\,(q_2, m_{i_2+1})$ 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 $1 \leq j \leq i$, $\ell^1_j = \ell^2_j$ and $n_j = m_j$.*

Proof. Let $s = n_1 = m_1$ and $t = n_{i_1+1} = m_{i_2+1}$. By the semantics of condition automata, the existence of run r_1 implies that $(s, t) \in [\![\ell^1_1 \circ \ldots \circ \ell^1_{i_1}]\!]_{\mathcal{T}}$ and the existence of run r_2 implies that $(s, t) \in [\![\ell^2_1 \circ \ldots \circ \ell^2_{i_2}]\!]_{\mathcal{T}}$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are id-transition-free, every $\ell^1_{j_1}$ and every $\ell^2_{j_2}$, with $1 \leq j_1 \leq i_1$ and $1 \leq j_2 \leq i_2$, is an edge label. As $\mathcal{A}_1$ and $\mathcal{A}_2$ are evaluated on a tree $\mathcal{T}$, there is only a single downward path from node s to node t. Hence, the two runs must traverse the same path, and follow the same edge labels. We conclude that $i_1 = i_2 = i$ and, for all $1 \leq j \leq i$, $\ell^1_j = \ell^2_j$. □

This allows us to prove the following:

Proposition 4.14. *Let Σ be a finite set of labels, let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$ and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be $\mathcal{F}$-free condition automata over Σ. There exists an $\mathcal{F}$-free condition automaton $\mathcal{A}_\cap$ over Σ such that, for every tree $\mathcal{T}$ over Σ, we have $[\![\mathcal{A}_\cap]\!]_{\mathcal{T}} = [\![\mathcal{A}_1]\!]_{\mathcal{T}} \cap [\![\mathcal{A}_2]\!]_{\mathcal{T}}$. The condition automaton $\mathcal{A}_\cap$ is acyclic whenever $\mathcal{A}_1$ or $\mathcal{A}_2$ is acyclic.*

Proof. Let $\mathcal{A}_1 = (S_1, \Sigma, C_1, I_1, F_1, \delta_1, \gamma_1)$ and $\mathcal{A}_2 = (S_2, \Sigma, C_2, I_2, F_2, \delta_2, \gamma_2)$. By Lemma 4.12, we assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ are id-transition-free. We construct $\mathcal{A}_\cap = (S_1 \times S_2, \Sigma, C_1 \cup C_2, I_1 \times I_2, F_1 \times F_2, \delta_\cap, \gamma_\cap)$ where

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

Let $\mathcal{T}$ be a tree over Σ and let m, n be a pair of nodes of $\mathcal{T}$. We have $(m, n) \in [\![\mathcal{A}_1]\!]_{\mathcal{T}} \cap [\![\mathcal{A}_2]\!]_{\mathcal{T}}$ if and only if there exists a run $(p_1, m)\,\ell^1_1 \cdots \ell^1_{i_1}\,(q_1, n)$ of $\mathcal{A}_1$ on $\mathcal{T}$ with $p_1 \in I_1$ and $q_1 \in F_1$ and a run $(p_2, m)\,\ell^2_1 \cdots \ell^2_{i_2}\,(q_2, n)$ of $\mathcal{A}_2$ on $\mathcal{T}$ with $p_2 \in I_2$ and $q_2 \in F_2$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are id-transition-free, by Lemma 4.13, and by the construction of $\mathcal{A}_\cap$, these runs exist if and only if there exists a run $((p_1, p_2), m)\,\ell_1 \cdots \ell_i\,((q_1, q_2), n)$ of $\mathcal{A}_\cap$ on $\mathcal{T}$ with $(p_1, p_2) \in I_1 \times I_2$ and $(q_1, q_2) \in F_1 \times F_2$. Hence, we conclude $(m, n) \in [\![\mathcal{A}_\cap]\!]_{\mathcal{T}}$. Observe that we did not add new condition expressions to the set of condition expressions in the proposed constructions. Hence, we conclude that $\mathcal{A}_\cap$ is $\mathcal{F}$-free whenever $\mathcal{A}_1$ and

$\mathcal{A}_2$ are $\mathfrak{F}$-free. We also observe that every run r of $\mathcal{A}_\cap$ on $\mathcal{T}$ can be split into runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on tree $\mathcal{T}$ of the same length as r. Hence, there can only be loops in $\mathcal{A}_\cap$ if both $\mathcal{A}_1$ and $\mathcal{A}_2$ have loops and we conclude that $\mathcal{A}_\cap$ is acyclic whenever $\mathcal{A}_1$ or $\mathcal{A}_2$ is acyclic. □

4.4. Closure under difference

Next, we show that condition automata, when used as tree queries, are also closed under difference. Usually, the difference of two finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ is obtained 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 when evaluated on a tree) is not a downward binary relation. Observe, however, that it is not necessary to consider the full complement for this purpose: since the difference of two downward binary relations is itself a downward relation, we can restrict ourselves to the *downward complement* of a binary relation.

Definition 4.15. Let $\mathcal{T} = (\mathcal{V}, \Sigma, \mathbf{E})$ be a tree. We define the *downward complement* of a binary relation $\mathrm{R} \subseteq \mathcal{V} \times \mathcal{V}$, denoted by $\overline{\mathrm{R}}_\downarrow$, as

$$\overline{\mathrm{R}}_\downarrow = \{(m, n) \mid (m, n) \notin \mathrm{R} \wedge m \text{ is an ancestor of } n \text{ in } \mathcal{T}\}.$$

If $\mathcal{A}_1$ and $\mathcal{A}_2$ are condition automata and $\mathcal{T}$ is a tree, all over the same finite set of labels, then we have $[\![\mathcal{A}_1]\!]_\mathcal{T} - [\![\mathcal{A}_2]\!]_\mathcal{T} \equiv [\![\mathcal{A}_1]\!]_\mathcal{T} \cap \overline{[\![\mathcal{A}_2]\!]_\mathcal{T}}_\downarrow$. Hence, we only need to show that condition automata are closed under downward complement. Recall that finite automata are closed under complement and the complement of an automaton can easily be constructed if the automaton is deterministic [25]. For the construction of the downward complement of a condition automaton, we introduce a notion similar to deterministic finite automata.

Definition 4.16. 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 $\mathcal{T}$ over Σ and for every pair of nodes m, n of $\mathcal{T}$ with m an ancestor of n, there exists exactly one run $(q, m)\, \ell \ldots (p, n)$ of $\mathcal{A}$ on $\mathcal{T}$ with $q \in I$.

We observe that if the condition automaton does not specify any conditions, then Definition 4.16 reduces to the classical definition of a deterministic finite automaton. Moreover, the definition of a deterministic condition automaton relies on the automaton being evaluated on trees, as more general graphs can have several identically-labeled paths between pairs of nodes.

Example 4.17. The condition automaton in Figure 6 is clearly not deterministic: runs of a single state can already start at two distinct initial states. In Figure 9 we exhibit a conditional automaton over $\Sigma = \{\ell_1, \ell_2\}$ that is deterministic. This deterministic condition automaton accepts node pairs (m, n), $m \neq n$, if m satisfies $\pi_2[{\ell_1}^3]$ and if there exists 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 $\pi_2[{\ell_1}^3]$.

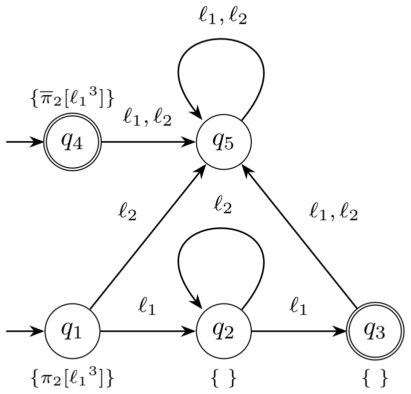

Figure 9: An example of a deterministic condition automaton.

In the construction of deterministic condition automata we use the *condition complement* of a condition e, denoted by $\text{ccompl}(e)$, and defined as follows:

$$\text{ccompl}(e) = \begin{cases} \emptyset & \text{if } e = \text{id}; \\ \text{id} & \text{if } e = \emptyset; \\ \overline{\pi}_j[e'] & \text{if } e = \pi_j[e'],\ j \in \{1,2\}; \\ \pi_j[e'] & \text{if } e = \overline{\pi}_j[e'],\ j \in \{1,2\}. \end{cases}$$

Observe that the condition complement of a projection expression is a co-projection expression, and vice-versa. If S is a set of conditions, then we use the notation $\text{ccompl}(S)$ to denote the set $\{\text{ccompl}(c) \mid c \in S\}$.

Lemma 4.18. *Let Σ be a finite set of edge labels, let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$, and let $\mathcal{A}$ be an $\mathcal{F}$-free condition automaton over Σ. There exists a deterministic condition automaton $\mathcal{A}_D$ over Σ that is path-equivalent to $\mathcal{A}$ on labeled trees. The condition automaton $\mathcal{A}_D$ is $\{{}^*\}$-free if ${}^* \in \mathcal{F}$ and $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \in \mathcal{F}$.*

Proof. Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$. By Lemma 4.12, we may assume that $\mathcal{A}$ is id-transition-free. We notice that $\mathcal{A}$ is not deterministic if either two initial states can hold on the same node or if there is a pair of transitions originating from a state q, labeled by ℓ, lead to two distinct states that can hold on the same node. Hence, to make $\mathcal{A}$ deterministic, we combine an exhaustive enumeration of all possible mutually-exclusive state conditions implied by C with the classical powerset construction to make the transition relation deterministic [25]. We construct the resulting automaton $\mathcal{A}_D = (S_D, \Sigma, C_D, I_D, F_D, \delta_D, \gamma_D)$ as follows:

$$S_D = \{(Q, V) \mid (V \subseteq C) \wedge (Q \subseteq \{q \mid q \in S \wedge \gamma(q) \subseteq V\})\};$$
$$C_D = C \cup \text{ccompl}(C);$$
$$I_D = \{(Q, V) \mid ((Q, V) \in S_D) \wedge (Q \subseteq I) \wedge$$

$$
\begin{aligned}
&\qquad\qquad\qquad\neg\exists Q' \; (((Q', V) \in S_D) \wedge (Q \subset Q' \subseteq I))\};\\
F_D &= \{(Q, V) \mid ((Q, V) \in S_D) \wedge (Q \cap F \neq \emptyset)\};\\
\delta_b &= \{((Q, V), \ell, (P, W)) \mid ((Q, V), (Q, W) \in S_D) \wedge\\
&\qquad\qquad\qquad (\forall p \; ((p \in P) \implies \exists q \; ((q \in Q) \wedge (q, \ell, p) \in \delta)))\};\\
\delta_D &= \{((Q, V), \ell, (P, W)) \mid (((Q, V), \ell, (P, W)) \in \delta_b) \wedge\\
&\qquad\qquad\qquad \neg\exists P' \; ((((Q, V), \ell, (P', W)) \in \delta_b) \wedge (P \subset P'))\};\\
\gamma_D &= \{((Q, V), c) \mid (Q, V) \in S_D \wedge (c \in V \vee c \in \mathrm{ccompl}(C - V))\}.
\end{aligned}
$$

Let $\mathcal{T}$ be a tree over Σ and let m, n be nodes of $\mathcal{T}$. For any node n' of $\mathcal{T}$, let $\zeta(n')$ denote the set $\{c \mid c \in C \wedge (n', n') \in \llbracket c \rrbracket_{\mathcal{T}}\}$. Both determinism of $\mathcal{A}_D$ and path equivalence of $\mathcal{A}_D$ and $\mathcal{A}$ are guaranteed, since this construction satisfies the following properties:

1. *We have $(n, n) \in \llbracket \mathsf{conditions}(\zeta(n) \cup \mathrm{ccompl}(C - \zeta(n))) \rrbracket_{\mathcal{T}}$ and, for all $W \subseteq C$ with $W \neq \zeta(n)$, $(n, n) \notin \llbracket \mathsf{conditions}(W \cup \mathrm{ccompl}(C - W)) \rrbracket_{\mathcal{T}}$.*
   By definition, we have $(n, n) \in \llbracket \mathsf{conditions}(\zeta(n) \cup \mathrm{ccompl}(C - \zeta(n))) \rrbracket_{\mathcal{T}}$. To show that no other choice for W is possible, we consider any $W \subseteq C$ with $\zeta(n) \neq W$. We show that $(n, n) \notin \llbracket \mathsf{conditions}(W \cup \mathrm{ccompl}(C - W)) \rrbracket_{\mathcal{T}}$. As $W \neq \zeta(n)$, there exists $c \in C$ such that $c \in W - \zeta(n)$ or $c \in \zeta(n) - W$:
   (a) $c \in \zeta(n) - W$. By $c \in \zeta(n)$ and $(n, n) \in \llbracket \mathsf{conditions}(\zeta(n) \cup \mathrm{ccompl}(C - \zeta(n))) \rrbracket_{\mathcal{T}}$, we have $(n, n) \in \llbracket c \rrbracket_{\mathcal{T}}$ and $(n, n) \notin \llbracket \mathrm{ccompl}(c) \rrbracket_{\mathcal{T}}$. As $c \notin W$, we have $\mathrm{ccompl}(c) \in W \cup \mathrm{ccompl}(C - W)$. Hence, $(n, n) \notin \llbracket \mathsf{conditions}(W \cup \mathrm{ccompl}(C - W)) \rrbracket_{\mathcal{T}}$.
   (b) $c \in W - \zeta(n)$. Since $c \notin \zeta(n)$, we have $(n, n) \notin \llbracket c \rrbracket_{\mathcal{T}}$. As $c \in W$, we have $c \in W \cup \mathrm{ccompl}(C - W)$. Hence, $(n, n) \notin \llbracket \mathsf{conditions}(W \cup \mathrm{ccompl}(C - W)) \rrbracket_{\mathcal{T}}$.
2. *There exists exactly one state $(P, V) \in I_D$ such that m satisfies (P, V).*
   By Property 1, $V = \zeta(m)$. By the construction of S_D, there exists exactly one set of states $Q \subseteq I$ such that $(Q, V) \in I_D$ and $\gamma((Q, V)) = V \cup \mathrm{ccompl}(C - V)$.
3. *Let $(P, V) \in S_D$ be a state that is satisfied by m. If there exists an ℓ-labeled edge (m, n) in $\mathcal{T}$, then there exists exactly one transition $((P, V), \ell, (Q, W)) \in \delta_D$ such that n satisfies (Q, W).*
   By Property 1, $W = \zeta(n)$. By the construction of S_D and δ_D, there is exactly one set of states $Q \subseteq S$ such that $(Q, W) \in S_D$ and $((P, V), \ell, (Q, W)) \in \delta_D$. By the choice of W, n must satisfy (Q, W).
4. *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 also exists exactly one run $((P, V), m) \ldots ((Q, W), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$.*
   This follows from a repeated application of Property 3.
5. *If $(p, n)\; \ell\; (p', n')$ is a run of $\mathcal{A}$ on $\mathcal{T}$ then, for every $(P, V) \in S_D$ with $p \in P$, there exists exactly one transition $((P, V), \ell, (P', V')) \in \delta_D$ such that n' satisfies (P', V'). For this transition, we have $p' \in P'$.*
   By Property 1, $V = \zeta(n)$ and $V' = \zeta(n')$. By the construction of δ_D, there is exactly one set of states $P' \subseteq S$ such that $((P, V), \ell, (P', V')) \in \delta_D$. Observe that we must have $\gamma(p') \subseteq V'$, hence $p' \in P'$.

6. *If there exists a run $(q_1, m) \dots (q_i, n)$ of $\mathcal{A}$ on $\mathcal{T}$ with $q_1 \in I$, then there also exists a run $((Q_1, V_1), m) \dots ((Q_i, V_i), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$ with $(Q_1, V_1) \in I_D$, and, for all j, $1 \leq j \leq i$, $q_j \in Q_j$.*
   By induction on the length of the run. The base cases are runs of length 0, which are covered by Property 2. The inductive cases are runs of length $i \geq 1$, for which Property 5 can be used in a straightforward manner to extend runs of length $i - 1$ to length i.
7. *If there exists a run $((P, V), m)\, \ell\, ((Q, W), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$, then, for all $q \in Q$, there also exists a run $(p, m)\, \ell\, (q, n)$, with $p \in P$, of $\mathcal{A}$ on $\mathcal{T}$.*
   By the construction of δ_D, there must be a $p \in P$ such that $(p, \ell, q) \in \delta$. By $p \in P$ and $q \in Q$ and by the definition of S_D and γ_D, we have $\gamma(p) \subseteq V \subseteq \gamma_D((P, V))$ and $\gamma(q) \subseteq W \subseteq \gamma_D((Q, W))$. Hence, m satisfies p and n satisfies q. Thus $(p, m)\, \ell\, (q, n)$ is a run of $\mathcal{A}$ on $\mathcal{T}$.
8. *If there exists a run $((Q_1, V_1), m) \dots ((Q_i, V_i), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$, then, for all $q_i \in Q_i$, there also exists a run $(q_1, m) \dots (q_i, n)$ of $\mathcal{A}$ on $\mathcal{T}$ with, for all j, $1 \leq j < i$, $q_j \in Q_j$.*
   By induction on the length of the run. The base cases involve runs of the form $((Q_i, V_i), n)$ of $\mathcal{A}_D$ on $\mathcal{T}$ with $Q_i \neq \emptyset$. By the definition of S_D and γ_D, we have $\gamma(q_i) \subseteq V_i \subseteq \gamma_D((Q_i, V_i))$. Hence, n satisfies q_i and we conclude that (q_i, n) is a run of $\mathcal{A}$ on $\mathcal{T}$. The inductive cases are runs of length $i \geq 1$, for which Property 7 can be used in a straightforward manner to extend runs of length $i - 1$ to length i.

By Property 2 and Property 4 we conclude that $\mathcal{A}_D$ is a deterministic condition automaton. By Property 6 and the construction of I_D and F_D, $[\![\mathcal{A}]\!]_{\mathcal{T}} \subseteq [\![\mathcal{A}_D]\!]_{\mathcal{T}}$. By Property 8 and the construction of I_D and F_D, $[\![\mathcal{A}_D]\!]_{\mathcal{T}} \subseteq [\![\mathcal{A}]\!]_{\mathcal{T}}$. Hence, we conclude that $\mathcal{A}$ and $\mathcal{A}_D$ are path-equivalent. The construction of C_D did not add any usage of *, and introduced $\overline{\pi}$ only when π was present and π only when $\overline{\pi}$ was present. Hence, the condition automata $\mathcal{A}_D$ is $\{^*\}$-free if $^* \in \mathcal{F}$ and $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \in \mathcal{F}$. □

Using Lemma 4.18, we can construct the downward complement of a condition automaton.

Proposition 4.19. *Let Σ be a finite set of edge labels, let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, ^*\}$, and let $\mathcal{A}$ be an $\mathcal{F}$-free condition automaton over Σ. There exists a condition automaton $\mathcal{A}'$ over Σ such that, for every tree $\mathcal{T}$ over Σ, we have $[\![\mathcal{A}']\!]_{\mathcal{T}} = \overline{[\![\mathcal{A}]\!]}_{\mathcal{T}_\downarrow}$. The condition automaton $\mathcal{A}'$ is $\{^*\}$-free if $^* \in \mathcal{F}$ and $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \in \mathcal{F}$.*

Proof. Let $\mathcal{A}_D = (S_D, \Sigma, C_D, I_D, F_D, \delta_D, \gamma_D)$ be a deterministic condition automaton equivalent to $\mathcal{A}$. We construct $\mathcal{A}' = (S_D, \Sigma, C_D, I_D, S_D - F_D, \delta_D, \gamma_D)$ by swapping final and non-final states. Hence, $[\![\mathcal{A}']\!]_{\mathcal{T}} = \overline{[\![\mathcal{A}]\!]}_{\mathcal{T}_\downarrow}$. Since no new condition expressions were added in the above construction, $\mathcal{A}'$ is $\{^*\}$-free if $^* \in \mathcal{F}$ and $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \in \mathcal{F}$. □

We can now conclude the following:

Corollary 4.20. *Let Σ be a finite set of edge labels, let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, {}^*\}$, and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be $\mathcal{F}$-free condition automata over Σ. There exists a condition automaton $\mathcal{A}_-$ over Σ such that, for every tree $\mathcal{T}$ over Σ, we have $[\![\mathcal{A}_-]\!]_{\mathcal{T}} = [\![\mathcal{A}_1]\!]_{\mathcal{T}} - [\![\mathcal{A}_2]\!]_{\mathcal{T}}$. The condition automaton $\mathcal{A}_-$ is $\{{}^*\}$-free if ${}^* \in \mathcal{F}$, $\{\pi, \overline{\pi}\}$-free if $\pi, \overline{\pi} \in \mathcal{F}$, and acyclic whenever $\mathcal{A}_1$ is acyclic.*

Proof. Since $[\![\mathcal{A}_1]\!]_{\mathcal{T}} - [\![\mathcal{A}_2]\!]_{\mathcal{T}} = [\![\mathcal{A}_1]\!]_{\mathcal{T}} \cap \overline{[\![\mathcal{A}_2]\!]_{\mathcal{T}}}_{\downarrow}$, we can apply Propositions 4.14 and 4.19 to construct $\mathcal{A}_-$. □

4.5. The collapse of $\cap$ and $-$ in downward queries

Proposition 4.14 and Corollary 4.20 can be used to remove intersection and difference from an expression at the highest level, but these results ignore expressions inside projections and coprojections. To fully remove intersection and difference, we use a bottom-up construction:

Theorem 4.21. *Let $\mathcal{F} \subseteq \{\pi, \overline{\pi}, \cap, -, {}^*\}$. On labeled trees, we have $\mathcal{N}(\mathcal{F}) \preceq_{\text{path}} \mathcal{N}(\underline{\mathcal{F}} - \{\cap, -\})$.*

Proof. Let Σ be a finite set of labels, and let e be an expression in $\mathcal{N}_\Sigma(\mathcal{F})$. We construct a path-equivalent expression in $\mathcal{N}_\Sigma(\underline{\mathcal{F}} - \{\cap, -\})$ in a bottom-up fashion by constructing a condition automaton $\mathcal{A}$ over Σ that is path-equivalent to e and satisfies the conditions put on automata corresponding to the class $\mathcal{N}(\underline{\mathcal{F}} - \{\cap, -\})$ (see Table 1). Using Proposition 4.9, the constructed condition automaton $\mathcal{A}$ can be translated to an expression in $\mathcal{N}_\Sigma(\underline{\mathcal{F}} - \{\cap, -\})$.

The base cases are expressions of the form $\emptyset$, id, and ℓ ($\ell \in \Sigma$), for which we directly construct condition automata using Proposition 4.8. We use Proposition 4.7 to deal with the operators $\circ$, $\cup$, and *. We deal with expressions of the form $f_j[e]$, $f \in \{\pi, \overline{\pi}\}$, $j \in \{1, 2\}$ by translating the condition automaton path-equivalent to e to an expression e', which is in $\mathcal{N}_\Sigma(\underline{\mathcal{F}} - \{\cap, -\})$, and then use Proposition 4.8 to construct the condition automaton path-equivalent to $f_j[e]$. Finally, we use Proposition 4.14 and Corollary 4.20 to deal with the operators $\cap$ and $-$. □

Observe that Theorem 4.21 does not strictly depend on the graph being a tree: indirectly, Theorem 4.21 depends on Lemma 4.13, which holds for all graphs in which every pair of nodes is connected by at most one directed path. Hence, the results of Theorem 4.21 can be generalized to, for example, forests.

4.6. Condition automata on chains

Condition automata as a tool to represent and manipulate expressions can also be used to simplify Boolean queries. To illustrate this, we use condition automata to simplify expressions in $\mathcal{N}(\mathcal{F})$, $\{\pi\} \subseteq \mathcal{F} \subseteq \{\pi, {}^*\}$, used to query chains. We do this by providing manipulation steps that reduce the total *weight* of the projections in an expression:

Definition 4.22. Let e be an expression in $\mathcal{N}(\pi, ^*)$. We define the *condition depth* of e, denoted by $\operatorname{cdepth}(e)$, as

$$\operatorname{cdepth}(e) = \begin{cases} 0 & \text{if } e \in \{\emptyset, \mathrm{id}\}; \\ 0 & \text{if } e = \ell, \text{ with } \ell \text{ an edge label;} \\ \operatorname{cdepth}(e') & \text{if } e = [e']^*; \\ \operatorname{cdepth}(e') + 1 & \text{if } e = \pi_j[e'],\ j \in \{1, 2\}; \\ \max(\operatorname{cdepth}(e_1), \operatorname{cdepth}(e_2)) & \text{if } e = e_1 \oplus e_2,\ \oplus \in \{\circ, \cup\}. \end{cases}$$

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

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

We now prove the following technical lemma:

Lemma 4.23. *Let Σ be a finite set of labels, and let $\mathcal{A}$ be a $\{\overline{\pi}\}$-free and* id*-transition-free condition automaton over Σ. If* $\operatorname{cdepth}(\mathcal{A}) > 0$*, then there exists a $\{\overline{\pi}\}$-free and* id*-transition-free condition automaton $\mathcal{A}_\pi$ over Σ such that*

1. *for every chain $\mathcal{C}$ over Σ, we have $[\![\mathcal{A}]\!]_{\mathcal{C}} = \emptyset$ if and only if $[\![\mathcal{A}_\pi]\!]_{\mathcal{C}} = \emptyset$;*
2. *either* $\operatorname{cdepth}(\mathcal{A}) > \operatorname{cdepth}(\mathcal{A}_\pi)$ *or both* $\operatorname{cdepth}(\mathcal{A}) = \operatorname{cdepth}(\mathcal{A}_\pi)$ *and* $\operatorname{cweight}(\mathcal{A}) > \operatorname{cweight}(\mathcal{A}_\pi)$.

The condition automaton $\mathcal{A}_\pi$ is acyclic and $\{^*\}$-free whenever $\mathcal{A}$ is acyclic and $\{^*\}$-free.

Proof. Let $\mathcal{A} = (S, \Sigma, C, I, F, \delta, \gamma)$. Choose a condition $c \in C$ with $\operatorname{cdepth}(\mathcal{A}) = \operatorname{cdepth}(c)$. If c is either $\emptyset$ or id, then we can eliminate it in a straightforward manner. Hence, without loss of generality, we assume that c is of the form $\pi_j[e']$, $j \in \{1, 2\}$. Let $\mathcal{A}' = (S', \Sigma, C', I', F', \delta', \gamma')$ be a $\{\overline{\pi}\}$-free and id-transition-free condition automaton equivalent to e' constructed using Proposition 4.8 and Lemma 4.12. It is straightforward to verify that $\operatorname{cdepth}(\mathcal{A}') = \operatorname{cdepth}(e') = \operatorname{cdepth}(c) - 1$. First, we assume $j = 1$. To eliminate c, we will integrate the behavior of $\mathcal{A}'$ into the behavior of $\mathcal{A}$. Let $\rho \notin S \cup S'$ be a fresh state. We construct the resulting automaton $\mathcal{A}_\pi = (S_\pi, \Sigma, C_\pi, I_\pi, F_\pi, \delta_\pi, \gamma_\pi)$ as follows:

$$S_c = \{q \mid q \in S \wedge c \in \gamma(q)\}; \tag{1}$$

$$S_{\neg c} = S - S_c; \tag{2}$$

$$S_{\neg 1} = \{(q, Q) \mid q \in S_c \wedge Q \subseteq S' \wedge Q \cap I' = \emptyset\}; \tag{3}$$

$$S_\pi = (S \times \mathcal{P}(S')) - ((S_{\neg j} \cup \{\rho\}) \times (\mathcal{P}(S') - \emptyset)); \tag{4}$$

$$C_\pi = (C - \{c\}) \cup C'; \tag{5}$$

$$I_1 = \{(q, \{q'\}) \mid q \in S_c \cap I \wedge q' \in I'\}; \tag{6}$$

$$I_\pi = \{(q, \emptyset) \mid q \in S_{\neg c} \cap I\} \cup I_j; \tag{7}$$

$$\begin{aligned} F_1 = {} & \{(q, Q) \mid q \in S_{\neg c} \cap F \wedge \emptyset \subset Q \subseteq F'\} \\ & \cup \{(q, Q) \mid q \in S_c \cap F \wedge \emptyset \subset Q \subseteq F' \cap I'\} \\ & \cup \{(\rho, Q) \mid \emptyset \subset Q \subseteq F'\}; \end{aligned} \tag{8}$$

$$F_\pi = \{(q, \emptyset) \mid q \in S_{\neg c} \cap F\} \cup F_j; \tag{9}$$

$$\begin{aligned} \delta_{\mathcal{P}(S')} = \{(P, \ell, Q) \mid {} & P \subseteq S' \wedge \ell \in \Sigma_\pi \wedge Q \subseteq S' \wedge \\ & (\forall p\ p \notin P \vee (\exists q\ q \in Q \wedge (p, \ell, q) \in \delta')) \wedge \\ & (\forall q\ q \notin Q \vee (\exists p\ p \in P \wedge (p, \ell, q) \in \delta'))\}; \end{aligned} \tag{10}$$

$$\begin{aligned} \delta_{1,\mathrm{b}} = \{((p, P \cup P'), \ell, (q, Q)) \mid {} & \\ & (p, P \cup P') \in S_\pi \wedge P' \subseteq F' \wedge (q, Q) \in S_\pi \wedge \\ & ((p, \ell, q) \in \delta \vee ((p = \rho \vee p \in F) \wedge q = \rho)) \wedge \\ & (P, \ell, Q) \in \delta_{\mathcal{P}(S')}\}; \end{aligned} \tag{11}$$

$$\begin{aligned} \delta_{1,c} = \{((p, P \cup P'), \ell, (q, Q \cup \{q'\})) \mid {} & \\ & (p, P \cup P') \in S_\pi \wedge (q, Q \cup \{q'\}) \in S_\pi \wedge \\ & (p, \ell, q) \in \delta \wedge (P, \ell, Q) \in \delta_{\mathcal{P}(S')} \wedge \\ & q \in S_c \wedge q' \in I' \wedge P' \subseteq F'\}; \end{aligned} \tag{12}$$

$$\delta_\pi = \delta_{j,\mathrm{b}} \cup \delta_{j,c}; \tag{13}$$

$$\begin{aligned} \gamma_\pi = \{((q, Q), c') \mid {} & (q, Q) \in S_\pi \wedge c' \in C_\pi \wedge \\ & ((q \neq \rho \wedge c' \in \gamma(q)) \vee (\exists q'\ q' \in Q \wedge c' \in \gamma'(q')))\}, \end{aligned} \tag{14}$$

in which $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$ is the power set of set S, and we use the values 1 and 2 and the variable j, $j \in \{1, 2\}$, to indicate that definitions depend on the type j of the condition $c = \pi_j[e']$.

We prove that $\mathcal{A}_\pi$ satisfies the necessary properties.

1. *Either* $\mathrm{cdepth}(\mathcal{A}) > \mathrm{cdepth}(\mathcal{A}_\pi)$ *or both* $\mathrm{cdepth}(\mathcal{A}) = \mathrm{cdepth}(\mathcal{A}_\pi)$ *and* $\mathrm{cweight}(\mathcal{A}) > \mathrm{cweight}(\mathcal{A}_\pi)$.

   Observe that $\mathrm{cdepth}(\mathcal{A}') < \mathrm{cdepth}(\mathcal{A})$. Hence, the property follows directly from (5), the definition of $\mathrm{cdepth}(\cdot)$, and the definition of $\mathrm{cweight}(\cdot)$.
2. *Let $\mathcal{C}$ be a chain over Σ, and let $c = \pi_1[e']$. If $(m, n) \in [\![\mathcal{A}]\!]_{\mathcal{C}}$, then there exists a node v of $\mathcal{C}$ such that $(m, v) \in [\![\mathcal{A}_\pi]\!]_{\mathcal{C}}$.*

   We show that a single run of $\mathcal{A}$ is simulated by a single run of $\mathcal{A}_\pi$ that, at the same time, also simulates the runs of $\mathcal{A}'$ starting at every state $q \in S$ with $c \in \gamma(q)$.

   Let $(q_1, n_1) \dots (q_z, n_z)$ be an id-transition-free run with $q_1 \in I$ and $q_z \in F$ proving $(n_1, n_z) \in [\![\mathcal{A}]\!]_{\mathcal{C}}$. Now consider a state q_i, $1 \leq i \leq z$, such that $c \in \gamma(q_i)$. Observe that, by (1), we have $c \in \gamma(q_i)$ if and only if $q_i \in S_c$. As n_i satisfies q_i, we must have $(n_i, n_i) \in [\![c]\!]_{\mathcal{C}}$. Hence, by the semantics of $\pi_1[\cdot]$, there must exist an id-transition-free run $(p_i, n_i) \dots (p'_i, m_i)$ of $\mathcal{A}'$ on $\mathcal{C}$ with $p_i \in I'$ and $p'_i \in F'$ proving that a node m_i exists such that $(n_i, m_i) \in [\![\mathcal{A}']\!]_{\mathcal{C}}$.

   For every state q_i with $q_i \in S_c$ we choose such a run $(p_i, n_i) \dots (p'_i, m_i)$ of $\mathcal{A}'$ on $\mathcal{C}$ with $p_i \in I'$ and $p'_i \in F'$. Let d be the maximum distance between, on the one hand, node n_1, and, on the other hand, node n_z and the nodes m_i in these runs. For every $1 \leq k \leq d$, we let n_k be the k-th node on the downward path starting at n_1, and we define

$$R_k = \{s \mid (s, n_k) \text{ is part of a run } (p_i, n_i) \dots (p'_i, m_i) \text{ with } 1 \leq i \leq z \text{ and } q_i \in S_c\}.$$

Furthermore, we define $r_k = q_k$ if $k \leq z$ and $r_k = \rho$ if $k > z$. Let $n_1 \ell_1 \dots n_d$ be the directed path from node n_1 to the node at distance d in chain $\mathcal{C}$. We prove that $((r_1, R_1), n_1)\, \ell_1 \dots ((r_d, R_d), n_d)$ is a run of $\mathcal{A}_\pi$ on $\mathcal{C}$, with $(r_1, R_1) \in I_\pi$ and $(r_d, R_d) \in F_\pi$, in the following steps:

(a) *For every k, $1 \leq k \leq d$, we have $(r_k, R_k) \in S_\pi$.* If $1 \leq k \leq z$, then $(r_k, R_k) \in S \times \mathcal{P}(S')$. If $r_k \in S_c$, we have $p_k \in I'$ and, hence, $p_k \in R_k$. By (3), we have $(r_k, R_k) \notin S_{\neg 1}$, and, hence, $(r_k, R_k) \in S_\pi$. For $k > z$, we have $(r_k, R_k) \in \{\rho\} \times \mathcal{P}(S')$. By the definition of R_k, we must have $R_k \neq \emptyset$. Hence, we conclude, $(r_k, R_k) \in S_\pi$.

(b) *We have $(r_1, R_1) \in I_\pi$.* By construction, we have $r_1 = q_1$ and $q_1 \in I$. If $r_1 \notin S_c$, then $R_1 = \emptyset$ and, by (7), $(r_1, R_1) \in I_\pi$. If $r_1 \in S_c$, then $R_1 = \{p_1\}$, with $p_1 \in I'$, and, by (6), $(r_1, R_1) \in I_\pi$.

(c) *For every k, $1 \leq k \leq d$, n_k satisfies (r_k, R_k).* By construction of R_k, we have, for every $s \in R_k$, that n_k satisfies s. If $1 \leq k \leq z$, then $r_k = q_k$ and n_k satisfies q_k. Hence, we also have $(n_k, n_k) \in [\![\mathsf{conditions}(\gamma(q_k) \setminus \{c\})]\!]_{\mathcal{C}}$. Hence, by (14), n_k satisfies (r_k, R_k).

(d) *For every k, $1 \leq k < d$, $((r_k, R_k), \ell_k, (r_{k+1}, R_{k+1})) \in \delta_\pi$.* Construct sets P and Q in the following way:

$$P = \{p \mid p \in R_k \wedge (\exists q\ q \in R_{k+1} \wedge (p, \ell_k, q) \in \delta')\};$$
$$Q = \{q \mid q \in R_{k+1} \wedge (\exists p\ p \in R_k \wedge (p, \ell_k, q) \in \delta')\}.$$

Let $P' = R_k - P$. The set of states P contains those states of R_k with a *successor state* in R_{k+1}: for every $p \in P$, there exists $q \in R_{k+1}$ such that $(p, \ell_k, q) \in \delta'$. Hence, P' contains all states from R_k for which there is no successor state in R_{k+1}. By (11) and (12), we have $((r_k, R_k), \ell_k, (r_{k+1}, R_{k+1})) \in \delta_\pi$ only if the states in P' are all final states. We now prove that this restriction on P' holds. Let $s \in P'$ be a state. By the construction of P, there does not exist a state $s' \in R_{k+1}$ such that $(s, \ell_k, s') \in \delta'$. Hence, s can only be a state used at the end of a run $(p_i, n_i) \dots (p'_i, m_i)$, $1 \leq i \leq k$, with $p'_i \in F'$ and $s = p'_i$. We conclude that $P' \subseteq F'$.

Let $Q' = R_{k+1} - Q$. The set of states Q contains those states of R_{k+1} with a *predecessor state* in R_k: for every $q \in Q$, there exists $p \in R_k$ such that $(p, \ell_k, q) \in \delta'$. Hence, Q' contains all states from R_{k+1} for which there is no predecessor state in R_k. By (11) and (12), we have $((r_k, R_k), \ell_k, (r_{k+1}, R_{k+1})) \in \delta_\pi$ only if there is at most a single state in Q', which must be an initial state. We now prove that these restrictions on Q' hold. Let $s_1, s_2 \in Q'$ be states. By the construction of Q, there does not exist a state $s' \in R_k$ such that

$(s', \ell_k, s_1) \in \delta'$ or $(s', \ell_k, s_2) \in \delta'$. Hence, both s_1 and s_2 can only be states used at the begin of the run $(p_k, n_i) \dots (p'_k, m_i)$ with $p_k \in I'$ and we have $s_1 = p_{k+1} = s_2 = p_{k+1}$. We conclude that Q' contains at most a single state, which must be an initial state.

If $r_{k+1} = \rho$, then, by construction of R_{k+1}, we have, for every $s \in R_{k+1}$ and every $1 \leq i \leq z$ with $q_i \in S_c$, $(s, n_{k+1}) \neq (p_i, n_i)$. Hence, s is not at the begin of any run $(p_i, n_i) \dots (p'_i, m_i)$. It follows that there exists a state $s' \in S'$ and a run $(p_i, n_i) \dots (p'_i, m_i)$ containing $(s', n_k)\, \ell_k\, (s, n_{k+1})$. Hence, $s' \in P$, $s \notin Q'$, and $Q' = \emptyset$. If $r_k \in S_c$, then, by construction, we have $p_k \in R_k$ and $p_k \in I'$. If $1 \leq k < z$, then $r_k = q_k$, $r_{k+1} = q_{k+1}$, and $(q_k, \ell, q_{k+1}) \in \delta$. We use (11) if $Q' = \emptyset$ and (12) if $Q' \neq \emptyset$ to conclude $((r_k, R_k), \ell_k, (r_{k+1}, R_{k+1})) \in \delta_\pi$. Else, if $k \geq z$, then $r_{k+1} = \rho$ and $Q' = \emptyset$. We use (11) to conclude $((r_k, R_k), \ell_k, (r_{k+1}, R_{k+1})) \in \delta_\pi$.

(e) *We have $(r_d, R_d) \in F_\pi$.* If $r_d \neq \rho$ then $d = z$ and $q_z \in F$. If $R_d = \emptyset$ then, by (9), we have $(r_d, R_d) \in F_\pi$. If $R_d \neq \emptyset$, then, as $n_z = n_d$ is the node with maximum node distance d to n_1 used in any of the runs $(p_i, n_i) \dots (p'_i, m_i)$ with $1 \leq i \leq z$ and $q_i \in S_c$, we must also have, for every $s \in R_d$, $s \in F'$. If $q_z \in S_c$, then, by construction, $p_z \in R_d$ with $p_z \in I'$. By (8) we conclude that, in all these cases, we have $(r_d, R_d) \in F_\pi$.

We conclude that $((r_1, R_1), n_1)\, \ell_1 \dots ((r_d, R_d), n_d)$ is a run of $\mathcal{A}_\pi$ on $\mathcal{C}$ with $(r_1, R_1) \in I_\pi$ and $(r_d, R_d) \in F_\pi$, and, hence $(n_1, n_d) \in [\![\mathcal{A}_\pi]\!]_{\mathcal{C}}$.

3. *Let $\mathcal{C}$ be a chain over Σ, and let $c = \pi_1[e']$. If $(m, n) \in [\![\mathcal{A}_\pi]\!]_{\mathcal{C}}$, then there exists a node v of $\mathcal{C}$ such that $(m, v) \in [\![\mathcal{A}]\!]_{\mathcal{C}}$.*

We show that a single run of $\mathcal{A}_\pi$ simulates a single run of $\mathcal{A}$ and, at the same time, also simulates the runs of $\mathcal{A}'$ starting at every state $q \in S$ with $c \in \gamma(q)$.

Let $((q_1, Q_1), n_1)\ell_1 \dots ((q_z, Q_z), n_z)$ be an id-transition-free run of $\mathcal{A}_\pi$ on $\mathcal{C}$ proving $(n_1, n_z) \in [\![\mathcal{C}]\!]_{\mathcal{A}_\pi}$. Hence, we have $(q_1, Q_1) \in I_\pi$ and $(q_z, Q_z) \in F_\pi$. Choose i such that $1 \leq i \leq z$, $q_i \neq \rho$, and $i = z$ or $q_{i+1} = \rho$. We prove that $(q_1, n_1)\, \ell_1 \dots (q_i, n_i)$ is a run of $\mathcal{A}$ on $\mathcal{C}$ with $q_1 \in I$ and $q_i \in F$, in the following steps:

(a) *We have $q_1 \in I$.* By (6) and (7), we have $(q_1, Q_1) \in I_\pi$ only if $q_1 \in I$.

(b) *For all k, $1 \leq k \leq i$, n_k satisfies q_k.* We have n_k satisfies (q_k, Q_k). By (14), node n_k satisfies all conditions in $\gamma(q_k) \setminus \{c\}$. Hence, if $q_k \notin S_c$, then n_k satisfies q_k. If $q_k \in S_c$, then, by (3), there exists a state $p_1 \in Q_k$ such that $p_1 \in I'$. We prove that there are states $p_1 \in Q_k, \dots, p_d \in Q_{k+d}$ such that $(p_1, n_k)\, \ell_k \dots (p_d, n_{k+d})$ is a run of $\mathcal{A}'$ on $\mathcal{C}$ with $p_1 \in I'$ and $p_d \in F'$. We do so by induction on the length of the run. The base case is (p_1, n_k) and, as $p_1 \in Q_k$, this case is already proven. Thus assume we have a run $(p_1, n_k)\, \ell_k \dots (p_e, n_{k+e})$ of $\mathcal{A}'$ on $\mathcal{C}$ with $1 \leq e < d$ and $p_e \notin F'$. We show that we can extend this run to a run of length $e + 1$. Observe that (13) depends on (10), via (11) and (12). If $q_{e+1} \neq \rho$ and $p_e \notin F$, then (11) or (12) applies, and, hence, by (10), there must be a state $p_{e+1} \in$

Q_{k+e+1} such that $(p_e, \ell_{k+e}, p_{e+1}) \in \delta'$. Else, if $q_{e+1} = \rho$ and $p_e \notin F$, then (11) applies, and, hence, by (10), there must be a state p_{e+1} such that $(p_e, \ell_{k+e}, p_{e+1}) \in \delta'$. We observe that this construction will terminate, as the original run has a finite length z. Hence, at some point we encounter a state $p_d \in F'$. We conclude $(n_k, n_{k+d}) \in [\![\mathcal{A}']\!]_{\mathcal{C}}$, and, by the semantics of $\pi_1[\cdot]$, we also conclude $(n_k, n_k) \in [\![c]\!]_{\mathcal{C}}$. Hence, n_k satisfies all conditions in $\gamma(q_k)$ and, in particular, n_k satisfies q_k.

(c) *For all $1 \leq k < i$, $(q_k, \ell_k, q_{k+1}) \in \delta$.* We have $q_k \neq \rho$ and $q_{k+1} \neq \rho$, and, by the definition of a run, $((q_k, Q_k), \ell_k, (q_{k+1}, Q_{k+1})) \in \delta_\pi$. When $q_k \neq \rho$ and $q_{k+1} \neq \rho$, then each of (11) and (12) guarantees that $(q_k, \ell_k, q_{k+1}) \in \delta$.

(d) *We have $q_i \in F$.* We distinguish three cases. If $z = i$ and $Q_i = \emptyset$, then, by (9), $(q_i, Q_i) \in F_\pi$ implies $q_i \in F$. If $z = i$ and $Q_i \neq \emptyset$, then, by (8), $(q_i, Q_i) \in F_\pi$ implies $q_i \in F$ and $Q_i \subseteq F'$. If $z \neq i$, then we must have $q_{i+1} = \rho$. Hence, by (11), $((q_i, Q_i), \ell_i, (q_{i+1}, Q_{i+1})) \in \delta_\pi$ implies $q_i \in F$.

Hence, we conclude $(n_1, n_i) \in [\![\mathcal{A}]\!]_{\mathcal{C}}$.

4. *$\mathcal{A}_\pi$ is a $\{\overline{\pi}\}$-free and* id*-transition-free condition automaton over Σ. The condition automata $\mathcal{A}_\pi$ is acyclic whenever $\mathcal{A}$ is acyclic and $\{^*\}$-free.*
By (5) and by (13) we immediately conclude that $\mathcal{A}_\pi$ is a $\{\overline{\pi}\}$-free and id-transition-free condition automaton over Σ, and that $\mathcal{A}_\pi$ is $\{^*\}$-free whenever $\mathcal{A}$ is $\{^*\}$-free. If $\mathcal{A}$ is acyclic and $\{^*\}$-free, then we can use the proofs of Property 3 to translate every run of $\mathcal{A}_\pi$ into runs of $\mathcal{A}$ and $\mathcal{A}'$. Using this translation, runs of $\mathcal{A}_\pi$ can only be unbounded in length if runs of $\mathcal{A}$ or of $\mathcal{A}'$ can be unbounded in length. Hence, $\mathcal{A}_\pi$ must be acyclic whenever $\mathcal{A}$ and $\mathcal{A}'$ are acyclic.

The case for $j = 2$ is proven similarly, in which case we change the definition of $\mathcal{A}_\pi$ as follows:

$$\begin{aligned}
S_{\neg 2} &= \{(q, Q) \mid q \in S_c \wedge Q \subseteq S' \wedge Q \cap F' = \emptyset\};\\
I_2 &= \{(q, Q) \mid q \in S_{\neg c} \cap I \wedge \emptyset \subset Q \subseteq I'\}\\
&\quad \cup \{(q, Q) \mid q \in S_c \cap I \wedge \emptyset \subset Q \subseteq I' \cap F'\}\\
&\quad \cup \{(\rho, Q) \mid \emptyset \subset Q \subseteq I'\};\\
F_2 &= \{(q, \{q'\}) \mid q \in S_c \cap F \wedge q' \in F'\};\\
\delta_{2,\mathrm{b}} &= \{((p, P), \ell, (q, Q \cup Q')) \mid\\
&\qquad\qquad (p, P) \in S_\pi \wedge Q' \subseteq I' \wedge (q, Q \cup Q') \in S_\pi \wedge\\
&\qquad\qquad ((p, \ell, q) \in \delta \vee (p = \rho \wedge (q = \rho \vee q \in I))) \wedge\\
&\qquad\qquad (P, \ell, Q) \in \delta_{\mathcal{P}(S')}\};\\
\delta_{2,c} &= \{((p, P \cup \{p'\}), \ell, (q, Q \cup Q')) \mid\\
&\qquad\qquad (p, P \cup \{p'\}) \in S_\pi \wedge (q, Q \cup Q') \in S_\pi \wedge\\
&\qquad\qquad (p, \ell, q) \in \delta \wedge (P, \ell, Q) \in \delta_{\mathcal{P}(S')} \wedge\\
&\qquad\qquad p \in S_c \wedge p' \in F' \wedge Q' \subseteq I'\}.
\end{aligned}$$

The proof that this construction is sound is analogous to the case for $j = 1$. □

Lemma 4.23 only removes a single projection operator. To fully remove projections, we repeat these removal steps until no projections are left. This leads to the following result:

Theorem 4.24. *Let $\mathcal{F} \subseteq \{\pi, {}^*\}$. On labeled chains, we have $\mathcal{N}(\mathcal{F}) \preceq_{\text{bool}} \mathcal{N}(\mathcal{F} - \{\pi\})$.*

Proof. Given a set of labels Σ, we use Proposition 4.8 to translate an expression in $\mathcal{N}_\Sigma(\mathcal{F})$ to a condition automaton $\mathcal{A}$ over Σ, then we repeatedly apply Lemma 4.23 to remove conditions, and, finally, we use Proposition 4.9 to translate the resulting $\{\pi\}$-free automaton over Σ back to an expression in $\mathcal{N}_\Sigma(\mathcal{F} - \{\pi\})$. Observe that only a finite number of condition removal steps on $\mathcal{A}$ can be made, as Lemma 4.23 guarantees that either $\text{cdepth}(\mathcal{A})$ strictly decreases or else $\text{cdepth}(\mathcal{A})$ does not change and $\text{cweight}(\mathcal{A})$ strictly decreases. □

We observed that Theorem 4.21 does not strictly depend on the graph being a tree. Similarly, Theorem 4.24 does not strictly depend on the graph being a chain: 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 3.6.

Proposition 4.25. *Let $\mathcal{F} \subseteq \{{}^*\}$. On labeled trees, we have $\mathcal{N}(\mathcal{F} \cup \{\pi_2\}) \preceq_{\text{bool}} \mathcal{N}(\mathcal{F})$, but $\mathcal{N}(\mathcal{F} \cup \{\pi_1\}) \not\preceq_{\text{bool}} \mathcal{N}(\mathcal{F})$.*

5. 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. [10]. 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. [20] studied 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 studied closure properties of these XPath fragments under intersection and complement. As such, the work by Benedikt et al. answered 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 $-$) and allow only for a very restricted form of transitive closure, required to define the `ancestor` and `descendant` axes. Hence, queries such as $[\ell \circ \ell]^*$ and $\ell_1 \circ [\ell_2 \circ \ell_2]^* \circ \ell_1$, used in Proposition 3.14, are not expressible in these XPath fragments.

When accounting for the difference between the node-labeled tree model used by Benedikt et al. [20] 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 $\mathcal{X}_{r,[]}$ and its fragments, are strictly less expressive than the navigational query language $\mathcal{N}(\pi_1, ^*)$. Furthermore, we observe that $\mathcal{X}$ is path-equivalent to $\mathcal{N}()$ and that $\mathcal{X}_{[]}$ 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$^{\approx}$ [8, 9, 21, 22] 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 to FO[3], and FO[3] is equivalent to general first-order logic [8]. 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

$$\begin{aligned}\exists n \exists c_1 \exists c_2 \exists c_3 \exists c_4\ \ell(n, c_1) \land \ell(n, c_2) \land \ell(n, c_3) \land \ell(n, c_4) \land \\ (c_1 \neq c_2) \land (c_1 \neq c_3) \land (c_1 \neq c_4) \land \\ (c_2 \neq c_3) \land (c_2 \neq c_4) \land (c_3 \neq c_4),\end{aligned}$$

which is true on all trees over $\Sigma = \{\ell\}$ that have a node with at least four distinct children. Although $\mathcal{N}(\pi, \overline{\pi}, \cap, -, ^*)$ 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 $\mathcal{N}(\pi, \overline{\pi}, \cap, -, ^*)$ 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 complement for Conditional XPath and Regular XPath$^{\approx}$ cannot readily be translated to closure properties for the navigational query languages we study.

The XPath algebra of Gyssens et al. [23], when restricted to the downward fragment, corresponds to the navigational query language $\mathcal{N}(\pi, \cap, -)$. This work studied the expressiveness of various XPath algebra fragments with respect to a given tree, whereas we study the expressive power with respect to the class of labeled and unlabeled trees and chains. The positive algebra of Wu et al. [24], when restricted to the downward fragment, corresponds to the navigational query language $\mathcal{N}(\pi, \cap)$. The expressivity results in this work are dependent on the availability of a `parent`-axis (or a converse operator), and, thus, are not directly relevant for the study of the downward-only fragments.

There has been some work on the expressive power of variations of the regular path queries and nested regular path queries [14], which are equivalent to fragments of the navigational query languages. Furthermore, on graphs the navigational query languages (both labeled and unlabeled) have already been studied in full detail [5, 17–19]. On graphs, we have separation results in almost all cases, the only exception being the Boolean equivalence of the fragments $\mathcal{N}(^*)$ and $\mathcal{N}()$ on unlabeled graphs. These known separation results were all proven on general graphs. A major contribution of our work is strengthening several of these separation results to also cover much simpler classes of graphs (trees and chains). Moreover, we have shown that the navigational query languages behave, in many cases, very differently on trees and chains than they do on graphs, resulting in the major redundancy in the expressive power of the navigational query languages that we have proven in this work.

Finally, we want to point out that several of the concepts used in this paper, or at least variations thereof or similar concepts, have been studied by many different communities, for example, in algebraic logic, modal logic, category theory, and formal languages, sometimes under different names. We discuss these variations next.

The concepts of *domain* (Do) and *range* (Rn) of a binary relation are common (e.g., Tarski and Givant [27]):

$$\begin{aligned}\mathrm{Do}(R) &= \{x \mid \exists y\ (x,y) \in R\};\\ \mathrm{Rn}(R) &= \{y \mid \exists x\ (x,y) \in R\}.\end{aligned}$$

Though $\mathrm{Do}(R)$ and $\mathrm{Rn}(R)$ are not binary relations, they stand in direct correspondece with the projections $\pi_1[R]$ and $\pi_2[R]$, respectively. In the field of category theory, the concepts of domain and range are special cases of *restrictions* and *range restrictions*, whereas the coprojection $\overline{\pi}_1[R]$ is a *kernel* [28–31].

The coprojection $\overline{\pi}_1$ also corresponds to operations considered in the field of logic, including modal logic and algebraic logic. More specifically, $\overline{\pi}_1$ corresponds precisely with the *dynamic negation* operation in Dynamic Predicate Logic [32] and Dynamic Relation Algebra [33, 34]. It is also called *test negation* in Propositional Dynamic Logic [35] and *pseudo-complement* in Abstract Algebra [36]. The concept of *domain* (i.e., π_1) and *codomain* (i.e., $\overline{\pi}_1$) also feature in the study of algebraic programming languages, and in particular in an extension of Kozen's Kleene Algebra with Tests (KAT) [37] with these domain operations [38]. In summary, $\overline{\pi}_1[R]$ can be viewed as computing the complement of the domain of R, i.e., the co- or anti-domain. Analogously, $\overline{\pi}_2[R]$ can be viewed as computing the complement of the range of R, i.e., the co- or anti-range.

In some works, the *converse* operation is called the *inverse* operation (see, e.g., Fletcher et al. [5]). In other works, the inverse operator $^{-1}$ is reserved for relations that are one-to-one functions (see, e.g., Tarski and Givant [27]). Additionally, the *composition* operation $\circ$ is called the *relative product* operator $|$ in the field of relation algebra [39].

We also mention that automata with test conditions have been considered in the context of formal languages. A concrete case is the class of *finite automata on guarded strings* considered by Kozen [40] in his study of KAT [37]. Such automata are like standard non-deterministic automata except that transitions may now, besides standard labels from an alphabet, also be labeled by a boolean combinations of some basic propositions (tests). These automata accept so called "guarded strings" which correspond to traces of a KAT program. As such an automaton on guarded strings can be viewed as a representation of the semantics of such a program.

The condition automata we consider in this paper bear resemblance to finite automata on guarded strings, but differ in that transitions are labeled by relation labels and conditions are placed on the states of the automata. More importantly, a condition automaton accepts a binary relation, rather than guarded strings (KAT program traces), and is, as such, a tool for computations on binary relations. A main result of our paper is that when restricted to trees, the intersection and set difference operations can be simulated using closure results on condition automata. Such a result is not available for finite automata on guarded strings.

6. 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 are able to present the complete Hasse diagrams of relative expressiveness, visualized in Figure 2. 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 proven using the concept of condition automata, which we use to represent and manipulate navigational expressions. We also use condition automata to show 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 node inequality via the diversity operator and 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). Particularly challenging are the interactions between $-$ and di. We conjecture, for example, that $\mathcal{N}(\mathrm{di}, \overline{\pi}, -) \not\preceq_{\mathrm{bool}} \mathcal{N}(\mathrm{di}, \overline{\pi}, \cap)$, but this conjecture is still wide open.

Another direction is the study of languages with only one of the projections (or one of the coprojections) as Proposition 4.25 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 [41]. Our results for $\mathcal{N}(\pi, \cap, {}^*)$ on unlabeled trees already hint at a collapse of FO[3] to FO[2] for Boolean queries: for $\Sigma = \{\ell\}$, the query ℓ^k can easily be expressed in FO[2] algebras with semi-joins via $\ell \ltimes (\cdots \ltimes \ell)$. A last avenue of theoretical 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. [20] and the ordered-sibling tree data model. Finally, from a practical perspective it remains open whether the redundancies we have proven in this paper—especially those for intersection and difference—can be used for optimizing tree querying.

Acknowledgement

Yuqing Wu carried out part of her work during a sabbatical visit to Hasselt University with a Senior Visiting Postdoctoral Fellowship of the Research Foundation Flanders (FWO).

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

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

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

[4] L. Libkin, W. Martens, D. Vrgoč, Querying graph databases with XPath, in: Proceedings of the 16th International Conference on Database Theory, ACM, 2013, pp. 129–140.

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

[6] J. Paredaens, J. Van den Bussche, M. Andries, M. Gemis, M. Gyssens, I. Thyssens, D. Van Gucht, V. Sarathy, L. Saxton, An overview of good, SIGMOD Record 21 (1) (1992) 25–31.

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

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

[9] 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, ACM, 2006, pp. 328–337.

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

[11] L. S. Colby, A recursive algebra for nested relations, Information Systems 15 (5) (1990) 567–582.

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

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

[14] P. Barceló, Querying graph databases, in: Proceedings of the 32nd Symposium on Principles of Database Systems, PODS '13, ACM, 2013, pp. 175–188.

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

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

[17] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, 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) (2015) 167–203.

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

[19] D. Surinx, G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, Y. Wu, Relative expressive power of navigational querying on graphs using transitive closure, Logic Journal of IGPL 23 (5) (2015) 759–788.

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

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

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

[23] G. H. L. Fletcher, M. Gyssens, J. Paredaens, D. Van Gucht, Y. Wu, Structural characterizations of the navigational expressiveness of relation algebras on a tree, Journal of Computer and System Sciences 82 (2) (2016) 229–259.

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

[25] P. Linz, An Introduction to Formal Language and Automata, Jones and Bartlett Publishers, Inc., USA, 2006.

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

[27] A. Tarski, S. Givant, A Formalization of Set Theory without Variables, Vol. 41 of AMS Colloquium Publications, American Mathematical Society, Providence, RI, USA, 1987.

[28] J. R. B. Cocket, S. Lack, Restriction categories I: categories of partial maps, Theoretical Computer Science 270 (1-2) (2002) 223–259.

[29] J. R. B. Cockett, E. Manes, Boolean and classical restriction categories, Mathematical Structures in Computer Science 19 (2) (2009) 357–416.

[30] J. R. B. Cockett, X. Guo, P. Hofstra, Range categories I: General theory, Theory and Applications of Categories 26 (2012) .

[31] M. Jackson, T. Stokes, Partial maps with domain and range: Extending Schein's representation, Communications in Algebra 37 (8) (2009) 2845–2870.

[32] J. Groenendijk, M. Stokhof, Dynamic predicate logic, Linguistics and Philosophy 14 (1) (1991) 39–100.

[33] M. Hollenberg, An equational axiomatization of dynamic negation and relational composition, Journal of Logic, Language and Information 6 (4) (1997) 381–401.

[34] M. Hollenberg, A. Visser, Dynamic negation, the one and only, Journal of Logic, Language and Information 8 (2) (1999) 137–141.

[35] J. van Benthem, Modal Logic for Open Minds, Center for the Study of Language and Information, 2010.

[36] M. Jackson, T. Stokes, Semilattice pseudo-complements on semigroups, Communications in Algebra 32 (8) (2006) 2895–2918.

[37] D. Kozen, Kleene algebra with tests, ACM Trans. Program. Lang. Syst. 19 (3) (1997) 427–443.

[38] J. Desharnais, B. Möller, G. Struth, Kleene algebra with domain, ACM Trans. Comput. Log. 7 (4) (2006) 798–833.

[39] R. Maddux, Relation Algebra, Vol. 150 of Studies in Logic and the Foundations of Mathematics, Elsevier Science, 2006.

[40] D. Kozen, Automata on guarded strings and applications, Computer Science Technical Reports TR2001-1833, Cornell University (2001).

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