RAIRO-Theor. Inf. Appl. 47 (2013) 3–23
DOI: 10.1051/ita/2012027

Available online at:
www.rairo-ita.org

ON CORE XPATH WITH INFLATIONARY FIXED POINTS *

Loredana Afanasiev[1] and Balder ten Cate[2]

Abstract. We prove the undecidability of Core XPath 1.0 (CXP) [G. Gottlob and C. Koch, in *Proc. of 17th Ann. IEEE Symp. on Logic in Computer Science, LICS '02 (Copenhagen, July 2002).* IEEE CS Press (2002) 189–202.] extended with an *Inflationary Fixed Point (IFP)* operator. More specifically, we prove that the satisfiability problem of this language is undecidable. In fact, the fragment of CXP+IFP containing only the self and descendant axes is already undecidable.

Mathematics Subject Classification. 68P15, 03B45, 03B70.

1. Introduction

The Extensible Markup Language (XML), first published as a recommendation by the W3 Consortium in 1998, has become a standard for the exchange, presentation, and storage of data across the World Wide Web. The XML format has proven to be versatile enough to describe virtually any kind of information, ranging from structured to unstructured, and from short Web Service messages to gigabyte-sized data collections (*e.g.*, [9]), and to serve a rich spectrum of applications. The fundamental data structure underlying XML data model is that of a

Keywords and phrases. Modal Logic, fixed points, XML Databases, XPath.

* *This work was supported by the Netherlands Organization for Scientific Research (NWO), grant 017.001.190. It was also partially funded by the ERC grant Webdam, agreement 226513. We would like to thank Anuj Dawar and Stephan Kreutzer for useful discussions.*

[1] Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA.
loredana.afanasiev@gmail.com

[2] Department of Computer Science, University of California at Santa Cruz, 1156 High Street, Santa Cruz, CA 95064, USA. btencate@ucsc.edu

finite ordered tree, where the nodes of the tree are the elements, attributes, and pieces of text in the document.

Because of the large amount of data available in XML format, different XML querying and processing languages have been developed, such as the XML Path Language (XPath), version 1.0 and 2.0 [19, 20] and XML Query Language (XQuery), version 1.0 [21]. XQuery is the main query language for XML, featuring, for example, constructs for joins and sorting, and XPath is a strict subset of XQuery, whose main functionality is to descibe ways to navigate though XML trees.

Since the data model of XML, being based on trees, is inherently recursive, it is important for the associated query languages to be able to express recursive queries. XPath has a very restricted form of recursion by means of the recursive axes (*e.g.*, `ancestor` and `descendant`). The main construct for expressing recursive queries in XQuery [21], however, is *recursive user-defined functions (RUDFs).* They are also the reason for the fact that XQuery's is Turing-complete. User-defined functions in XQuery make it possible to express *any* types of recursion, and they make the language inherently procedural, largely evading automatic optimization approaches beyond simple strategies such as tail-recursion elimination and unfolding. This places the burden of optimization on the user's shoulders. Another difficulty with the RUDFs is that they do not seem to fit naturally into the algebraic frameworks commonly adopted by the database community for query optimization. Most XQuery engines have an underlying algebra that facilitates optimizations (*e.g.*, Natix Physical Algebra (NPA) [8], or TAX, a tree algebra for XML used by the Timber engine [13]), but since there is no proper algebraic correspondent for user-defined recursive functions, these algebras cannot be used for recursive queries.

For these reasons, in [1, 2], an extension of the XML query language XQuery with an inflationary fixed point operator was developed and implemented. The inflationary fixed point operator provides a declarative means to specify recursive queries, and is more amenable to query optimization since it blends in naturally with algebra-based query optimization frameworks such as the one of MonetDB/XQuery [4]. Indeed, it was shown in [2] that a significant performance gain can be achieved in this way.

While the empirical evidence is there, a foundational question remains: *how feasible is it to do static analysis for recursive queries specified by means of the inflationary fixed point operator? Specifically, are there substantial fragments of XQuery with the inflationary fixed point operator for which static analysis tasks such as satisfiability are decidable?*

In this paper, we give a strong negative answer. Our main result states that, already for the downward-looking fragment of Core XPath 1.0 with the inflationary fixed point operator (CXP+IFP), satisfiability is undecidable. Core XPath 1.0 (CXP) [10] is the core navigational fragment of XPath 1.0 and thus of XQuery. The proof is based on a reduction from the undecidable halting problem for 2-register machines (*cf.* [5]), and borrows ideas from the work of Dawar *et al.* [7] on the

Modal Iteration Calculus (MIC), an extension of modal logic with inflationary fixed points.

A second question we address in this paper is the relationship between CXP+IFP and MIC. While similar in spirit, it turns out that the two formalisms differ in subtle and important ways. Nevertheless, we obtain a translation from 1MIC (the fragment of MIC that does not involve simultaneous induction) to CXP+IFP node expressions.

A further implication of this translation is the fact that CXP extended with IFP is strictly more expressive than CXP extended with the *transitive closure* (TC) operator, also known as Regular XPath [15]. The result follows from the ability of 1MIC to define non-regular string languages [7].

1.1. Background and related work on fixed point logics

Logics with fixed point operators have been studied extensively in finite model theory, where it has been shown that complexity classes such as NLOGSPACE, and PTIME, and PSPACE can be characterized by means of extensions of first-order logic with fixed point operators (*cf.* [12]). In particular, on finite ordered structures, first-order logic extended with the transitive closure operator (FO(TC)) captures the complexity class NLOGSPACE (meaning that FO(TC) can express exactly those queries that can be evaluated in NLOGSPACE), and in the same way, first-order logic extended with the least fixed point operator (FO(LFP)) captures the complexity class PTIME, and first-order logic extended with the inflationary fixed point operator (FO(IFP)) captures PTIME. Incidentally, these results rely on the fact that the fixed point operators in question can bind second order variables of arbitrarily large arity. They do not depend on the question whether simultaneous fixed point definitions are allowed, as it turns out that allowing simultaneous fixed point definitions does not increase the expressive power of the languages in question.

FO(TC), FO(LFP), and FO(IFP) are undecidable for satisfiability, for the simple reason that satisfiability is already undecidable for first-order logic itself. This has lead to the question whether certain decidable fragments of first-order logic, such as modal logic or the two-variable fragment, can be extended with fixed point operators while preserving decidability. In the case of the two-variable fragment, the answer is negative: even the two-variable fragment extended with monadic transitive closure is already undecidable for satisfiability [11]. Modal logic, on the other hand, turned out to be more robust. In particular, extensions of modal logic with monadic transitive closure (in the form of Propositional Dynamic Logic and Regular XPath) and with least fixed point (in the form of the Modal μ-Calculus) are decidable and have been studied in depth [6]. Note that, in the setting of modal logic, it is most natural to consider monadic versions of the fixed point operators.

It is in this context that the extension of modal logic with the monadic inflationary fixed point operator was introduced and studied in [7], under the name *Modal Iteration Calculus* (MIC). Unfortunately, the modal iteration calculus turns out to be less well behaved than Propositional Dynamic Logic and the Modal μ-Calculus. In particular, it has an undecidable satisfiability problem and the expressive power of the logic depends on whether simultaneous fixed point definitions are allowed (the undecidability was shown to hold already for the fragment 1MIC of MIC in which simultaneous fixed point definitions are not allowed). The results we present in this paper imply something stronger, namely that satisfiability of 1MIC formulas on finite trees is already undecidable.

2. Preliminaries

2.1. Core XPath 1.0 extended with IFP (CXP+IFP)

Core XPath 1.0 (CXP) was introduced in [10] to capture the navigational core of XPath 1.0. We will follow here the definition of CXP given in [16], which differs from the one of [10], as it allows unions to be applied to arbitrary expressions (we do not know whether our proofs could be adapted for the more restrictive syntax used in [10]). On the other hand, we will consider only the downward axes *child* and *descendant* (plus the *self* axis), both in order to facilitate the comparison with MIC, and because this will already suffice for our undecidability result. We will briefly comment on the other axes later.

We consider the extension of CXP, which we call CXP+IFP, with an inflationary fixed-point operator. This inflationary fixed-point operator was first proposed in [1, 2] in the context of XQuery, and is naturally adapted here to the setting of CXP. We first give the syntax and semantics of CXP+IFP, then discuss the intuition behind the operator.

Definition 2.1 (Syntax and Semantics of CXP+IFP). *Let Σ be a set of labels and VAR a set of variables. The CXP+IFP expressions are defined as follows:*

$$\begin{aligned} axis &::= \mathsf{self} \mid \mathsf{child} \mid \mathsf{desc} \\ step &::= axis{::}l \mid axis{::}* \\ \alpha &::= step \mid \alpha_1/\alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \ \mid \alpha[\varphi] \mid X \mid \mathsf{with}\ X\ \mathsf{in}\ \alpha_1\ \mathsf{recurse}\ \alpha_2 \\ \varphi &::= \mathsf{false} \mid \langle\alpha\rangle \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid X\ , \end{aligned}$$

where $l \in \Sigma$ and $X \in VAR$. The α expressions are called path expressions, *the φ expressions are called* node expressions. *The* with ... in ... recurse ... *operator is called the* WITH *operator, while X, α_1, and α_2 in the expression* with X in α_1 recurse α_2 *are called the* variable, *the* seed, *and the* body *of the recursion.*

The CXP+IFP expressions are evaluated on finite node-labeled trees. *Let $T = (N, R, L)$ be a finite node-labeled tree, where N is a finite set of nodes, $R \subset N \times N$ is the child relation in the tree, and L is a function from N to a set of labels. Let $g(\cdot)$ be an assignment function from variables to sets of nodes, $g : VAR \rightarrow \wp(N)$.*

Then the semantics of CXP+IFP expressions are as follows:

$$\begin{aligned}
[\![\mathsf{self}]\!]_{T,g} &= \{(u,u) \mid u \in N\} \\
[\![\mathsf{child}]\!]_{T,g} &= R \\
[\![\mathsf{desc}]\!]_{T,g} &= R^+ \text{ (the transitive closure of } R\text{)} \\
[\![axis{::}l]\!]_{T,g} &= \{(u,v) \in [\![axis]\!]_{T,g} \mid L(v) = l\} \\
[\![axis{::}*]\!]_{T,g} &= [\![axis]\!]_{T,g} \\
\\
[\![\alpha_1/\alpha_2]\!]_{T,g} &= \{(u,v) \mid \exists w.(u,w) \in [\![\alpha_1]\!]_{T,g} \wedge (w,v) \in [\![\alpha_2]\!]_{T,g}\} \\
[\![\alpha_1 \cup \alpha_2]\!]_{T,g} &= [\![\alpha_1]\!]_{T,g} \cup [\![\alpha_2]\!]_{T,g} \\
[\![\alpha[\varphi]]\!]_{T,g} &= \{(u,v) \in [\![\alpha]\!]_{T,g} \mid v \in [\![\varphi]\!]_{T,g}\} \\
[\![X]\!]_{T,g} &= N \times g(X), X \in VAR \\
[\![\mathsf{with}\ X\ \mathsf{in}\ \alpha_1 \\
\mathsf{recurse}\ \alpha_2]\!]_{T,g} &= \text{union of all sets } \{w\} \times g_k(X)\text{, for all } w \in N, \\
&\quad \text{where } g_k \text{ is obtained in the following manner, for } i \geq 1\text{:} \\
&\quad g_1 := g[X \mapsto \{v \in N \mid (w,v) \in [\![\alpha_1]\!]_{T,g}\}], \\
&\quad g_{i+1} := g_i[X \mapsto g_i(X) \cup \{v \in N \mid (w,v) \in [\![\alpha_2]\!]_{T,g_i}\}], \\
&\quad \text{and } k \text{ is the least natural number for which } g_{k+1}{=}g_k. \\
\\
\{\!|\mathsf{false}|\!\}_{T,g} &= \emptyset \\
\{\!|\langle\alpha\rangle|\!\}_{T,g} &= \{u \in N \mid (u,v) \in [\![\alpha]\!]_{T,g} \text{ for some } v \in N\} \\
\{\!|\neg\varphi|\!\}_{T,g} &= N \setminus \{\!|\varphi|\!\}_{T,g} \\
\{\!|\varphi_1 \wedge \varphi_2|\!\}_{T,g} &= \{\!|\varphi_1|\!\}_{T,g} \cap \{\!|\varphi_2|\!\}_{T,g} \\
\{\!|\varphi_1 \vee \varphi_2|\!\}_{T,g} &= \{\!|\varphi_1|\!\}_{T,g} \cup \{\!|\varphi_2|\!\}_{T,g} \\
\{\!|X|\!\}_{T,g} &= g(X), X \in VAR
\end{aligned}$$

We will sometimes write $T, g, u \Vdash \varphi$ if $u \in \{\!|\varphi|\!\}_{T,g}$.

While the semantics $[\![\alpha]\!]_{T,g}$ of a path expression α is defined as a binary relation, it is natural to think of it as a function mapping each node u to a set of nodes $\{v \mid (u,v) \in [\![\alpha]\!]_{T,g}\}$, which we denote by $Result^g_u(\alpha)$. It represents the result of evaluating α in the context node u (using the assignment g). The semantics of the variables and of the WITH operator is most naturally understood from this perspective, and can be equivalently stated as follows:

- $Result^g_u(X) = g(X)$, *i.e.*, when X is used as a path expression, it evaluates to $g(X)$ regardless of the context node.
- $Result^g_u(\mathsf{with}\ X\ \mathsf{in}\ \alpha_1\ \mathsf{recurse}\ \alpha_2) = X_k$, where $X_1 = Result^{g[X\mapsto\emptyset]}_u(\alpha_1)$, $X_{i+1} = X_i \cup Result^{g[X\mapsto X_i]}_u(\alpha_2)$ for $i \geq 1$, and k is the smallest number such that $X_k = X_{k+1}$.

Note that, at each iteration, the context node of the evaluation of α_1 or α_2 remains u.

When a variable X is used as a node expression, it simply tests whether the current node belongs to the set assigned to X.

The example query below yields the set of nodes that can be reached from the context node by following the transitive closure of the child$::a$ relation.

$$\mathsf{with}\ X\ \mathsf{in}\ \mathsf{child}{::}a\ \mathsf{recurse}\ X/\mathsf{child}{::}a$$

The query below yields the set of nodes that are labeled with a and are at an even distance from the context node.

$$(\textsf{with } X \textsf{ in self::}* \textsf{ recurse } X/\textsf{child::}*/\textsf{child::}*)/\textsf{self::}a$$

It is important to note that (unlike MIC) the language provides no way to test whether a given node belongs to the result of $\textsf{with } X \textsf{ in } \alpha_1 \textsf{ recurse } \alpha_2$, it only allows us to *go to* a node belonging to the result set. From the point of view of XQuery and XPath, it is very natural to define the inflationary fixed point operator in this way, *i.e.*, as an operator on path expressions. However, this has some subtle consequences, as we explain next.

The semantics of the $\textsf{WITH}$ operator we give here differs slighly from the original semantics used in [1, 2]. According to the original semantics, when $Result^g_u(\textsf{with } X \textsf{ in } \alpha_1 \textsf{ recurse } \alpha_2)$ is computed, the result of α_1 is only used as a seed of the recursion but is not itself added to the fixed point set. In other words, $Result^g_u(\textsf{with } X \textsf{ in } \alpha_1 \textsf{ recurse } \alpha_2)$ was defined there as X_k, where $X_0 = Result^{g[X\mapsto\emptyset]}_u(\alpha_1)$, $X_1 = Result^{g[X\mapsto X_0]}_u(\alpha_2)$, $X_{i+1} = X_i \cup Result^{g[X\mapsto X_i]}_u(\alpha_2)$ for $i \geq 1$, and k is the least number such that $X_k = X_{k+1}$. The semantics we use here is arguably mathematically cleaner and more intuitive since it is truly inflationary: all the nodes assigned to the recursion variable during fixed-point computation end up in the result.

2.2. Propositional Modal Logic extended with IFP (ML+IFP)

The language ML+IFP we consider is an extension of Propositional Modal Logic (ML) [3] with a monadic IFP operator. It is also known as 1MIC, the fragment of Modal Iteration Calculus (MIC) that does not involve simultaneous induction, and it was first introduced in [7], where it was also shown that its satisfiability problem is undecidable on arbitrary structures.

Definition 2.2 (ML+IFP). *Let Σ be a set of labels and VAR a set of variables. Then the syntax of ML+IFP is defined as follows:*

$$\varphi ::= \bot \mid l \mid X \mid \Diamond\varphi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \big(\textsf{ifp } X \leftarrow \varphi\big)$$

where $l \in \Sigma$, $X \in VAR$.

The semantics of ML+IFP is given in terms of Kripke models. To facilitate the comparison with CXP+IFP, we will assume that the Kripke models assign a unique label to each node, rather than a set of labels. This is not essential, since for a finite set of labels Σ this property can be expressed with a Modal Logic formula. Let $T = (N, R, L)$ be a Kripke model, where N is a set of nodes, $R \subseteq N \times N$ is a binary relation on the nodes in N, and L is a valuation function that assigns a label from Σ to each node in N. Let $g(\cdot)$ be an assignment function from variables to sets of nodes, $g : VAR \to \wp(N)$. Then the semantics of ML+IFP formulas are *as follows:*

$$\begin{aligned}
[\![\bot]\!]_{T,g} &= \emptyset\\
[\![l]\!]_{T,g} &= \{n \in N \mid L(n) = l\}\\
[\![X]\!]_{T,g} &= g(X)\\
[\![\Diamond\varphi]\!]_{T,g} &= \{u \mid \exists v.(u,v) \in R \wedge v \in [\![\varphi]\!]_{T,g}\}\\
[\![\neg\varphi]\!]_{T,g} &= N \setminus [\![\varphi]\!]_{T,g}\\
[\![\varphi_1 \wedge \varphi_2]\!]_{T,g} &= [\![\varphi_1]\!]_{T,g} \cap [\![\varphi_2]\!]_{T,g}\\
[\![\varphi_1 \vee \varphi_2]\!]_{T,g} &= [\![\varphi_1]\!]_{T,g} \cup [\![\varphi_2]\!]_{T,g}\\
[\![\mathsf{ifp}\ X \leftarrow \varphi]\!]_{T,g} &= g_k(X), \textit{ where } g_k \textit{ is obtained in the following manner:}\\
&\quad g_0 := g[X \mapsto \emptyset],\\
&\quad g_{i+1} := g_i[X \mapsto g_i(X) \cup [\![\varphi]\!]_{T,g_i}], \textit{ for } i \geq 0,\\
&\quad \textit{where } k \textit{ is the minimum number for which } g_{k+1}{=}g_k.
\end{aligned}$$

We write $T, g, u \Vdash \varphi$ if $u \in [\![\varphi]\!]_{T,g}$. If a formula has no free variables, we may leave out the assignment and write $T, u \Vdash \varphi$ or $u \in [\![\varphi]\!]_T$.

It was shown in [7] that the satisfiability problem for ML+IFP on arbitrary Kripke models is highly undecidable (*i.e.*, neither recursively enumerable nor co-recursively enumerable. As we will show below, it is undecidable on finite trees as well.

3. Relation between ML+IFP and CXP+IFP

In this section, we give a truth-preserving translation from ML+IFP to CXP+IFP (*i.e.*, a translation that preserves truth or falsity at any given node of a structure). In fact, the translation yields CXP+IFP expressions that use only the *self* and *descendant* axes. It follows that this fragment of CXP+IFP already has (at least) the expressive power of ML+IFP.

One of the main differences between ML+IFP and CXP+IFP is that, in the former, fixed-point expressions are node expressions that test whether the current node belongs to the fixed point of a formula, while in the latter, fixed-point expressions are path expressions that travel to nodes belonging to the fixed point of a formula. Another difference is that, in CXP+IFP, during the entire fixed point computation, the expressions are evaluated from a fixed context node, whereas in ML+IFP, whether a node is added to the set at some stage of the fixed point computation is determined by local properties of the subtree below that node.

In our translation from ML+IFP to CXP+IFP we have to overcome these differences. The main idea of the translation of ML+IFP formulas of the form $\mathsf{ifp}\ X \leftarrow \varphi$ will be that, during the fixed point computation, we treat leaf nodes in a special way, never adding them to the fixed point set but keeping track of them separately. Specifically, we first compute the set Y of all leaf nodes satisfying $\mathsf{ifp}\ X \leftarrow \varphi$. Next, we let $X_0 = \emptyset$ and X_{i+1} is computed as $X_i \cup ([\![\varphi]\!]_{T,g[X \mapsto X_i \cup Y]} - Y)$. Observe how the nodes in Y are added to the input and substracted again from the output. Let X_k be the fixed point of the sequence $X_0 \subseteq X_1 \subseteq \cdots$. Then we have that $[\![\mathsf{ifp}\ X \leftarrow \varphi]\!]_{T,g} = X_k \cup Y$. The advantage of this construction is that, since the leaves are never added during the fixed point computation, they can be freely used for signalling that the context node was added to the set X: if the context node is added at some stage, we add a leaf node as well, and the presence of a leaf node in the result set will be used as a sign that we test for afterwards.

Before we give the details of the construction, we first note that when computing the inflationary fixed point of an ML+IFP formula, any leaf node that is added to the fixed point set is in fact already added at the first stage of the fixed point computation. This is expressed by the following lemma.

Lemma 3.1. *Let u be any node in a Kripke model T, and let $\varphi(X)$ be any ML+IFP formula and g an assignment. If u has no successors, then $u \in [\![\textsf{ifp}\ X \leftarrow \varphi]\!]_{T,g}$ iff $u \in [\![\varphi]\!]_{T,g[X\mapsto\emptyset]}$.*

Proof. Follows from the fact that the modal formula φ only speaks about the submodel generated by u, *i.e.*, the submodel consisting only of the node u itself. □

In what follows, we will use $\oslash$ as shorthand for $\textsf{self}::*[\textsf{false}]$, $\textsf{desc-or-self}::*$ as shorthand for $\textsf{desc}::*\cup\textsf{self}::*$, and $\textsf{leaf}$ as shorthand for $\neg\langle\textsf{child}::*\rangle$. Also, for node expressions φ,ψ and a variable X, such that X only occurs in φ in the form of node tests, we will denote by $\varphi^{X/\psi}$ the node expression obtained from φ by replacing all free occurrences of X by the node expression ψ.

The translation $\tau(\cdot)$ from ML+IFP formulas to CXP+IFP node expressions is given by equation (3.1).

$$\begin{aligned}
\tau(\bot) &= \textsf{false}\\
\tau(l) &= \langle\textsf{self}::l\rangle\\
\tau(\varphi_1\wedge\varphi_2) &= \tau(\varphi_1)\wedge\tau(\varphi_2)\\
\tau(\varphi_1\vee\varphi_2) &= \tau(\varphi_1)\vee\tau(\varphi_2)\\
\tau(\neg\varphi) &= \neg\tau(\varphi)\\
\tau(X) &= X\\
\tau(\Diamond\varphi) &= \langle\textsf{child}::*[\tau(\varphi)]\rangle\\
\tau\big(\textsf{ifp}\ X\leftarrow\varphi\big) &= \langle\big(\textsf{with}\ X\ \textsf{in desc-or-self}::*[\tau(\varphi)^{X/\textsf{false}}\wedge\neg\tau(\varphi)_{\textsf{leaf}}]\ \textsf{recurse}\\
&\qquad\quad \textsf{desc-or-self}::*[\tau(\varphi)^{X/(X\vee\tau(\varphi)_{\textsf{leaf}})}\wedge\neg\tau(\varphi)_{\textsf{leaf}}]\cup\\
&\qquad\quad \textsf{self}::*[X\vee\tau(\varphi)_{\textsf{leaf}}]/\textsf{desc-or-self}::*\big)[\textsf{leaf}]\rangle\\
&\text{where } \tau(\varphi)_{\textsf{leaf}} = \tau(\varphi)^{X/\textsf{false}}\wedge\textsf{leaf}
\end{aligned}\tag{3.1}$$

The last clause of the above translation follows exactly the strategy that we described above: $\tau(\varphi)_{\textsf{leaf}}$ computes the set Y of leaf-nodes belonging to the inflationary fixed point of φ, and every time the body of the fixed point operator is evaluated, the nodes in Y are added to the input and subtracted from the output, until possibly at some stage the context node comes to satisfy φ, in which case all its descendants are added to the fixed point set. Finally, we test whether the resulting fixed point set contains a leaf. The latter holds if and only if the context node satisfies $\big(\textsf{ifp}\ X\leftarrow\varphi\big)$.

Theorem 3.2. *Let $T=(N,R,L)$ be a node-labeled finite tree, g an assignment, and u a node in T. Then $T,g,u\Vdash\varphi$ iff $T,g,u\Vdash\tau(\varphi)$.*

Proof. The proof is by simultaneous induction on the form of the formula φ. The cases for $\bot$, l, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\neg\psi$, X, and $\Diamond\psi$ are immediate. Therefore, let $\varphi = \big(\mathsf{ifp}\ X \leftarrow \psi\big)$.

Since ML+IFP formulas and CXP+IFP node expressions can only see the subtree of the context node, we may assume without loss of generality that u is the root of the tree T. We write T_u instead of T, to make this explicit.

Let g_i, $0 \leq i \leq k$, be the variable assignments computed for φ in accordance with the semantics of the IFP operator (see Def. 2.2) on T_u, where $g_0 = g[X \mapsto \emptyset]$ and where k is the least natural number such that either $u \in g_k(X)$ or $g_k(X) = g_{k+1}(X)$, whichever happens first. Similarly, let g'_i, $1 \leq i \leq k$, be the first k variable assignments computed for the recursive sub-expression of $\tau(\varphi)$ in accordance with the semantics of the WITH operator (see Def. 2.1) for u as the context.

Let $Y = [\![\tau(\varphi)_{\mathsf{leaf}}]\!]_{T,g}$. It follows from the induction hypothesis, together with Lemma 3.1, that Y is precisely the set of all leaf nodes belonging to $g_k(X)$, and moreover, for every $1 \leq i \leq k$, Y is precisely the set of all leaf nodes belonging to $g_i(X)$.

Now, a straightforward induction shows that, for each $i \leq k$, $g_i(X)$ is exactly the disjoint union of the sets $g'_i(X)$ and Y (note that we use here the fact that $u \notin g_{k-1}(X)$).

Now, there are two possibilities: either $u \in g_k(X)$ (in which case u satisfies $\big(\mathsf{ifp}\ X \leftarrow \varphi\big)$) or $u \notin g_k(X)$ (in which case u does not satisfy $\big(\mathsf{ifp}\ X \leftarrow \varphi\big)$). In the first case, it is easy to see that $g'_{k+1}(X)$ contains all nodes below u, and in particular, contains a leaf node, and therefore $\tau(\big(\mathsf{ifp}\ X \leftarrow \varphi\big))$ is satisfied. In the second case, $g'_{k+1}(X) = g'_k(X)$, and therefore u does not satisfy $\tau(\big(\mathsf{ifp}\ X \leftarrow \varphi\big))$. This concludes the proof. □

We can conclude that CXP+IFP node expressions have (at least) the expressive power of ML+IFP. Since the desc axis is definable from the child axis, the same holds of course for the fragment of CXP+IFP without the desc axis. What is more surprising is that the same holds for the fragment of CXP+IFP without the child axis. The next lemma shows that the use of the child axis in the above translation can be avoided (provided that we keep, of course, the desc axis). Note that the child axis was only used in the translation of formulas of the form $\Diamond\varphi$.

Proposition 3.3. *For any node expression φ, $\langle \mathit{child}{::}*[\varphi]\rangle$ is equivalent to the following node expression (which does not use the child axis):*

$$\begin{gathered}
\langle\Big(\mathit{with}\ X\ \mathit{in}\ \mathit{desc}{::}*/\mathit{desc}{::}*[\mathit{leaf}]\ \mathit{recurse}\\
\mathit{self}{::}*[\langle\mathit{desc}{::}*[\mathit{leaf} \wedge \neg X \wedge \varphi]\rangle]\Big)[\neg\mathit{leaf}]\rangle\\
\vee\\
\langle\Big(\mathit{with}\ X\ \mathit{in}\ \mathit{desc}{::}*/\mathit{desc}{::}*[\neg\mathit{leaf}]\ \mathit{recurse}\\
\mathit{desc}{::}*[\neg\mathit{leaf} \wedge \neg X \wedge \varphi]/\mathit{desc}{::}*\Big)[\mathit{leaf}]\rangle
\end{gathered}$$

Proof. Let $T = (N, R, L)$ be a finite nodel-labeled tree, $u \in N$ a node, and g: $VAR \to \wp(N)$ an assignment. We will show that the first disjunct of the node expression is true at u under the assignment g if and only if u has a child satisfying φ (under g) that is a leaf. Similarly, it can be shown that the second disjunct is true if and only if u has a child satisfying φ that is not a leaf.

Thus, let us consider the first disjunct. In the first step of the inflationary fixed point computation, all leaf nodes below u are added to the set X except those that are a child of u. Next, u itself is added to the set X just in case it has a descendant satisfying φ that is a leaf and that was not marked with X already. After these two steps, the fixed point is reached. It is easy to see that the set X obtained in this way contains a non-leaf node if and only if it contains the node u, which holds if and only if u has a descendant satisfying φ that is a leaf and that was nor marked by X in the first step of the fixed point computation. The latter holds if and only if u has a child that is a leaf and that satisfies φ.

In the same way, for the second disjunct of the node expression, it can be shown that the inflationary fixed point set X contains a leaf node if and only if u has a child satisfying φ that is not a leaf. $\square$

4. Undecidability of CXP+IFP and ML+IFP on finite trees

We show that the satisfiability problem for ML+IFP on finite trees is undecidable, and therefore also (by our earlier translation), the satisfiability problem for CXP+IFP.

Theorem 4.1. *The satisfiability problem of ML+IFP on finite trees is undecidable.*

Corollary 4.2. *The satisfiability problem of CXP+IFP is undecidable, even if the* child *axis is disallowed.*

The proof, given in Section 4.2, is based on a reduction from the halting problem for 2-register machines (*cf.* [5]).

4.1. Two-Register machines

A 2-register machine is a very simple kind of deterministic automaton without input and output. It has two registers containing integer values, and instructions for incrementing and decrementing the content of the registers. These 2-register automata form one of the simplest types of machines for which the halting problem is undecidable. The formal definition is as follows:

A *2-register machine* M is a tuple $M = (Q, \delta, q_0, q_f)$, where Q is a finite set of *states*, δ is a *transition function* from Q to a set of instructions I, defined below, and q_0, q_f are designated states in Q, called *the initial state* and *the final state*, respectively [5].

The set of *instructions* I consists of four kinds of instructions:

- $INC_A(q')$: increment the value stored in A and move to state q';
- $INC_B(q')$: increment the value stored in B and move to state q';
- $DEC_A(q', q'')$: if the value stored in A is bigger than 0 then decrement it by one and move to state q', otherwise move to state q'' without changing the value in A nor B; and
- $DEC_B(q', q'')$: if the value stored in B is bigger than 0 then decrement it by one and move to state q', otherwise move to state q'' without changing the value in A nor B.

A *configuration* of the machine M is a triple $S = (q, a, b)$, where q is a state in Q, and a, b are non-negative integers that correspond to the numbers stored in the registers A and B, respectively. The configuration $S_0 = (q_0, 0, 0)$ is called the *initial configuration*, and the configuration $S_f = (q_f, 0, 0)$ is called the *final configuration.*

A *successful run* of the machine is a sequence of configurations, $S_i = (q_i, a_i, b_i)$, $0 \leq i \leq n$, $n > 0$, such that:

- the sequence starts with the initial configuration, S_0, and it ends with the final configuration S_f, and
- any pair of consecutive configurations in the sequence, S_i and S_{i+1}, *satisfies* δ, *i.e.*, the state and the register values in the successor configuration correspond to the instruction attributed to the state in the predecessor configuration by δ:
  - if $\delta(q_i) = INC_A(q'_i)$, then $S_{i+1} = (q'_i, a_i + 1, b_i)$;
  - if $\delta(q_i) = INC_B(q'_i)$, then $S_{i+1} = (q'_i, a_i, b_i + 1)$;
  - if $\delta(q_i) = DEC_A(q'_i, q''_i)$, then $S_{i+1} = (q'_i, a_i - 1, b_i)$, if $a_i > 0$, else $S_{i+1} = (q''_i, a_i, b_i)$;
  - if $\delta(q_i) = DEC_B(q'_i, q''_i)$, then $S_{i+1} = (q'_i, a_i, b_i - 1)$, if $b_i > 0$, else $S_{i+1} = (q''_i, a_i, b_i)$.

Theorem 4.3 ([5])**.** *The following question is undecidable:* given a 2-register machine, is there a successful run of this machine?

Note that, since a 2-register machine is deterministic without input, it can have only one minimal successful run, and any other successful other run must contain the first one as a prefix. We may in fact assume without loss of generality that the machine does not pass through the final configuration $(q_f, 0, 0)$ more than once, and hence has at most one successful run. Two further assumptions we can safely make are: (i) the initial and final states are distinct (if $q_0 = q_f$ then the machine trivially has a successful run), and (ii) no two subsequent configurations on any run of the machine have the same state (this can be ensured by adding additional intermediate states if necessary).

4.2. The reduction

In this section, we construct a ML+IFP formula that is satisfied in the root of a finite labeled tree *if and only if* all the paths from the root to the leaves represent a successful run of a given 2-register machine. If a successful run exists, then there is a tree that satisfies this formula, and vice versa, if there is no successful run, then there is no tree that satisfies the formula.

For the remainder of this section, fix a 2-register machine $M = (Q, \delta, q_0, q_f)$. The set of labels used in the formula will be $\Sigma = Q \cup \{a, b, \$\}$, where Q is the set of states of the 2-register machine, a, b are symbols used for representing the register content in each configuration, and \$ is another symbol used for marking the end of the encoding of the successful run. It is convenient in what follows to treat these labels as mutually exclusive. In other words, when we write a symbol such as a, we actually mean $a \wedge \bigwedge_{c \in \Sigma \setminus \{a\}} \neg c$.

We model the registers A and B of a 2-register machine with paths labeled with a and b, respectively. The number of nodes in the path corresponds to the integer number stored in the respective register. Then we prove that we can express the equality of two register values in ML+IFP. This is needed in order to express that two configurations of the machine satisfy the transition function δ. Once we can express that two configurations satisfy the transition function, we construct a formula that forces the existence of a sequence of configurations that forms a successful run.

It will be convenient to consider regular expressions describing paths in the tree. By a *path* in a tree, we mean a sequence of nodes $v_1, \dots, v_n$ $(n \geq 1)$ such that any two consecutive nodes satisfy the child relation, *i.e.*, $(v_i, v_{i+1}) \in R$, for $1 \leq i \leq n-1$. A path that ends with a leaf node is called a *branch*. A prefix of a path $v_1, \dots, v_n$ is any path $v_1, \dots, v_i$ with $i \leq n$. In order to describe paths, we will use expressions built up inductively from ML+IFP formulas using the regular operations of composition, union $(+)$, and transitive closure $(\cdot^+)$ as well as reflexive transitive closure $(\cdot^*)$. We call such expressions *regular path expressions*. For example, $a(\neg a)$ is a regular path expression that is satisfied by paths of length two whose first node satisfies a and whose second node does not, and $(\top\top)^*$ is a regular path expression that is satisfied by paths of even length.

We want to build a formula that describes a successful run of a given 2-register machine. For this purpose, we encode a configuration of this machine, $S = (q, n, m)$, $n, m \geq 0$, with a path that satisfies $qa^{n+1}b^{m+1}$, i.e, we represent the values n and m stored in the A and B registers with a sequence of $n+1$ a-labels and a sequence of $m+1$ b-labels. A sequence of configurations $S_1, \dots, S_k$ is encoded by a path that satisfies $q_1 a^{n_1+1} b^{m_1+1} \dots q_k a^{n_k+1} b^{m_k+1}\$$, where \$ is a special label indicating the end of the sequence. In order to describe a successful run, we first build a formula describing that a pair of configurations satisfy the transition function δ of the given machine, then we build a formula that ensures that every consecutive pair of configurations satisfies δ. In order to describe a pair of configurations that satisfy δ, we need to be able to express the equality constraints that δ enforces on the register values before and after a transition. For example, for $\delta(q) = INC_A(q')$ and two configurations that satisfy δ, $S = (q, n, m)$ and $S' = (q', n+1, m)$, $n, m > 0$, we need to express that a path satisfies $qa^n b^m q' a^{n+1} b^m$.

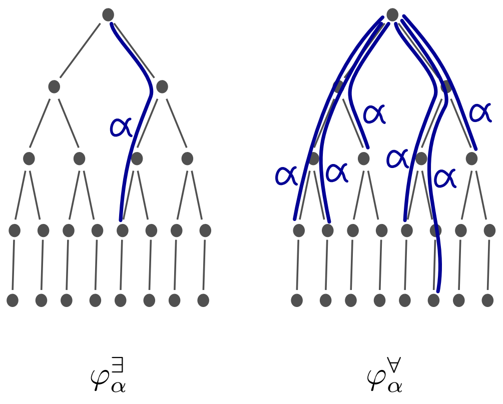

FIGURE 1. Graphical depiction of the meaning of $\varphi_\alpha^\exists$ and $\varphi_\alpha^\forall$.

Below, in Lemma 4.5, we show a generic formula that expresses the equality of the lengths of two a-labeled sequences. But first, we prove an auxiliary lemma that helps with the notations and interpretation of the formulas.

Lemma 4.4. *Let α be any regular path expression. Then there are* ML+IFP *formulas $\varphi_\alpha^\exists$ and $\varphi_\alpha^\forall$ such that for any finite labeled tree T and node v,*

1. *$T, v \Vdash \varphi_\alpha^\exists$ iff there is a path starting with v satisfying α, and*
2. *$T, v \Vdash \varphi_\alpha^\forall$ iff every branch starting with v has a prefix-path that satisfies α.*

(cf. Fig. 1)

Proof. We know that the statement holds for the modal μ-calculus (it follows from the fact that the modal μ-calculus is the bisimulation invariant fragment of MSO on finite trees [14]). To see that it holds also for ML+IFP we proceed as follows:

Let an expression α be given. First we replace each ML+IFP formula ψ occurring in α by a new corresponding fresh propositional variable p_ψ. Let the resulting expression be α'. Then, clearly, α' is an expression built up from formulas of the modal μ-calculus using concatenation, union, and star. Hence, there are formulas $\varphi_{\alpha'}^\exists$ and $\varphi_{\alpha'}^\forall$ of the modal μ-calculus satisfying the required conditions with respect to α'. Now, replace each p_ψ back by the original formula ψ (making sure that no free occurrences of variables in ψ accidentally get bound by a fixed point operator during the substitution – this can be ensured by renaming bound variables appropriately). It follows that the resulting formulas satisfy the required conditions with respect to α. □

In the following, we rely heavily on this lemma.

Lemma 4.5. *There is a formula, $\varphi_{a^n b a^n c}^\forall$, such that for any finite labeled tree T and a node v in this tree, $T, v \Vdash \varphi_{a^n b a^n c}^\forall$ iff there is a $k > 0$ such that every branch starting with v has a prefix-path that satisfies $a^k b a^k c$.*

Proof. In [7], a formula was constructed that, on finite strings (*i.e.*, finite trees in which each node has at most one child), defines the language $a^n b^{\geq n}$. Our construction below differs from the one in [7] in that our formula expresses exact equality, and, more importantly, in the fact that it works on arbitrary finite trees, which makes it a non-trivial generalization.

We define $\varphi^{\forall}_{a^nba^nc}$ as in equation (4.1).

$$\varphi^{\forall}_{a^nba^nc} := \varphi^{\forall}_{a^nba^{\geq n}c} \wedge \neg\varphi^{\exists}_{a^nba^{>n}c+aa^+c}, \tag{4.1}$$

where

$$\varphi^{\forall}_{a^nba^{\geq n}c} := \varphi^{\forall}_{a^*ba^*c} \wedge \big(\mathsf{ifp}\ X \leftarrow \varphi^{\forall}_{a(a\wedge X)^*b(a\wedge\neg X)a^*c} \vee \varphi^{\exists}_{a(a\wedge X)^*c}\big)$$

$$\varphi^{\exists}_{a^nba^{>n}c+aa^+c} := \big(\mathsf{ifp}\ X \leftarrow \varphi^{\exists}_{a(a\wedge X)^*b(a\wedge\neg X)a^*ac} \vee \varphi^{\exists}_{a(a\wedge X)^*ac}\big).$$

The idea behind the two conjuncts of the formula is that $\varphi^{\forall}_{a^nba^{\geq n}c}$ expresses something slightly too weak, since it only enforces the second sequence of a-nodes on each path to be *at least* as long as the first sequence. The second conjunct $\neg\varphi^{\exists}_{a^nba^{>n}c+aa^+c}$ corrects for this by enforcing that there is no path on which the second sequence of a-nodes is strictly longer than the first sequence. For technical reasons, the formula $\varphi^{\exists}_{a^nba^{>n}c+aa^+c}$ in question expresses something weaker than the existence of a path satisfying $a^nba^{>n}c$: it also accepts nodes where a path starts satisfying aa^+c. However, this clearly does not affect the correctness of the overall formula $\varphi^{\forall}_{a^nba^nc}$.

Below, we prove that the formulas $\varphi^{\forall}_{a^nba^{\geq n}c}$ and $\varphi^{\exists}_{a^nba^{>n}c+aa^+c}$ do indeed have the intended meaning, which is captured by the following claims:

1. $T, v \Vdash \varphi^{\forall}_{a^nba^{\geq n}c}$ iff $\exists k$, such that every branch starting with v has a prefix-path that satisfies $a^{\leq k}ba^{\geq k}c$.
2. $T, v \Vdash \varphi^{\exists}_{a^nba^{>n}c+aa^+c}$ iff there exists a path starting with v that satisfies a^nba^mc, for some $m > n > 0$, or that satisfies a^nc, for some $n \geq 2$.

It is clear from the above discussion that the lemma follows directly from (1) and (2). Below we give the proof for (1). The proof for (2) is similar. Note that we rely on Lemma 4.4 for the existence and the semantics of the formulas $\varphi^{\exists}_{a(a\wedge X)^*b(a\wedge\neg X)}$, $\varphi^{\forall}_{a(a\wedge X)^*b(a\wedge\neg X)}$, $\varphi^{\exists}_{a(a\wedge X)^*ac}$, and $\varphi^{\forall}_{a^*ba^*c}$.

Let $g(\cdot)$ be a variable assignment and let $g_i(\cdot)$, $0 \leq i$, be the variable assignments obtained for $\big(\mathsf{ifp}\ X \leftarrow \varphi^{\forall}_{a(a\wedge X)^*b(a\wedge\neg X)} \vee \varphi^{\exists}_{a(a\wedge X)^*c}\big)$ in accordance with the semantics of the IFP operator (Def. 2.2), where $g_0 = g[X \mapsto \emptyset]$. First, we show, by induction on i, $0 < i$, the following:

i. A node in $[\![\varphi^{\exists}_{a^+c}]\!]_{T,g}$ is added to $g_i(X)$ at the recursion step i iff there is a path from that node that satisfies a^ic and there is no other path from that node that satisfies $a^{<i}c$ (*i.e.*, that satisfies a^j for some $j < i$).

Here, by "is added to $g_i(X)$ at the recursion step i", we mean that the node belongs to $g_i(X)$ and not to $g_{i-1}(X)$. Then, using this equivalence, again by induction on i,

$0 < i$, we show the following:

ii. A node in $[\![\varphi^{\forall}_{a^*ba^*c}]\!]_{T,g}$ is added to $g_i(X)$ at the recursion step i iff every branch starting with that node has a prefix-path that satisfies $a^{\leq i}ba^*a^ic$ and i is the least number with this property.

Suppose $u \in [\![\varphi^{\exists}_{a^+c}]\!]_{T,g}$. It is easy to see that u is added to $g_1(X)$ iff $u \in [\![\varphi^{\exists}_{a(a\wedge X)^*c}]\!]_{T,g[X\mapsto\emptyset]}$ iff there is a path starting with u that satisfies ac. Next, suppose that (i) holds for some $i \geq 1$. We show that (i) holds for $i+1$.

Suppose $u \in [\![\varphi^{\exists}_{a^+c}]\!]_{T,g}$ and u is added to $g_{i+1}(X)$. Then $u \in [\![\varphi^{\exists}_{a(a\wedge X)^*c}]\!]_{T,g_i}$. From this it follows that u is labeled with a and there is a successor w labeled with c or $w \in g_i(X)$. Note that $w \in [\![\varphi^{\exists}_{a^*c}]\!]_{T,g}$. In the first case, by induction hypothesis, u was added already to $g_1(X)$, which contradicts our assumption. In the second case, $w \in g_i(X)$ and w was added to $g_i(X)$ at the recursion step i, otherwise, by the same argument, u would be already in $g_i(X)$. By induction hypothesis, there is a path starting with w that satisfies a^ic and thus, there is a path starting with u that satisfies $a^{i+1}c$.

Conversely, suppose that there is a path from u that satisfies $a^{i+1}c$ and there is no other path from that node that satisfies $a^{<i+1}c$. Let w be the successor of u on that path. Then there is a path from w that satisfies a^ic and there is no other path from w that satisfies $a^{<i}c$. By induction hypothesis, w was added to $g_i(X)$ and thus, u must be added to $g_{i+1}(X)$.

This concludes the proof of (i). We now proceed with the proof of (ii).

Suppose $u \in [\![\varphi^{\forall}_{a^*ba^*c}]\!]_{T,g}$. Again it is easy to see that u is added to $g_1(X)$ iff $u \in [\![\varphi^{\forall}_{a(a\wedge X)^*b(a\wedge\neg X)a^*c}]\!]_{T,g[X\mapsto\emptyset]}$ iff every branch starting with u has a prefix-path that satisfies $abaa^*c$. Further, suppose that (ii) holds for i, $0 < i$. We show that (ii) holds for $i+1$.

Suppose $u \in [\![\varphi^{\forall}_{a^*ba^*c}]\!]_{T,g}$ and u is added to $g_{i+1}(X)$. Then we know that $u \in [\![\varphi^{\forall}_{a(a\wedge X)^*b(a\wedge\neg X)a^*c}]\!]_{T,g_i}$ and thus, every successor w of u (there is at least one successor) is in $g_i(X)$. Suppose that w was added to $g_j(X)$ at iteration step $j \leq i$. By induction hypothesis, every branch from w has a prefix-path that satisfies $a^{\leq j}ba^*a^jc$, thus it satisfies $a^{\leq i}ba^*c$. By statement (i) proven above, every branch from u has prefix-path that satisfies $a^*ba^*a^{i+1}c$. From the last two statements it follows that every branch from u has a prefix-path that satisfies $a^{\leq i+1}ba^*a^{i+1}c$. Note that $i+1$ is the least number with this property, otherwise u would have been added at an earlier iteration step.

For the other direction, suppose that every branch from u has a prefix-path that satisfies $a^{\leq i+1}ba^*a^{i+1}c$ and $i+1$ is the least number with this property. From this, it follows that every branch from a successor w of u, has a prefix-path that satisfies $a^{\leq i}ba^*a^{i+1}c$, and hence $a^{\leq i}ba^*a^ic$. Let j be the least number with this property for w. Then, by induction hypothesis, it follows that w was added to $g_j(X)$ at iteration step j. Let j_0 be the least number such that $g_{j_0}(X)$ contains all successors of u. Note that j_0 equals i, otherwise every branch from u has a prefix-path that satisfies $a^{\leq i}ba^*a^ic$, which contradicts our initial assumption. From this, it follows that $u \in [\![\varphi^{\forall}_{a(a\wedge X)^{\leq i}ba^*a^ic}]\!]_{T,g_i}$ and i is the least number with this property. From the fact that every branch from u has a prefix-path that satisfies $a^{\leq i+1}ba^*a^{i+1}c$ and conform the statement (i) proven above, it follows that $u \in [\![\varphi^{\forall}_{a^*b(a\wedge\neg X)a^*(a\wedge X)^ic}]\!]_{T,g_i}$. From the last two statements it follows that u is added to $g_{i+1}(X)$ at iteration step $i+1$.

Now, based on (ii) we can prove the statement of (1): $T, v \Vdash \varphi^{\forall}_{a^nba^{\geq n}c}$ iff $v \in g_k(X) \cap [\![\varphi^{\forall}_{a^*ba^*c}]\!]_{T,g}$, where k is the recursive step at which the computation of the IFP operator ends $\iff \exists n_0, 0 < n_0 \leq k$, such that v was added to $g_{n_0}(X)$ at the iteration n_0 and $v \in [\![\varphi^{\forall}_{a^*ba^*c}]\!]_{T,g} \iff \exists n_0, 0 < n_0 \leq k$, every branch starting with that node has a prefix-path that satisfies a^nba^mc, for some $0 < n \leq n_0 \leq m$.

The proof for (2) is similar to the proof for (i). $\square$

Lemma 4.5 can be extended to other formulas of the form $\varphi^{\forall}_{\alpha a^n\beta a^n\gamma}$, where α, β, and γ are a certain kind of regular expressions denoting sequences of labels. We do not attempt a general result along these lines, but we consider instances of this general pattern for which it is clear that the proof of Lemma 4.5 works. In particular, in order for the proof of Lemma 4.5 to work, it is important that β and γ are incompatible, in the sense that they cannot be satisfied by the same path. Below, we give two examples of such formulas and state their semantics. We use these examples further on for our reduction.

Intuitively, the first example is a formula that describes a transition of the 2-register machine, in which register A is incremented. The second example describes a transition in which register A is decremented. The variable Y in these formulas is intended to represent that the remaining part of the run is already known to be correct (and will be bound by an IFP-operator).

Let $q, q', q'' \in Q$ such that $q \neq q'$ and $q \neq q''$, and let $Q'_Y := q' \wedge Y$, $Q''_Y := q'' \wedge Y$, and $E_Y := Y \vee \$$, where Y is a variable in VAR.

Example 4.6. Consider $\varphi^{\forall}_{qa^nb^*Q'_Ya^{n+1}b^*E_Y}$ defined by equation (4.2). Similarly to Lemma 4.5, we can prove that $\varphi^{\forall}_{qa^nb^*Q'_Ya^{n+1}b^*E_Y}$ is valid in a node v of a finite labeled tree T iff every branch starting with v has a prefix-path that satisfies $qa^nb^*Q'_Ya^{n+1}b^*E_Y$, for some $n > 0$.

$$\begin{aligned}
\varphi^{\forall}_{qa^nb^*Q'_Ya^{n+1}b^*E_Y} &:= \varphi^{\forall}_{qa^nb^*Q'_Ya^{\geq n+1}b^*E_Y} \wedge \\
&\quad\ \neg\varphi^{\exists}_{qa^nb^*Q'_Ya^{>n+1}b^*E_Y + a^*a^3b^*E_Y} \\
\varphi^{\forall}_{qa^nb^*Q'_Ya^{\geq n+1}b^*E_Y} &:= \varphi^{\forall}_{qa^*b^*Q'_Ya^*b^*E_Y} \wedge \\
&\quad\ \Box\big(\mathsf{ifp}\ X \leftarrow \varphi^{\forall}_{a(a\wedge X)^*b^*Q'_Y(a\wedge\neg X)a^*ab^*E_Y} \vee \\
&\quad\ \varphi^{\exists}_{a(a\wedge X)^*(a\wedge\neg X)b^*E_Y}\big) \\
\varphi^{\exists}_{qa^nb^*Q'_Ya^{>n+1}b^*E_Y + a^*a^3b^*E_Y} &:= q \wedge \Diamond\big(\mathsf{ifp}\ X \leftarrow \varphi^{\exists}_{a(a\wedge X)^*b^*Q'_Y(a\wedge\neg X)a^*a^2b^*E_Y} \vee \\
&\quad\ \varphi^{\exists}_{a(a\wedge X)^*(a\wedge\neg X)^2b^*E_Y}\big).
\end{aligned} \tag{4.2}$$

Example 4.7. For convenience, consider the following alternative notations for the until formulas ψ_1 EU $\psi_2 := \varphi^{\exists}_{\psi_1^*\psi_2}$ and ψ_1 AU $\psi_2 := \varphi^{\forall}_{\psi_1^*\psi_2}$. Consider

$\varphi^{\forall}_{qa^*b^{n+1}Q'_Ya^*b^nE_Y}$ defined by equation (4.3) and let $g(\cdot)$ be a variable assignment that covers Y. Similarly to Lemma 4.5, we can prove that $\varphi^{\forall}_{qa^*b^{n+1}Q'_Ya^*b^nE_Y}$ is valid in a node v of a finite labeled tree T and given $g(\cdot)$ $\iff$ every branch starting with v has a prefix-path that satisfies $qa^*b^{n+1}Q'_Ya^*b^nE_Y$, for some $n > 0$.

$$\begin{aligned}
\varphi^{\forall}_{qa^*b^{n+1}Q'_Ya^*b^nE_Y} &:= \varphi^{\forall}_{qa^*b^{n+1}Q'_Ya^*b^{\geq n}E_Y} \wedge \\
&\quad\ \neg\varphi^{\exists}_{qa^*b^{n+1}Q'_Ya^*b^{>n}E_Y+b^*b^2E_Y} \\
\varphi^{\forall}_{qa^*b^{n+1}Q'_Ya^*b^{\geq n}E_Y} &:= \varphi^{\forall}_{qa^*b^*Q'_Ya^*b^*E_Y} \wedge \big(a \ \mathsf{AU}\ (b\wedge \\
&\quad\ \big(\mathsf{ifp}\ X \leftarrow \varphi^{\forall}_{b(b\wedge X)^*Q'_Ya^*(b\wedge\neg X)b^*E_Y} \vee \varphi^{\exists}_{b(b\wedge X)^*E_Y}\big)\big)\big) \\
\varphi^{\exists}_{qa^*b^{n+1}Q'_Ya^*b^{>n}E_Y+b^*b^2E_Y} &:= q \wedge \big(a \ \mathsf{EU}\ (b\wedge \\
&\quad\ \big(\mathsf{ifp}\ X \leftarrow \varphi^{\exists}_{b(b\wedge X)^*Q'_Ya^*(b\wedge\neg X)b^*bE_Y} \vee \\
&\quad\ \varphi^{\exists}_{b(b\wedge X)^*(b\wedge\neg X)E_Y}\big)\big)\big).
\end{aligned} \tag{4.3}$$

In a similar fashion, we construct the following formulas:

$$\begin{array}{llll}
\varphi^{\forall}_{qa^nb^*Q'_Ya^nb^*E_Y} & \varphi^{\forall}_{qa^nb^*Q'_Ya^{n+1}b^*E_Y} & \varphi^{\forall}_{qa^{n+1}b^*Q'_Ya^nb^*E_Y} & \varphi^{\forall}_{qa^nbQ''_Ya^nbE_Y} \\
\varphi^{\forall}_{qa^*b^mQ'_Ya^*b^mE_Y} & \varphi^{\forall}_{qa^*b^mQ'_Ya^*b^{m+1}E_Y} & \varphi^{\forall}_{qa^*b^{m+1}Q'_Ya^*b^mE_Y} & \varphi^{\forall}_{qab^nQ''_Yab^nE_Y}
\end{array}$$

An equivalent of Lemma 4.5 can be proven for each of these formulas. Finally, we are ready to define the transition function of a given 2-register machine.

$$\begin{aligned}
\varphi^{\forall}_{qa^nb^mQ'_Ya^{n+1}b^mE_Y} &:= \varphi^{\forall}_{qa^nb^*Q'_Ya^{n+1}b^*E_Y} \wedge \varphi^{\forall}_{qa^*b^mQ'_Ya^*b^mE_Y} \\
\varphi^{\forall}_{qa^nb^mQ'_Ya^nb^{m+1}E_Y} &:= \varphi^{\forall}_{qa^nb^*Q'_Ya^nb^*E_Y} \wedge \varphi^{\forall}_{qa^*b^mQ'_Ya^*b^{m+1}E_Y} \\
\varphi^{\forall}_{qa^{n+1}b^mQ'_Ya^nb^mE_Y} &:= \varphi^{\forall}_{qa^{n+1}b^*Q'_Ya^nb^*E_Y} \wedge \varphi^{\forall}_{qa^*b^mQ'_Ya^*b^mE_Y} \\
\varphi^{\forall}_{qa^nb^{m+1}Q'_Ya^nb^mE_Y} &:= \varphi^{\forall}_{qa^nb^*Q'_Ya^nb^*E_Y} \wedge \varphi^{\forall}_{qa^*b^{m+1}Q'_Ya^*b^mE_Y}
\end{aligned} \tag{4.4}$$

$$\begin{aligned}
Tr_q(Y) &:= \begin{cases}
\varphi^{\forall}_{qa^nb^mQ'_Ya^{n+1}b^mE_Y} & \text{if } \ \delta(q) = INC_A(q'), \\
\varphi^{\forall}_{qa^nb^mQ'_Ya^nb^{m+1}E_Y} & \text{if } \ \delta(q) = INC_B(q'), \\
\varphi^{\forall}_{qa^{n+1}b^mQ'_Ya^nb^mE_Y} \vee \varphi^{\forall}_{qab^nQ''_Yab^nE_Y} & \text{if } \ \delta(q) = DEC_A(q', q''), \\
\varphi^{\forall}_{qa^nb^{m+1}Q'_Ya^nb^mE_Y} \vee \varphi^{\forall}_{qa^nbQ''_Ya^nbE_Y} & \text{if } \ \delta(q) = DEC_B(q', q'')
\end{cases} \\
Tr(Y) &:= \bigvee\nolimits_{q\in Q} Tr_q(Y).
\end{aligned} \tag{4.5}$$

Recall from Section 4.1 that we assume without loss of generality that the 2-register machine M is such that no two subsequent configurations on a run have the same state and therefore q' and q'' are always distinct from q in the formulas in (4.5). Thus, generalizing from the above examples and the proof of Lemma 4.5, we have the following.

Lemma 4.8. *Let $T = (N, R, L)$ be a finite labeled tree and $g(\cdot)$ be a variable assignment that covers the free variable Y. Then $T, v, g \Vdash Tr(Y)$ iff every branch starting with v has a prefix-path that satisfies $qa^n b^m q' a^{n'} b^{m'} E_Y$, for some pair of triples, $S = (q, n, m)$ and $S' = (q', n', m')$, that satisfies δ $(n, n', m, m' > 0)$.*

Having the formula that describes a transition, we can build the formula φ_{run}, below, that describes a successful run of a given 2-register machine; φ_{run} enforces that every branch starting from a node in the tree represents a successful run of the given machine.

$$\begin{aligned} Q_s &:= \varphi^{\forall}_{q_s ab(\bigvee_{q \in Q} q)} \\ Q_f &:= \varphi^{\forall}_{q_f ab\$} \\ \varphi_{run} &:= Q_s \wedge \big(\mathsf{ifp}\ Y \leftarrow Tr(Y) \vee Q_f\big). \end{aligned} \tag{4.6}$$

Theorem 4.9. *The formula φ_{run} is satisfiable iff the 2-register machine M has a successful run.*

Proof. ($\Longleftarrow$) Suppose that the sequence of configurations, $S_1, \ldots, S_n$, $n > 0$, is a successful run of M, with $S_i = (q_i, k_i, \ell_i)$. In particular, $S_1 = (q_0, 0, 0)$ is the initial configuration and $S_n = (q_f, 0, 0)$ is the final configuration. Also, as explained in Section 4.1, we may assume that $n > 1$, and that $q_i \neq q_{i+1}$ for $1 \leq i < n$. Let T be the tree consisting of a single branch, such that the sequence of labels of the nodes on the branch forms the string $q_1 a^{k_1+1} b^{\ell_1+1} \cdots q_n a^{k_n+1} b^{k_n+1}\$$. Let u_i (for $1 \leq i \leq n$) be the i-th node on the branch whose label belongs to Q. It is clear that the root of the tree, u_1, satisfies Q_s, and that u_n satisfies Q_f. Furthermore, for each $i \leq n$, $T, u_i \Vdash \big(\mathsf{ifp}\ Y \leftarrow Tr(Y) \vee Q_f\big)$ as can be shown by a straightforward induction on $n - i$. It follows that $T, u_1 \Vdash \varphi_{run}$.

($\Longrightarrow$) Suppose $T, v \Vdash \varphi_{run}$. Let $g(\cdot)$ be a variable assignment. Since $T, v \Vdash \big(\mathsf{ifp}\ Y \leftarrow Tr(Y) \vee Q_f\big)$ we have that $v \in g_k(Y)$, where $g_k(\cdot)$ is the last variable assignment obtained following the definition of the IFP operator (Def. 2.2) for $\big(\mathsf{ifp}\ Y \leftarrow Tr(Y) \vee Q_f\big)$. One can show by a straightforward induction on i, $1 \leq i \leq k$, that for every branch starting with a node $u \in g_i(Y)$, the sequence of labels of the nodes on this branch, up to the first node satisfying \$, forms an encoding of a run of the 2-register machine, starting in some (not necessarily initial) configuration and ending in the final configuration. In particular, since $v \in g_k(Y)$, and also $T, v \Vdash Q_s$, we then have that for every branch starting at v, the sequence of labels of the nodes on this branch, up to the first node saytisfying \$, forms an encoding of a successful run of the 2-register machine (starting in the initial configuration). $\square$

We have shown that the undecidable halting problem for 2-register machines reduces to the satisfiability problem for ML+IFP on finite trees. Theorem 4.1 now follows.

5. Discussions and conclusions

We proved that the fragment of CXP+IFP with only *self* and *descendant* axes is undecidable. This implies the undecidability of CXP+IFP with all the axes. Moreover, since the undecidability proof for ML+IFP, as well as the translation from ML+IFP into CXP+IFP, works on strings too, no matter what axis one takes (along with the *self* axis) the fragment of CXP+IFP with only that axis is undecidable. Recall that the transitive axes (*e.g.*, *descendant*, *ancestor*, *following-sibling*, *preceding-sibling*) are easily defined from the corresponding non-transitive axes using the IFP operator.

This result means that a complete static analysis of recursive queries specified by means of the IFP operator is not feasible. In other words, we cannot do better than implementing sound-but-not-complete query optimizations, such as the distributivity-based optimization presented in [1, 2].

Another recursion operator that has been studied extensively in the context of CXP, is the *transitive closure* (TC) of path expressions and the language is known as Regular XPath [15]. Note that we can express the transitive closure of a path expression α by using the IFP operator as follows:

$$\alpha^+ = \textsf{with } X \textsf{ in } \alpha \textsf{ recurse } X/\alpha$$

Regular XPath falls within monadic second-order logic (MSO) [17], while CXP+IFP can define (among all finite strings) non-regular string languages [7] (*cf.* also Lem. 4.5).

that satisfy $a^n b^n c$, $n > 0$, which is not a regular string language and thus not definable in MSO [18]. From this it follows that CXP+IFP is strictly more expressive than Regular XPath.

Note that the definition of the TC operator *via* IFP does not use negation on the recursion variable. Thus the TC operator can be expressed also *via* a *least fixed point* (LFP) operator, which is a non-inflationary fixed point operator that does not allow the recursion variable to occur under an odd number of negations. If we consider CXP extended with LFP, then this language still falls within MSO and is at least as expressive as Regular XPath but strictly less expressive than CXP+IFP on finite trees.

In conclusion, when choosing a recursion operator to extend CXP, one should keep in mind that the inflationary fixed point operator is the most expressive and expensive operator (with undecidable static analysis problems) of the three recursion operators discussed above.

5.1. Remaining questions

One natural follow-up question is whether CXP+IFP node expressions are strictly more expressive than ML+IFP formulas.

Other natural follow-up questions concern fragments of CXP+IFP. Recall that in CXP+IFP, the variables can be used both as atomic path expressions and as atomic node expressions. The former is the most natural, but the translation we gave from ML+IFP to CXP+IFP crucially uses the latter. Our conjecture is that the fragment of CXP+IFP in which variables are only allowed as atomic path expressions is also undecidable.

It is also natural to consider CXP+IFP expressions where the fixed point variables occur only under an even number of negations, so that the WITH-operator computes the least fixed point of a monotone operation. Note that this fragment is decidable, since it is contained in monadic second-order logic. Thus, further questions like the complexity of the static analysis problems and the expressive power of this language are open to investigation.

References

[1] L. Afanasiev, T. Grust, M.J. Marx, J. Rittinger and J. Teubner, An inflationary fixed point operator in XQuery, in *Proc. of 24th Int. Conf. on Data Engineering, ICDE '08 (Cancun, Apr. 2008)*. IEEE CS Press (2008) 1504–1506.

[2] L. Afanasiev, T. Grust, M.J. Marx, J. Rittinger and J. Teubner, Recursion in XQuery: Put your distributivity safety belt on, in *Proc. of 12th Int. Conf. on Extending Database Technology, EDBT '09 (St. Petersburg, March 2009)*. ACM Press (2009) 345–356.

[3] P. Blackburn, M. de Rijke and Y. Venema, Modal Logic, *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press **53** (2002).

[4] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger and J. Teuber, MonetDB/XQuery: a fast XQuery processor powered by a relational engine, in *Proc. of 25th ACM SIGMOD Int. Conf. on Management of Data, SIGMOD '06 (Chicago, IL, June 2006)*. ACM Press (2006) 479–490.

[5] E. Börger, E. Grädel and Y. Gurevich, The Classical Decision Problem. Springer (1997).

[6] J. Bradfield and C. Stirling, Modal μ-calculi, in Handbook of Modal Logic, edited by P. Blackburn, J. van Benthem and F. Wolter. Elsevier (2007) 721–756.

[7] A. Dawar, E. Grädel and S. Kreutzer, Inflationary fixed points in modal logic. *ACM Trans. Comput. Log.* **5** (2004) 282–315.

[8] T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele and T. Westmann, Anatomy of a native XML base management system. *VLDB J.* **11** (2002) 292–314.

[9] Georgetown Protein Information Resource, Protein sequence database (2001). Available on http://www.cs.washington.edu/research/xmldatasets/

[10] G. Gottlob and C. Koch, Monadic queries over tree-structured data, in *Proc. of 17th Ann. IEEE Symp. on Logic in Computer Science, LICS '02 (Copenhagen, July 2002)*. IEEE CS Press (2002) 189–202.

[11] E. Grädel, M. Otto and E. Rosen, Undecidability results on two-variable logics, in *Proc. of 14th Ann. Symp. on Theoretical Aspects of Computer Science, STACS '97 (Lübeck, Feb./March 1997), Lect. Notes Comput. Sci.* Vol. 1200, edited by R. Reischuk and M. Morvan, Springer (1997) 249–260.

[12] N. Immerman, *Descriptive Complexity*. Springer (1999).

[13] H.V. Jagadish, L.V.S. Lakshmanan, D. Srivastava and K. Thompson, TAX: A tree algebra for XML, in *Revised Papers from 8th Int. Workshop on Database Programming Languages, DBPL 2001 (Frascati, Sept. 2001)*, *Lect. Notes Comput. Sci.* Vol. 2397, edited by G. Ghelli and G. Grahne, Springer (2002) 149–164.

[14] D. Janin and I. Walukiewicz, On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic, in *Proc. of 7th Int. Conf. on Concurrency Theory, CONCUR 1996 (Pisa, Aug. 1996)*, *Lect. Notes Comput. Sci.* Vol. 1119, edited by U. Montanari and V. Sassone, Springer (1996) 263–277.

[15] M. Marx, XPath with conditional axis relations, in *Proc. of 9th Int. Conf. on Extending Database Technology, EDBT 2004 (Heraclion, March 2004)*, *Lect. Notes Comput. Sci.* Vol. 2992, edited by E. Bertino et al., Springer (2004) 477–494.
[16] M. Marx and M. de Rijke, Semantic characterizations of navigational XPath. *ACM SIGMOD Record* **34** (2005) 41–46.
[17] B. ten Cate, *Regular XPath: algebra, logic and automata,* unpublished note presented at AUTOMATHA Workshop on Algebraic Theory of Automata and Logic (2006).
[18] W. Thomas, Languages, automata, and logic, in *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, edited by G. Rozenberg and A. Salomaa. Springer (1997) 389–455.
[19] XML path language (XPath) version 1.0, edited by J. Clark and S. DeRose. World Wide Web Consortium (1999). http://www.w3.org/TR/xpath/.
[20] XML path language (XPath) 2.0, edited by A. Berglund et al. World Wide Web Consortium (2007). Available on http://www.w3.org/TR/xpath20/.
[21] XQuery 1.0: An XML query language, edited by S. Boag et al. World Wide Web Consortium (2007). Available on http://www.w3.org/TR/xquery/.

Communicated by R. Matthes and T. Uustalu.
Received September 28, 2012. Accepted September 28, 2012.