On Core XPath with Inflationary Fixed Points

Loredana Afanasiev
Informatics Institute, University of Amsterdam,
Science Park 107, NL-1098 XG Amsterdam, The Netherlands
lafanasi@science.uva.nl
Balder ten Cate
INRIA Saclay - Île-de-France and ENS de Cachan,
61 avenue du President Wilson, F-94235 Cachan Cedex, France
balder.tencate@gmail.com

Abstract

In this report, we prove the undecidability of Core XPath 1.0 (CXP) [6] extended with an *Inflationary Fixed Point (IFP)* operator. 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.

1 Introduction

In [1], an extension of the XML query language XQuery with an inflationary fixed point operator was proposed and studied. The motivation for this study stems from practical use cases. The existing mechanism in XQuery for expressive recursive queries (i.e., user defined recursive functions) is procedural in nature, which makes queries both hard to write and hard to optimize. 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 [3]. Indeed, it was shown in [1] that a significant performance gain can be achieved in this way.

While the empirical evidence is there, a foundational question remains: how feasible it is to do static analysis for recursive queries specified by means of the fixed point operator. Specifically, are there substantial fragments of XQuery with the 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. The proof is based on a reduction from the undecidable halting problem for 2-register machines (cf. [4]), and borrows ideas from the work of Dawar et al. [5] 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 ways. Nevertheless, we obtain a translation from 1MIC (the fragment of MIC that does not involve simultaneous induction) to CXP+IFP node expressions.

In [5], after showing that the satisfiability problem for MIC on arbitrary structures is highly undecidable, the authors ask whether there are still useful fragments, and also whether the logic has any relevance for practical applications. Our results shed some light on these questions. We obtain as a part of our investigation that the satisfiability problem for 1MIC is already undecidable on finite trees, and the relationship between MIC and CXP+IFP adds relevance to the study of MIC.

2 Preliminaries

2.1 Core XPath 1.0 Extended with IFP (CXP+IFP)

Core XPath 1.0 (CXP) was introduced in [6] to capture the navigational core of XPath 1.0. The definition that we use here differs slightly from the one of [6]. We 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 suffice already for our undecidability result. We will briefly comment on the other axes later. Other differences with [6] are that we allow filters and union to be applied to any expressions.

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] in the context of XQuery, and is here naturally adapted to the setting of CXP. We first give the syntax and semantics of CXP+IFP, and 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:

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

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 nodes, $g : VAR \to \wp(N)$. Then the semantics of CXP+IFP expressions are as follows:

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

◁

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 $Result_u^g(\alpha)$. 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:

> $Result_u^g(X) = g(X)$, i.e., when X is used as a path expression, it evaluates to $g(X)$ regardless of the context node.
>
> $Result_u^g(\textsf{with } X \textsf{ in } \alpha_1 \textsf{ recurse } \alpha_2) = X_k$, where $X_1 = Result_u^{g[X\mapsto\emptyset]}(\alpha_1)$, $X_{i+1} = X_i \cup Result_u^{g[X\mapsto X_i]}(\alpha_2)$ 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.

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

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

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

It is important to note that (unlike MIC) the language provides no way to test whether a given node belongs to the result of with X in α_1 recurse α_2, it only allows 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, it has some subtle consequences.

We remark that semantics of the WITH operator we give here differs slighly from the original semantics used in [1]. According to the original semantics, when $Result_u^g(\textsf{with } \alpha_1 \textsf{ in } \alpha_2 \textsf{ recurse })$ 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, $Result_u^g(\textsf{with } X \textsf{ in } \alpha_1 \textsf{ recurse } \alpha_2)$ was defined there as X_k, where $X_0 = Result_u^{g[X\mapsto\emptyset]}(\alpha_1)$, $X_1 = Result_u^{g[X\mapsto X_0]}(\alpha_2)$, $X_{i+1} = X_i \cup Result_u^{g[X\mapsto X_i]}(\alpha_2)$ 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 more clean and 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) [2] 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 [5], 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:

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

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. 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 to each 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:

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

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

It was shown in [5] that the satisfiability problem for ML+IFP on arbitrary Kripke models is highly undecidable. As we will show below, it is undecidable on finite trees as well.

3 Relationship between ML+IFP and CXP+IFP

In this section, we give a truth-preserving translation from ML+IFP to CXP+IFP. In fact, the translation yields CXP+IFP expressions that use only the *self* and *descendant* axes. It follows that this fragment of CXP+IFP has already (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 for 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 separely. More precisely, 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 leafs 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 $u \in [\![\mathit{ifp}\, X \leftarrow \varphi]\!]_{T,g}$ iff $u \in [\![\varphi]\!]_{T,g[X\mapsto\emptyset]}$.*

In what follows we will use $\oslash$ as shorthand for $\mathsf{self}::*[\mathsf{false}]$, $\mathsf{desc\text{-}or\text{-}self}::*$ as shorthand for $\mathsf{desc}::* \cup \mathsf{self}::*$, and leaf as shorthand for $\neg\langle\mathsf{child}::*\rangle$. 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 by replacing all free occurrences of X in φ by the node expression ψ.

The translation $\tau(\cdot)$ from ML+IFP formulas to CXP+IFP node expressions is given by Equation (1).

$$
\begin{array}{rcl}
\tau(\bot) & = & \mathsf{false} \\
\tau(l) & = & \langle \mathsf{self}::l \rangle \\
\tau(\varphi_1 \wedge \varphi_2) & = & \tau(\varphi_1) \wedge \tau(\varphi_2) \\
\tau(\neg\varphi) & = & \neg\tau(\varphi) \\
\tau(X) & = & X \\
\tau(\Diamond\varphi) & = & \langle \mathsf{child}::*[\tau(\varphi)] \rangle \\
\tau\big(\mathsf{ifp}\, X \leftarrow \varphi\big) & = & \langle \big(\mathsf{with}\, X\, \mathsf{in}\, \mathsf{desc\text{-}or\text{-}self}::^\star[\tau(\varphi)^{X/\mathsf{false}} \wedge \neg\mathsf{leaf}]\ \mathsf{recurse} \\
 & & \qquad \mathsf{desc\text{-}or\text{-}self}::^\star[\tau(\varphi)^{X/(X\vee\tau(\varphi)_{\mathsf{leaf}})} \wedge \neg\mathsf{leaf}] \cup \\
 & & \qquad \mathsf{self}::*[X \vee \tau(\varphi)_{\mathsf{leaf}}]/\mathsf{desc}::* \qquad\qquad \big)[\mathsf{leaf}] \rangle \\
 & & \text{where } \tau(\varphi)_{\mathsf{leaf}} = \tau(\varphi)^{X/\mathsf{false}} \wedge \mathsf{leaf}
\end{array}
\tag{1}
$$

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

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, $\langle \mathsf{child}::*[\varphi] \rangle$ is equivalent to the following node expression (which does not use the child axis):*

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

4 The Undecidability of CXP+IFP and of 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 disllowed.*

The proof is based on a reduction from the halting problem for 2-register machines (cf. [4]). 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 already 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 *initial and final states*, respectively. 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.

The problem whether a given two-register machine M has a successful run (starting in the initial state with both register values 0, and ending in the final state with both register values 0) is undecidable.

A run of M can be represented as a string over the alphabet $Q \cup \{a, b, \$\}$ of the form $q_1\vec{a}_1\vec{b}_1 \ldots q_n\vec{a}_n\vec{b}_n\$$, where each $q_i \in Q$ represents the state of the automaton at the i-th step, and $\vec{a}_i, \vec{b}_i$ are sequences of as respectively bs whose length represents the register content at the i-th step ($ is used to mark the end of the string). We construct an ML+IFP formula which expresses that *for each branch from the current node, the string consisting of the letters of the nodes on the branch encodes an accepting run of the 2-register machine*. It follows that the formula is satisfiable if and only if M has a successful run.

5 Discussion

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 translation we gave from ML+IFP to CXP+IFP crucially uses the latter. We are currently investigating the fragment of CXP+IFP in which variables are only allowed as atomic path expressions.

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.

Acknowledgements

The first author is supported by the Netherlands Organization of Scientific Research (NWO) grant 017.001.190. The second author is supported NWO grant 639.021.508 and by ERC Advanced Grant Webdam on Foundation of Web data management. We thank Anuj Dawar for a useful discussion.

References

[1] L. Afanasiev, T. Grust, M. Marx, J. Rittinger, and J. Teubner. Recursion in XQuery: put your distributivity safety belt on. In M. L. Kersten, et al., eds., *Proc. of 12th Int. Conf. on Extending Database Technology, EDBT 2009 (St. Petersburg, March 2009)*, v. 360 of *ACM Int. Conf. Proc. Series*, pp. 345–356. ACM Press, 2009.

[2] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, v. 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 2001.

[3] 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 ACM SIGMOD Int. Conf. on Management of Data (Chicago, IL, June 2006)*, pp. 479–490. ACM Press, 2006.

[4] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.

[5] A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. *ACM Trans. on Comput. Logic*, 5(2):282–315, 2004.

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