ISSN 2007-9737

# Depth-First Reasoning on Trees

Yensen Limón[1], Everardo Bárcenas[2,1], Edgard Benítez-Guerrero[2], María Auxilio Medina[3]

[1] Universidad Veracruzana, Veracruz, Mexico

[2] CONACYT, Ciudad de Mexico, Mexico

[3] Universidad Politécnica de Puebla, Puebla, Mexico

zs16017367@estudiantes.uv.mx, iebarcenaspa@conacyt.mx, edbenitez@uv.mx, maria.medina@uppuebla.edu.mx

**Abstract.** The $\mu$-calculus is an expressive modal logic with least and greatest fixed-point operators. This formalism encompasses many temporal, program and description logics, and it has been widely applied in a broad range of domains, such as, program verification, knowledge representation and concurrent pervasive systems. In this paper, we propose a satisfiability algorithm for the $\mu$-calculus extended with converse modalities and interpreted on unranked trees. In contrast with known satisfiability algorithms, our proposal is based on a depth-first search. We prove the algorithm to be correct (sound and complete) and optimal. We also describe an implementation. The extension of the $\mu$-calculus with converse modalities allows to efficiently characterize standard reasoning problems (emptiness, containment and equivalence) of XPath queries. We also describe several query reasoning experiments, which shows our proposal to be competitive in practice with known implementations.

**Keywords.** Calculus, automated reasoning, depth-first search, XPath.

## 1 Introduction

The propositional $\mu$-calculus is a modal logic with least and fixed-point operators, expressively corresponding to the monadic second order logic MSO [9]. This logic is known to subsume many temporal, program and description logics such as the Linear Temporal Logic LTL, the Propositional Dynamic Logic PDL, the Computation Tree Logic CTL and $\mathcal{ALC}_{reg}$, which is an expressive description logic with negation and regular roles [4]. Due to its expressive power and nice computational properties, the $\mu$-calculus has been extensively used as a reasoning framework in a wide range of domains, such as program verification, knowledge representation and concurrent pervasive systems [4]. In concurrent pervasive systems, logic-based reasoning frameworks have been successfully tested in context-aware scenarios [2, 3]. In this paper, we propose a reasoning (satisfiability), algorithm for the $\mu$-calculus with converse, where formulas are interpreted over finite unranked tree models. The algorithm is based on a depth-first search, and its complexity is optimal, that is, exponential time with respect to the input formula. We also describe an implementation of the algorithm.

XPath is the standard query language for semi-structured data (XML). This query language also takes an important role in many XML technologies, such as, XProc, XSLT and XQuery. Although the full XPath query language is known to be undecidable [15], the $\mu$-calculus with converse has been successfully used as a reasoning framework for the navigation core of XPath, known as regular path queries [5, 7]. We also describe a logic characterization of regular path queries

Preliminary results were presented in [11, 12]

in terms of $\mu$-calculus formulas. Since the logic is closed under negation and the proposed algorithm is optimal, our implementation can be used for optimal standard reasoning (emptiness, containment and equivalence), of regular path queries.

### 1.1 Related Work

The validity/satisfiability problem of the $\mu$-calculus is in EXPTIME-complete [4]. Furthermore, in [4], several $\mu$-calculus extensions are studied: converse programs (modalities), allows expressing backwards (past) properties; nominals are special formulas to denote individuals; and graded modalities express numerical constraints on node occurrences. The $\mu$-calculus extended with either converse and nominals, converse and graded modalities, or nominals and graded modalities are also complete in EXPTIME. However, the extension consisting of all three features (nominals, converse and graded modalities), known as the fully enriched $\mu$-calculus, leads to undecidability.

These results were obtained by the development of corresponding automata machinery, which was not reported to be implemented. Tableau-based decision algorithms for the $\mu$-calculus with converse and nominals, converse and functional modalities (restricted graded modalities), and nominals and functional modalities are presented in [13].

The complexity of these algorithms is also single exponential time. Implementations of the algorithms are also described. In contrast with this work, where the logic formulas are interpreted over Kripke structures (graphs), in the current paper, we propose a satisfiability algorithm for $\mu$-calculus with converse over finite unranked trees.

A decision algorithm for the monadic second order logic, equally expressive as the $\mu$-calculus, was proposed in [8]. This algorithm, based on automata, was shown useful in practice on the verification of hardware and programs, however its complexity is non-elementary.

In [5], it is also introduced an automata machinery for the $\mu$-calculus with converse over trees. This machinery supports model checking in linear time and decidability in exponential time. Also in this case, implementation is not reported. Another EXPTIME decision algorithm for this logic, the $\mu$-calculus with converse over trees, is presented in [7]. This algorithm is based on a breadth-first search in the style of Fischer-Ladner [6]. In the current paper, we also propose a Fischer-Ladner algorithm, but based in a depth-first search. Experiments show competitive results with respect to the breadth-first search algorithm.

XPath has been widely studied before from the formal perspective [15, 5, 7].

Although it is known that the full language is undecidable [15], several complexity results have been obtained for decidable fragments. The navigation core of XPath, containing all features to allow multi-directional navigation (children, siblings, ancestors, descendants, etc.), is known as regular path queries. Emptiness, containment and equivalence of regular path queries are known to be in EXPTIME [5].

The sole known reasoning solver reported so far for regular path queries can be found in [7]. In this paper, we also describe several reasoning experiments for regular path queries. These experiments show that our implementation is competitive in running time with respect to the other known implementation.

### 1.2 Outline

We first introduce the $\mu$-calculus with converse modalities in Section 2. Then in Section 3, we introduce the notion of Fischer-Ladner trees, which is the syntactic structure constructed by the satisfiability algorithm, which is described in detail in Section 4. Also in this section, it is shown that the algorithm is correct (sound and complete) and optimal (EXPTIME). In Section 5, we describe a linear characterization of regular path queries in terms of $\mu$-calculus formulas. We also report in this section several query reasoning experiments. We conclude in Section 6, with a summary of the paper together with a brief discussion of further research perspectives.

## 2 The $\mu$-Calculus on Trees

In this section, we introduce the $\mu$-calculus with converse. Formulas are interpreted over finite unranked trees. The alphabet is considered by two sets, $PROP$ and $MOD$, where $PROP$ is a set of proposition and $MOD = \{1, 2, 3, 4\}$, is the set of modalities.

**Definition 1** (Syntax)**.** *The set of $\mu$-calculus formulas is defined by the following grammar:*

$$\varphi ::= p \mid X \mid \neg\varphi \mid \varphi \vee \psi \mid \langle m\rangle\varphi \mid \mu X.\varphi$$

*where $p$ is a proposition, $m$ a modality, and $X$ is a variable.*

We assume variables can only occur bounded and in the scope of a modality.

Formulas are interpreted as subset nodes in unranked trees. Propositions are used as labels for nodes, negation ($\neg$), is interpreted as set complement, disjunctions are interpreted as set union. We write $\phi \wedge \varphi$ instead of $\neg(\neg\phi \vee \neg\varphi)$.

Modal formulas $\langle m\rangle\phi$ holds in nodes where there is an $m$-adjacent node supporting $\phi$. The least fixed-point is intuitively interpreted as a recursion operator.

We now present the tree structures defined in style of kripke.

**Definition 2** (Tree structure)**.** *A tree structure, or simply a tree, is a tuple $(N, R^m, L)$ where:*

- *$N$ is a set of nodes;*
- *$R^m$ is a family of binary relations of nodes $(N \times N)$ forming a tree structure; and*
- *$L$ is a function labeling $L : N \mapsto 2^{PROP}$.*

we give a formal description of the semantic of a formula, where $Var$ is a set of fixed-point variables, $2^N$ is the power set of nodes:

$$sust(V, K, X, \varphi) := V'(Y) = V(Y) \text{ where } Y \neq X.$$
$$sust(V, K, X, \varphi) := V'(Y) = [\![\varphi]\!]^K_V \text{ where } Y = X.$$

**Definition 3** (Semantics)**.** *Consider a tree $K$ and a valuation $V : Var \mapsto 2^N$, where $Var$ is a set of variables. The semantics of the formula is defined as follows:*

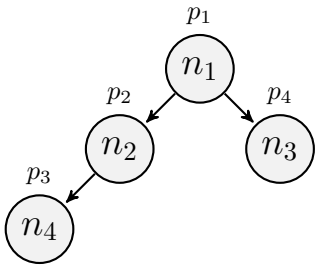


**Fig. 1.** Tree model example where $\phi = \mu X.(p_3 \vee \langle 1\rangle X)$, then $n_1 = \{\phi, \langle 1\rangle p_2, p_1\}$, $n_2 = \{\phi, p_2\}$ , $n_3 = \{p_4\}$ and $n_4 = \{\phi, p_3\}$

$$\begin{array}{rcl}
[\![p]\!]^K_V & = & \{n \mid p \in L(n)\}, \\
[\![\neg\varphi]\!]^K_V & = & N \backslash [\![\varphi]\!]^K_V, \\
[\![\varphi \vee \psi]\!]^K_V & = & [\![\varphi]\!]^K_V \cup [\![\psi]\!]^K_V, \\
[\![\langle m\rangle\varphi]\!]^K_V & = & \{n \mid R(n, m) \cap [\![\varphi]\!]^K_V \neq \emptyset\}, \\
[\![X]\!]^K_V & = & V(X), \\
[\![\mu X.\varphi]\!]^K_V & = & \bigcap\{N' \mid [\![\varphi]\!]^K_{V[^{\mu X.\varphi}/_X]} \subseteq N'\}.
\end{array}$$

A formula $\phi$ is satisfiable, if and only if, there is a tree (model), such as the interpretation of $\phi$ over the tree is not empty, that is $[\![\phi]\!]^K_V \neq \emptyset$.

**Example 1.** *In Figure 1, there is a graphical representation of tree model. Consider for instance the following formula:*

$$\langle 1\rangle p_2 \wedge p_1,$$

*This formula selects nodes names $p_1$ with a child named $p_2$. In Figure 1, this formula holds in $n_1$. Recursive navigation can be expressed with the fixed-point:*

$$\mu X.(p_3 \vee \langle 1\rangle X) \wedge \langle 1\rangle p_2 \wedge p_1.$$

*This formula holds in a $p_1$ node with a descendant $p_3$ at node $n_4$ and a child $p_2$ at node $n_2$. In Figure 1, this formula also holds in $n_1$.*

## 3 Fischer-Ladner Trees

The satisfiability algorithm builds a syntactic version of tree structures, called Fischer-Ladner trees [6], where nodes are sets of subformulas. In this Section, we define this notion of trees.

ISSN 2007-9737



There is a well-know bijection between n-ary and binary trees: one relation is for the first-child (parent), and the other for the following (previous) sibling. In Figure 2, there is a graphical representation of this bijection. Hence, without loss of generality, for technical convenience, we work on binary trees. Also, modal formulas should be re-interpreted:

— $\langle 1\rangle\varphi$ holds in nodes where its first child supports $\varphi$;

— $\langle 2\rangle\varphi$ is interpreted in nodes where $\varphi$ holds in its following (right) sibling;

— $\langle 3\rangle\varphi$ stands where $\varphi$ is the parent; and

— $\langle 4\rangle\varphi$ where $\varphi$ is the previous (left) sibling.

We also consider formulas in negation normal form only: negation only occurs in front of propositions and formulas $\langle m\rangle\top$, where $\top$ stands for $p \vee \neg p$. We then define the following function.

$$\begin{gathered} nnf(X) = \neg X, \\ nnf(p) = \neg p, \\ nnf(\varphi \vee \psi) = nnf(\varphi) \wedge nnf(\psi), \\ nnf(\langle m\rangle\varphi) = \langle m\rangle nnf(\varphi) \vee \neg\langle m\rangle\top, \\ nnf(\mu X.\varphi) = \mu x.nnf(\varphi)\left[{}^{X}/_{\neg X}\right]. \end{gathered}$$

We then define the negation normal form of a formula, as the resulting formula of replacing negations $\neg\varphi$ with $nnf(\varphi)$.

Some notions are required before introducing Fischer-Ladner trees.

**Definition 4.** *We define a binary relation $R^{FL}$ on formulas with $i = 1, 2$ :*

$$\begin{gathered} R^{FL}(\varphi, nnf(\varphi)), \quad R^{FL}(\varphi_1 \wedge \varphi_2, \varphi_i), \\ R^{FL}(\langle m\rangle\varphi, \varphi), \quad R^{FL}(\mu X.\varphi, \varphi[{}^{\mu x.\varphi}/_{X}]), \\ R^{FL}(\varphi_1 \vee \varphi_2, \varphi_i). \end{gathered}$$

**Definition 5** (Fischer-Ladner Closure)**.** *Given a formula $\varphi$, the Fischer-Ladner closure of $\varphi$ is defined as $FL(\varphi) = FL(\varphi)_k$, where $k$ is the smallest positive integer satisfying $FL(\varphi)_k = FL(\varphi)_{k+1}$, where:*

$$FL(\varphi)_0 = \{\varphi\},$$
$$FL(\varphi)_{i+1} = FL(\varphi)_i \cup \{\psi' \mid R^{FL}(\psi, \psi'), \psi \in FL(\varphi)_i\}.$$

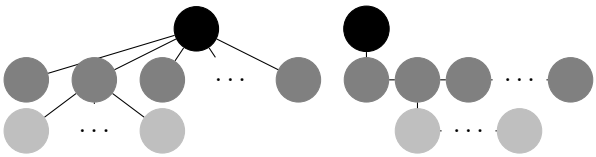

**Fig. 2.** Example the transformation of tree n-ary with the binary tree [1]

We now define the lean set for the syntactic nodes. This set is intuitively composed by the propositions and modal subformulas of the input formula. The propositions are used to name the nodes, and modal subformulas give the topological information of candidate trees.

**Definition 6** (Lean)**.** *Given a formula $\varphi$ and a proposition $p'$ not occurring in $\varphi$, the lean of $\varphi$ is defined as follows for all $m \in MOD$:*

$$lean(\varphi) = \{p, \langle m\rangle\varphi \in FL(\varphi)\} \cup \{p', \langle m\rangle\top\}.$$

**Example 2.** *Consider the following formula:*

$$\varphi = \langle 1\rangle\langle 1\rangle\neg p_1 \wedge \langle 1\rangle\neg p_1 \wedge \neg p_1 \wedge \mu X.(p_1 \vee \langle 1\rangle X) \wedge \langle 2\rangle p_2 \wedge p_3.$$

*The lean of $\varphi$ is defined as follows:*

$$\begin{aligned} lean(\varphi) = \{&p_1, p_2, p_3, \langle 1\rangle\neg p_1, \langle 1\rangle\langle 1\rangle\neg p_1, \langle 1\rangle\mu X.(p_1 \vee \\ &\langle 1\rangle X), p', \langle 2\rangle p_2, \langle 1\rangle\top, \langle 2\rangle\top, \langle 3\rangle\top, \langle 4\rangle\top\}. \end{aligned}$$

We are now ready to define nodes in Fischer-Ladner trees.

**Definition 7** (Nodes)**.** *Given a formula $\varphi$, a node in Fischer-Ladner trees is defined as subset of $lean(\varphi)$, such as:*

— *it contains at least one proposition;*

— *if $\langle m\rangle\psi$ occurs in it, also $\langle m\rangle\top$ does;*

— *both $\langle 3\rangle\top$ and $\langle 4\rangle\top$ do not occur in it.*

**Example 3.** *Consider the same formula and corresponding lean as in Example 2. We then define the following nodes:*

— $n_1 : \{p_3, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X), \langle 1\rangle\neg p_1, \langle 1\rangle\langle 1\rangle\neg p_1, \langle 2\rangle p_2\}$;

— $n_2 : \{p', \langle 1\rangle\neg p_1, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$;

— $n_3 : \{p', \langle 1\rangle \mu X.(p_1 \vee \langle 1\rangle X)\}$;

— $n_4 : \{p_1\}$;

— $n_5 : \{p_2\}$.

And we now finally define Fischer-Ladner trees.

**Definition 8** (Fischer-Ladner Tree)**.** *We inductively define Fischer-Lader trees as follows:*

— *The empty tree $\emptyset$ is a Fischer-Ladner tree; and*

— *$(n, X_1, X_2)$ is a Fischer-Ladner tree, provided that $n$ is a node (called the root), and $X_1$ and $X_2$ are Fischer-Ladner trees.*

when clear from context, we often write simply a tree instead of a Fischer-Ladner tree.

**Example 4.** *Consider the nodes in Example 3. We then define the following tree:*

$$T = (n_1, (n_2, (n_3, (n_4, \emptyset, \emptyset), \emptyset), \emptyset), (n_5, \emptyset, \emptyset)),$$

*Figure 3, ilustrates the tree.*

## 4 Depth-First Search Based Satisfability Algorithm

In this section we describe satisfiability algorithm of $\mu$-calculus based on depth-first search. The Algorithm 1 decides whether a formula is satisfiable or unsatisfiable. In the $main$ function the algorithm creates the required nodes. Then the nodes are iterated. The satisfiability algorithm builds the tree starting from the node that satisfies the formula. This node is used as a candidate and from it begins the construction of the tree. Once the node is selected, the algorithm enters the $search$ function.

The Algorithm 2, shows the $search$ function. When the formula is $p, \top, \neg p$ or $\neg\langle m\rangle\top$, then, the algorithm evaluates the satisfiability of the node and returns $true$. If it contains fixpoint, then, algorithm makes its expansion and called the function $search$. The cases for disjunction and conjunction, the algorithm invokes the $search$ function on both sides. When the formulas have the form of modality, it searches the next node. The formula can contain modalities $\langle 1\rangle, \langle 2\rangle, \langle 3\rangle or \langle 4\rangle$.

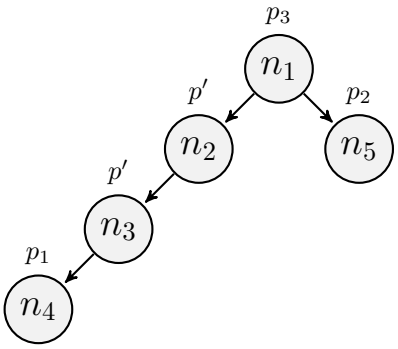


**Fig. 3.** Fischer-Ladner tree model for $\varphi = \langle 1\rangle\langle 1\rangle\neg p_1 \wedge \mu X.(p_1 \vee \langle 1\rangle X) \wedge \langle 1\rangle\neg p_1 \wedge \neg p_1 \wedge \langle 2\rangle p_2 \wedge p_3$, where $n_1 = \{p_3, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X), \langle 1\rangle\neg p_1, \langle 1\rangle\langle 1\rangle\neg p_1, \langle 2\rangle p_2\}$, $n_2 = \{p', \langle 1\rangle\neg p_1, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_3 = \{p', \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_4 = \{p_1\}$ and $n_5 = \{p_2\}$

The current node is deleted from the list of nodes, the algorithm obtains a list of the following nodes with the next function, the node list is iterated and the tree is updated. It is called the search function, if it returns $false$, thus, the next node is removed from the tree. Each time a node is added, it is passed to the status used and it can not be reused in subsequent levels of the tree, avoiding in such a way to build an infinite tree. If there are not more nodes to search, then the algorithm returns $false$. The algorithm ends when all nodes have been traveled or the tree is built.

Now we define when a formula is satisfiable.

**Definition 9.** *Given a formula $\phi$ and a node $n$, we define the satisfiability $n \vdash \phi$ as follows:*

$$\frac{}{n \vdash \top}, \qquad \frac{\varphi \in n}{n \vdash \varphi}, \qquad \frac{n \vdash \varphi}{n \vdash \varphi \vee \psi},$$

$$\frac{n \vdash \psi}{n \vdash \varphi \vee \psi}, \qquad \frac{\varphi \notin n}{n \vdash \neg\varphi}, \qquad \frac{n \vdash \varphi \quad n \vdash \psi}{n \vdash \varphi \wedge \psi},$$

$$\frac{n \vdash \varphi[{}^{\mu x.\varphi}/_{x}]}{n \vdash \mu x.\varphi}.$$

The $\Delta_m$ function is responsible for verifying the union of one node with respect to another node through a modality $m$. Where $n'$ is possible to join node, $n_1$ the current node, $m$ are modalities $\langle 1\rangle$ or $\langle 2\rangle$, $\bar{m}$ are reverse modalities $\langle 3\rangle$ or $\langle 4\rangle$ and $\langle m\rangle\varphi$ modal formula.

**Definition 10.** *Given two nodes $n$, $n'$ and formula $\varphi$, it is determined that these nodes are consistent with respect to the formula $\Delta_m(n, n')$ where $m \in \{1, 2, 3, 4\}$, if and only if modal formulas $\langle m\rangle\varphi_1, \langle \bar{m}\rangle\varphi_2 \in lean(\varphi)$, it states the following:*

— $\forall\langle m\rangle\varphi \in$ *lean* :$\langle m\rangle\varphi \in n \Leftrightarrow n' \vdash \varphi$,

— $\forall\langle \bar{m}\rangle\varphi \in$ *lean* :$\langle \bar{m}\rangle\varphi \in n' \Leftrightarrow n \vdash \varphi$.

**Example 5.** *Given the formula $\varphi = \langle 1\rangle\langle 1\rangle\neg p_1 \wedge \langle 1\rangle\neg p_1 \wedge\neg p_1\wedge \mu X.(p_1 \vee \langle 1\rangle X) \wedge \langle 2\rangle p_2 \wedge p_3$ used in Example 2 the unions are the following:*

— $\Delta_m(n_1, n_2)$,

— $\Delta_m(n_1, n_5)$,

**Algorithm 1** Depth-first search satisfiability algorithm for $\mu$-calculus with converse over trees

```
function MAIN(φ)
    Y ← N^φ
    T ← ∅
    for all n_i of Y do
        if n_i ⊢ φ then
            T ← (n_i, ∅, ∅)
            if search(n_i, n_i, φ) then
                return true
            else
                T ← ∅
            end if
        end if
    end for
    return false
end function
```

— $\Delta_m(n_2, n_3)$

— $\Delta_m(n_3, n_4)$

*where $n_1 = \{p', \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X), \langle 1\rangle\neg p_1, \langle 1\rangle\langle 1\rangle\neg p_1, \langle 2\rangle p_2, p_3\}$, $n_2 = \{p', \langle 1\rangle\neg p_1, \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_3 = \{p', \langle 1\rangle\mu X.(p_1 \vee \langle 1\rangle X)\}$, $n_4 = \{p_1\}$ and $n_5 = \{p_2\}$.*

*These nodes are consistent because the child nodes contain the witnesses of the modal formulas.*

The $next$ function obtains the set of nodes subsequent to a candidate node. This function is defined by a triplet of elements $(n, \langle m\rangle\varphi, Y)$. Where $n$ is the current node, $\langle m\rangle\varphi$ modal formula, $Y$ the set of nodes unused and $T$ is tree.

**Definition 11** (Nodes)**.** *Given a tree $T$, the $nodes$ function is defined as follows:*

— $nodes(\emptyset) = \{\}$,

— $nodes((n, T_1, T_2)) = \{n\} \cup nodes(T_1) \cup nodes(T_2)$,

A node $Tn$ is denoted as follows:

— $T_n$ is a $n \in nodes(T)$.

**Definition 12** (SubTree)**.** *The subtree $T'$ of a given tree $T$ ($T' \subseteq T$) is defined as follows:*

— $T \subseteq T$,

— $T \subseteq (n, T_1, T_2)$ *if* $T \subseteq T_1$ *or* $T \subseteq T_2$,

**Definition 13** (Root)**.** *The $root(T)$ function takes the tree as input and returns a node root of the tree.*

— $root(\emptyset) = \emptyset$,

— $root((n, T_1, T_2)) = n$.

**Definition 14** (neighborhood)**.** *Given a tree $T$ and node $n$, the neighborhood function obtains adjacent nodes. Considering that exist trees $T', T_1$ and $T_2$:*

— $parent(n, T) = n'$ *such that,* $(n', T_n, T') \subseteq T$,

— $child(n, T) = n'$ *such that,* $(n, (n', T_1, T_2), T') \subseteq T$,

— $ps(n, T) = n'$ *such that,* $(n', T', T_n) \subseteq T$,

— $fs(n, T) = n'$ *such that,* $(n, T', (n', T_1, T_2)) \subseteq T$,

— $neighborhood(n, T) = \{parent(n, T), child(n, T), ps(n, T), fs(n, T)\}$.

**Definition 15** (Delete)**.** *Given a tree $T$ and node $n$, the node is removed of tree, the $Delete$ function is defined as follows:*

— $delete(\emptyset, n) = \emptyset$,

— $delete((n, T_1, T_2), n) = (\emptyset, T_1, T_2)$,

**Algorithm 2** Search function of the satisfiability algorithm

```
function SEARCH(n_0, n_1, φ)
    if φ = p then
        if p ∈ n_1 then
            return true
        end if
    end if
    if φ = ¬p then
        if p ∉ n_1 then
            return true
        end if
    end if
    if φ = ¬⟨m⟩⊤ then
        if ⟨m⟩⊤ ∉ n_1 then
            return true
        end if
    end if
    if φ = ⊤ then
        return true
    end if
    if φ = φ_1 ∧ φ_2 then
        if search(n_0, n_1, φ_1) and
            search(n_0, n_1, φ_2) then
            return true
        end if
    end if
    if φ = φ_1 ∨ φ_2 then
        if search(n_0, n_1, φ_1) or
            search(n_0, n_1, φ_2) then
            return true
        end if
    end if
    if φ = ⟨m⟩φ then
        Y ← Y\{n_1}
        X ← next(n_1, ⟨m⟩φ, Y ∪ {n_0}, T)
        for all n_j of X do
            T ← update(T, n_1, n_j, m)
            if search(n_1, n_j, φ) then
                return true
            else
                T ← delete(T, n_j)
            end if
        end for
    end if
    if φ = μx.φ then
        return search(n_0, n_1, φ[^{μx.φ}/_x])
    end if
    return false
end function
```

— $delete((n', T_1, T_2), n)$= $(n', delete(T_1, n), delete(T_2, n))$,

*where* $n \neq n'$.

**Definition 16** (Update)**.** *Given a tree* $T$*, two different nodes* $n$ *and* $n'$ *and a modality* $m$*, the* $Update$ *function is defined as follows:*
*When* $n' \notin nodes(T)$,
– $update((n, T_1, T_2), n, n', 3)$ = $(n', (n, T_1, T_2), \emptyset)$,
– $update((n, T_1, T_2), n, n', 4)$ = $(n', \emptyset, (n, T_1, T_2))$,
– $update((n, \emptyset, T_2), n, n', 1)$ = $(n, (n', \emptyset, \emptyset), T_2)$,
– $update((n, T_1, \emptyset), n, n', 2)$ = $(n, T_1, (n', \emptyset, \emptyset))$,
– $update((n'', T_1, T_2), n, n', m)$ = $(n'', update(T_1, n, n', m), update(T_2, n, n', m))$,
– $update(\emptyset, n, n', m)$= $\emptyset$.
*When* $n' \in nodes(T)$,
– $update(T, n, n', m)$= $T$,
*where* $m$ *is modality,* $T$ *is Tree and* $n$ *is a node.*

**Definition 17** (Next)**.** *Given two nodes* $n, n'$ *and a formula. We say that, these nodes are consistent modally with regard to formula, it is denoted by* $\Delta_m(n, n')$*, if and only if, for all formulas* $\langle m\rangle\phi$*,* $\langle \bar{m}\rangle\phi$ *and* $T$ *is tree, the* $next$ *function is defined as follows:*

— $next(n, \langle m\rangle\varphi, Y, T) = \{n' \in Y \cup neighborhood(n, T) \mid \Delta_m(n, n')\}$,

*where* $m = 1, 2, 3, 4$.

**Theorem 1** (Soundness)**.** *If the satisfiability algorithm returns* $true$ *for the input formula* $\phi$*, then there is tree model satisfying* $\phi$*:*

*Proof.* By induction on the formula $\phi$. If the algorithm returns $true$, we know that there is a tree $T = (n, T_1, T_2)$. We will now construct a tree model $K(T) = (N, R, L)$.

— $N = \{n | n$ is a node in $T\}$;

— We now define the edges of $K$. For every triple $(n, T_1, T_2)$ of $T$, we define $R(n, 1) = n_1$ and $R(n, 2) = n_2$, thus $R(n_1, 3) = n$ and $R(n_2, 4) = n$, provided that $n_1$ and $n_2$ are the respective roots of $T_1$ and $T_2$

— if a proposition $p \in n$ and a node $n \in N$, then $L(n) = p$.

We now show that $K(T)$ satisfies $\phi$ by induction.

When the formula $\phi$ has the form: $p, \top, \neg p$ or $\neg\langle m\rangle\top$, then, the algorithm stops and returns $true$. Hence, there exists a node $n$ to consider. The tree model is $T = (n, \emptyset, \emptyset)$ and $T \vdash \phi$. From Definition 9 of $n \vdash \phi$, we know that $\phi$ is satisfiable. The tree structure is $K = (\{n\}, \emptyset, L(n))$. Thus, in the case $p$ then $p \in L(n)$. Now in the case $\neg p$ where $p \notin L(n)$. For the case $\top$ is valid in the node without restriction. Finally, in the case $\neg\langle m\rangle\top$ is valid when $\langle m\rangle\top \notin n$. Therefore $\phi$ is satisfied by $K(T)$.

The cases for disjunction and conjunction are also easy. Thus, if it is a disjunction $\varphi_1 \vee \varphi_2$, then where at least one $\varphi_1$ or $\varphi_2$ is satisfied by $n \vdash \varphi_1$ or $n \vdash \varphi_2$. The structure tree is $T = (n, T_1, T_2)$ and $T \vdash \varphi_1 \vee \varphi_2$. Now by induction $K(T) \models \varphi_1$ or $K(T) \models \varphi_2$, thus $K(T) \models \varphi_1 \vee \varphi_2$. Therefore $\varphi_1 \vee \varphi_2$ is satisfied by $K(T)$.

In the case of a conjunction $\varphi_1 \wedge \varphi_2$, both $\varphi_1$ and $\varphi_2$ are satisfied by $n \vdash \varphi_1$ and $n \vdash \varphi_2$ hence $T \vdash \varphi_1 \wedge \varphi_2$, then by induction $K(T) \models \varphi_1$ and $K(T) \models \varphi_2$. Therefore $\varphi_1 \wedge \varphi_2$ is satisfied by $K(T)$.

The case where $\phi$ is a modal formula $\langle m\rangle\varphi$. As we know that $n' \vdash \varphi$, thus there is a subtree $T'$ of $T$, then we apply induction in $T' \vdash \varphi$ and $K(T') \models \varphi$, where we define $K_1 = (N', R', L')$, hence $K_0 = (N, R, L)$ where $N = N' \cup \{n\}$, $R = R' \cup \{(n, m, n'), (n, \bar{m}, n')\}$ and $L = L' \cup \{(n, p) | n \vdash p\}$. Therefore $\langle m\rangle\varphi$ is satisfied by $K(T)$.

The case for $\phi$ is a fix-point formula $\mu X.\varphi$. Let us recall that there exists an expansion equivalent in fixed-point formula: $[\![\mu X.\varphi]\!]_V^K = [\![\varphi[^{\mu X.\varphi}/_X]]\!]_V^K$ shown in the article [14].We know that there is a tree $T \vdash \varphi[^{\mu X.\varphi}/_X]$. Recall variables can only occur in the scope of a modality. Therefore, we remember that this form only exists where the formula is $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$ or $\langle m\rangle\psi$ but they were shown above.

□

**Theorem 2** (Completeness)**.** *If a formula $\phi$ is satisfiable, then the algorithm returns $true$.*

*Proof.* By assumption, we know that given a tree structure $K$ that satisfies a formula $\phi$. The algorithm builds a tree that satisfies $\phi$ and return $true$. This proof is divided in two steps. First we build a tree of $\phi$ with $K$ structure and we show how a tree $T$ satisfies $\phi$. Second step we show how a tree can be built where $\phi$ is satisfied.

We will now construct a K structure $T(K) = (n, T_1, T_2)$.

— for every node $n$ in $K$, there is a corresponding node $n'$ in $T(K)$, such that for every $\varphi$ in $lean(\phi)$, if $n \in [\![\varphi]\!]_\emptyset^K$, then $\varphi \in n'$; and

— for every node $n_0$ in $K$, if $R^K(n_0, 1) = n_1$ , $R^K(n_0, 2) = n_2$, $R^K(n_1, 3) = n_0$ and $R^K(n_2, 4) = n_0$, then $(n'_0, T_1, T_2)$ is a subtree of $T(K)$, where $root(T_i) = n'_i (i = 1, 2)$ and $n'_1$, $n'_2$ and $n'_0$ are the corresponding nodes, defined in the previous step, of $n_1$, $n_2$ and $n_0$. When $R^K(n_0, i)(i \in \{1, 2\})$ is not defined, then $T_i = \emptyset$.

We now show that $T(K)$ satisfies the formula $\phi$.

We consider a tree structure satisfying the formula $\phi$. By induction on a formula $\phi$. When the formula $\phi$ has the form: $p, \top, \neg p$ or $\neg\langle m\rangle\top$, these are the base cases. The tree structure satisfying $\phi$ is a $T = (n, \emptyset, \emptyset)$ where $n$ is a candidate node. In the case of $p$ is in the $lean$, the $T$ in $T(K)$ satisfies $p$. In the case when $\phi$ is $\top$, then any $T$ is fine. Now in the cases where $\phi$ is $\neg p$ and $\neg\langle m\rangle\phi$, none of them belong to the $lean$. Thus $\phi$ is $\neg p$ and $\neg\langle m\rangle\phi$ must satisfy the tree $T$, thus $T(K) \models \phi$.

The cases for disjunction and conjunction are also easy. Thus, if it is a disjunction $\varphi_1 \vee \varphi_2$, now by induction $T(K) \models \varphi_1$ or $T(K) \models \varphi_2$, thus $T(K) \models \varphi_1 \vee \varphi_2$. otherwise , if it is a conjunction $\varphi_1 \wedge \varphi_2$, now by induction $T(K) \models \varphi_1$ and $T(K) \models \varphi_2$, thus $T(K) \models \varphi_1 \wedge \varphi_2$.

The case where $\phi$ is a modal formula $\langle m\rangle\varphi$. Then we know that $\langle m\rangle\varphi$ is in the $lean$, according to $T(K)$, then $\varphi$ corresponding the corresponding node $n'$ in $T(K)$ of $n$ contains $\langle m\rangle\varphi$, hence $T(K) \models \langle m\rangle\varphi$.

In the case when $\phi$ is a fix-point formula $\mu X.\varphi$. Let us remember that there is an equivalent expansion in fix-point formula: $[\![\mu X.\varphi]\!]_V^K = [\![\varphi[^{\mu X.\varphi}/_X]]\!]_V^K$. We recall that $X$ variables can

only occur in the scope of a modality. Therefore, we remember that only exist this form where the formula is $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$ or $\langle m \rangle \psi$ and they were shown above. The variables in fix-point occur only in the modalities, thus they never occur in this case.

We consider the tree structure $K$ satisfying a formula $\phi$, and we follow by induction on $\phi$. The base case is easy, since $K$ is $(n, \emptyset, \emptyset)$. In the first case all nodes are created. Then $\phi$ enter to the $search$ function and is evaluated in the base cases. When the formula $\phi$ has the form: $p, \top, \neg p$ or $\neg\langle m \rangle \top$ it is immediate. In the cases of disjunction and conjunction the search function is called with each subformula. The case for $\phi$ is a fix-point formula $\mu X.\varphi$, it calls the $search$ function with equivalent expansion in fix-point formula. When $\phi$ has the form $\langle m \rangle \varphi$, the formula is evaluated in $\phi = \langle m \rangle \varphi$ and called the $next$ function, where the following nodes are searched. After that, the tree $K$ is updated to the next node calling the $update$ function. We know that the $update$ function generates the relationship of two nodes and updates the tree. Then by induction, we know the tree $T(K)$ has been produced by the algorithm in $search$ function. Therefore $T(K)$ is built by the algorithm. □

**Theorem 3** (Complexity)**.** *The satisfiability algorithm is in EXPTIME.*

*Proof.* Notice that, the lean has linear size $K$ with respect to the formula $\phi$ size. Notice the $N^{\phi}$ size is exponential with respect to the lean size. The next step is to initialize the tree. Hence, the first loop is clearly at most exponential. The next step is $n \vdash \phi$, then the cost is linear. Assigning node to the tree ($T \leftarrow (n_i, \emptyset, \emptyset)$) is also a step. Now the search function has distinct cases. In the case where $\phi$ is $p$ is a step, the next step $p \in n$ the cost is linear. Now in the case where $\phi$ is $\neg p$ is a step, the next step is $\neg p \notin n$, then the cost is linear. In the case where $\phi$ is $\neg\langle m \rangle \top$ is also a step, the next step is $\langle m \rangle \top \notin n$, then the cost is linear. Finally when the case where $\phi$ is $\top$ is a step. In the case where the formula is $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$, thus $search$ function is k-steps until the formula expands and ends in a base case.

The formula is expanded when it obtains a subformula of main formula. When the formula is $\langle m \rangle \phi$ is a step. Besides, when it removes the node from the list ($Y \leftarrow Y \backslash \{n_1\}$), the cost is at most exponential time because it iterates all the set of nodes. The $next$ function has an exponential time cost. This function obtains nodes that can join and the nodes are used in the next cycle. The $update$ function perform a assignation of tree, the assignment is a step. But the search for $Update$ function in the tree is at most exponential. The next step is to expand the formula and get the modal formula. Then call the $search$ function.

Now if the search function return false, the tree is modified with the $Delete$ function. The $Delete$ function reassigns the tree, this assignment is a step but the search in the $Delete$ function in the tree is at most exponential. Now in case where the formula is $\mu X.\phi$, then $\mu X.\phi$ is exponential and limits the number of nodes [10]. However if the $search$ function returns $false$, the algorithm generates possible branches. Hence, the branch created has backtracking when the $search$ function returns $false$. Consequently, the algorithm can be generating false branches. Hence the number of branches is: $(1/2)n(n+1)$. Hence, the tree size is at most exponential. □

## 5 Regular Path Queries

In this section, we introduce the XPath queries. XPath language is used to navigate XML documents.

**Definition 18** (Syntax)**.** *The set of queries is defined as follows.*

$$
\begin{array}{lcl}
\varrho & := & \top \mid \alpha \mid p \mid \alpha : p \mid \varrho/\varrho \mid \varrho[\beta] \\
\beta & := & \beta \vee \beta \mid \neg\beta \mid \rho \\
\rho & := & \varrho \mid /\rho \mid \rho \cap \rho \mid \rho \cup \rho \mid \rho \backslash \rho \\
\alpha & := & self \mid child \mid parent \mid descendant \mid \\
 & & desc{-}or{-}self \mid desc{-}or{-}self \mid \\
 & & ancestor \mid anc{-}or{-}self \mid \\
 & & foll{-}sibling \mid prec{-}sibling \mid \\
 & & following \mid preceding
\end{array}
$$

**Definition 19** (XPath semantics)**.** *The semantics of XPath queries is defined by function $[\![\cdot]\!]^{\cdot}$ with respect to a structure $K$, to pairs of nodes in $K$.*

ISSN 2007-9737



$$
\begin{array}{rcl}
[\![\top]\!]^K & = & N \times N \\
[\![p]\!]^K & = & \{(n,n) \mid p \in L(n)\} \\
[\![\alpha]\!]^K & = & \{(n_1,n_2) \mid n_1 \xrightarrow{\alpha} n_2\} \\
[\![\alpha : p]\!]^K & = & \{(n_1,n_2) \in [\![\alpha]\!]^K \mid p \in L(n_2)\} \\
[\![\varrho_1/\varrho_2]\!]^K & = & [\![\varrho_1]\!]^K \circ [\![\varrho_2]\!]^K \\
[\![\varrho[\beta]]\!]^K & = & \{(n_1,n_2) \in [\![\varrho]\!]^K \mid n_2 \in [\![\beta]\!]^K\} \\
[\![\varrho]\!]^K & = & \{n_2 \mid (n_1,n_2) \in [\![\varrho]\!]^K\} \\
[\![\neg\beta]\!]^K & = & N \backslash [\![\beta]\!]^K \\
[\![\beta_1 \vee \beta_2]\!]^K & = & [\![\beta_1]\!]^K \cup [\![\beta_2]\!]^K \\
[\![/\varrho]\!]^K & = & \{(r,n) \in [\![\varrho]\!]^K \mid r \textit{ is the root }\} \\
[\![\rho_1 \cap \rho_2]\!]^K & = & [\![\rho_1]\!]^K \cap [\![\rho_2]\!]^K \\
[\![\rho_1 \cup \rho_2]\!]^K & = & [\![\rho_1]\!]^K \cup [\![\rho_2]\!]^K \\
[\![\rho_1 \backslash \rho_2]\!]^K & = & [\![\rho_1]\!]^K \backslash [\![\rho_2]\!]^K
\end{array}
$$

*where the function $[\![\cdot]\!]^{\cdot}$ is used for the qualifiers or filters and it is distinguished from the interpretation of the path $[\![\cdot]\!]^{\cdot}$.*

**Example 6.** *Let us consider the following example:*

$$/descendant : q$$

*where we select the $descendant$ node with the proposition $q$. The path is evaluated from the root and find the node with proposition $q$. Another example, let us consider the following path:*

$$descendant : q/parent : p$$

*The path select the $descendant$ $q$ and this contains a $parent$ $p$.*

*Figure 4 shows the query for two previous examples. Then, the query $/descendant : q$ select the $n_3$ and the query $descendant : q/parent : p$ select the $n_2$.*

**Definition 20** (Reasoning problems)**.** *We define the containment, emptiness and equivalence in the context of regular path queries.*

- *A query $\rho$ is empty, if and only if, for every tree $K$, its interpretation is empty, that is, $[\![\rho]\!]^K = \emptyset$;*
- *A query $\rho_1$ is contained in a query $\rho_2$, if and only if, for every tree $K$, each pair of nodes in the interpretation of $\rho_1$ is in the intepretation of $\rho_2$, this is, $[\![\rho_1]\!]^K \subseteq [\![\rho_2]\!]^K$; and*

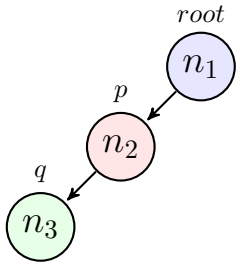


**Fig. 4.** Model of the queries:$/descendant : q$ and $descendant : q/parent : p$

- *Two queries $\rho_1$ and $\rho_2$ are equivalent, if and only if, for every tree $K$, $\rho_1$ is contained in $\rho_2$ and vice versa, that is, $[\![\rho_1]\!]^K \subseteq [\![\rho_2]\!]^K$ and $[\![\rho_2]\!]^K \subseteq [\![\rho_1]\!]^K$.*

Regular queries can be expressed as $\mu$-calculus formulas. Now we show the characterization of the formulas Xpath to terms of the $\mu$-calculus inspired by [7].

**Definition 21** (XPath queries into $\mu$-calculus formulas)**.** *Given a context formula $C$, the translation $F$ from regular path queries into $\mu$-calculus is defined as follows:*

$$
\begin{array}{rcl}
F(self, C) & = & C \\
F(child, C) & = & \mu Z.\langle 3\rangle C \vee \langle 4\rangle Z \\
F(foll{-}sibling, C) & = & \mu Z.\langle 4\rangle C \vee \langle 4\rangle Z \\
F(prec{-}sibling, C) & = & \mu Z.\langle 2\rangle C \vee \langle 2\rangle Z \\
F(parent, C) & = & \langle 1\rangle \mu Z.C \vee \langle 2\rangle Z \\
F(descendant, C) & = & \mu Z.\langle 3\rangle (C \vee Z) \vee \langle 4\rangle Z \\
F(desc{-}or{-}self, C) & = & \mu Z.C \vee \mu Y.\langle 3\rangle (Y \vee Z) \\
 & & \vee \langle 4\rangle Y \\
F(ancestor, C) & = & \langle 1\rangle \mu Z.C \vee \langle 1\rangle Z \vee \langle 2\rangle Z \\
F(anc{-}or{-}self, C) & = & \mu Z.C \vee \langle 1\rangle \mu Y.Z \vee \langle 2\rangle Y \\
F(following, C) & = & F(desc{-}or{-}self, n_1) \\
F(preceding, C) & = & F(desc{-}or{-}self, n_2) \\
n_1 & = & F(foll{-}sibling, n_3) \\
n_2 & = & F(prec{-}sibling, n_3) \\
n_3 & = & F(anc{-}or{-}self, C)
\end{array}
$$

$$
\begin{aligned}
F(\alpha : p, C) &= F(\alpha, C) \wedge p \\
F(\varrho_1/\varrho_2, C) &= F(\varrho_2, F(\varrho_1, C)) \\
F(\varrho[\beta], C) &= F(\varrho, C) \wedge F^{\leftarrow}(\beta, C) \\
F(/\varrho, C) &= F(\rho, C \wedge (\neg\langle 3\rangle\top \wedge \\
&\quad \neg\langle 4\rangle\top)) \\
F(\rho_1 \cap \rho_2, C) &= F(\rho_1, C) \wedge F(\rho_2, C) \\
F(\rho_1 \cup \rho_2, C) &= F(\rho_1, C) \vee F(\rho_2, C) \\
F(\rho_1 \backslash \rho_2, C) &= F(\rho_1, C) \wedge \neg F(\rho_2, C) \\
F^{\leftarrow}(\neg\beta, C) &= \neg F^{\leftarrow}(\beta, C) \\
F^{\leftarrow}(\beta_1 \vee \beta_2, C) &= F^{\leftarrow}(\beta_1, C) \vee F^{\leftarrow}(\beta_2, C) \\
F^{\leftarrow}(\varrho_1/\varrho_2, C) &= F^{\leftarrow}(\varrho_1, F^{\leftarrow}(\varrho_2, C)) \\
F^{\leftarrow}(\varrho[\beta], C) &= F^{\leftarrow}(\varrho, C \wedge F^{\leftarrow}(\beta, \top)) \\
F^{\leftarrow}(\alpha : p, C) &= F^{\leftarrow}(\alpha, C) \wedge p \\
F^{\leftarrow}(self, C) &= C \\
F^{\leftarrow}(child, C) &= \langle 1\rangle \mu Z.C \vee \langle 2\rangle Z \\
F^{\leftarrow}(foll{-}sibling, C) &= \mu Z.\langle 2\rangle C \vee \langle 2\rangle Z \\
F^{\leftarrow}(prec{-}sibling, C) &= \mu Z.\langle 4\rangle C \vee \langle 4\rangle Z \\
F^{\leftarrow}(parent, C) &= \mu Z.\langle 3\rangle C \vee \langle 4\rangle Z \\
F^{\leftarrow}(descendant, C) &= \langle 1\rangle \mu Z.C \vee \langle 1\rangle Z \vee \langle 2\rangle Z \\
F^{\leftarrow}(desc{-}or{-}self, C) &= \mu Z.C \vee \langle 1\rangle \mu Y.Z \vee \langle 2\rangle Y \\
F^{\leftarrow}(ancestor, C) &= \mu Z.\langle 3\rangle (C \vee Z) \vee \langle 4\rangle Z \\
F^{\leftarrow}(anc{-}or{-}self, C) &= \mu Z.C \vee \mu Y.\langle 3\rangle (Y \vee Z) \\
&\quad \vee \langle 4\rangle Y \\
F^{\leftarrow}(following, C) &= F^{\leftarrow}(desc{-}or{-}self, n_1^{\leftarrow}) \\
F^{\leftarrow}(preceding, C) &= F^{\leftarrow}(desc{-}or{-}self, n_2^{\leftarrow}) \\
n_1^{\leftarrow} &= F^{\leftarrow}(foll{-}sibling, n_3^{\leftarrow}) \\
n_2^{\leftarrow} &= F^{\leftarrow}(prec{-}sibling, n_3^{\leftarrow}) \\
n_3^{\leftarrow} &= F^{\leftarrow}(anc{-}or{-}self, C)
\end{aligned}
$$

**Example 7.** *We consider the following path showed in Example 6:*

$$/descendant : q$$

*then, the query is translated as follows:*

$$\mu Z.(\langle 3\rangle (C \wedge (\neg\langle 3\rangle\top \wedge \neg\langle 4\rangle\top) \vee Z) \vee \langle 4\rangle Z) \wedge q$$

*Another example is the following path:*

$$/descendant : q/parent : p$$

*then, the query is translated as follows:*

$$F_1 = \mu Z.(\langle 3\rangle ((C \wedge (\neg\langle 3\rangle\top \wedge \neg\langle 4\rangle\top) \vee Z) \vee \langle 4\rangle Z) \wedge q$$

$$F_2 = \langle 1\rangle \mu Y.C \vee \langle 2\rangle Y \wedge p$$

*Now the formula $F_1$ is passed as context $C$ to formula $F_2$.*

$$\langle 1\rangle \mu Y.(\mu Z.(\langle 3\rangle (C \wedge (\neg\langle 3\rangle\top \wedge \neg\langle 4\rangle\top) \vee Z) \vee \langle 4\rangle Z) \wedge q) \vee \langle 2\rangle Y \wedge p$$

Now we show reasoning problems with XPath queries. The problems are solved using the logic as a reasoning framework, and they are emptiness, containment and equivalence of queries.

**Theorem 4.** *(Query reasoning [7]). For any XPath queries $\rho, \rho_1, \rho_2$, tree $K$ and valuation $V$, the following holds:*

- $[\![\rho]\!]^K = \emptyset$ *if and only if* $[\![F(\rho, \top)]\!]_V^K = \emptyset$*;*
- $[\![\rho_1]\!]^K \subseteq [\![\rho_2]\!]^K$ *if and only if* $[\![F(\rho_1, \top) \wedge \neg F(\rho_2, \top)]\!]_V^K = \emptyset$*; and*
- $F(\rho, \top)$ *has linear size with respect to* $\rho$ *and* $\neg F(\rho_1, \top) \wedge F(\rho_2, \top)$ *has linear size with respecto to* $\rho_1$ *and* $\rho_2$*.*

We now show a set of queries to be evaluated in reasoning problems.

XPath Decision Problem

| | |
|---|---|
| $e_1$ | /self::a[ child::b [child::c /child::d] /child::b [ descendant::d/descendant::d ] /child:: b [child::c/child::d] ] |
| $e_2$ | /self::a[ child::b [child::c /child::d] /child::b [ descendant::d/child::e] /child:: b [descendant::c/child::d] ] |
| $e_3$ | child::a/descendant::b/child::d [preceding-sibling::c]/child::e |
| $e_4$ | child::a/descendant::b/ descendant::c / following-sibling::d / descendant::e |
| $e_5$ | descendant::a/descendant::b/following::d /descendant::e |
| $e_6$ | descendant::a/descendant::b[descendant::c] /following::d/descendant::e ∩ descendant::a/ descendant::d[preceding::c]/descendant::e |

XPath Decision Problem [7]

| | |
|---|---|
| $q_1$ | /a[.//b[c/*//d]/b[c//d]/b[c/d]] |
| $q_2$ | /a[.//b[c/*//d]/b[c/d]] |
| $q_3$ | a/b//c/foll-sibling::d/e |
| $q_4$ | a/b//d[prec-sibling::c]/e |
| $q_5$ | a/c/following::d/e |
| $q_6$ | a/b[//c]/following::d=e ∩ a/d[preceding::c]/e |

These experiments were performed to determine the containment of a query. In [7] an algorithm based on breadth-first search in the style of Fischer-Ladner is presented. They present results

**Table 1.** Result of experiments.

| XPath Decision Problem | Time (ms) | Lean size |
|---|---|---|
| $e_1 \subseteq e_2$ | 107.0 | 46 |
| $e_2 \nsubseteq e_1$ | 106.0 | 46 |
| $e_3 \subseteq e_4$ | 52.0 | 28 |
| $e_6 \subseteq e_5$ | 90.0 | 27 |
| $e_5 \nsubseteq e_6$ | 74.0 | 27 |
| $q_1 \subseteq q_2$ and $q_2 \nsubseteq q_1$ | 353.0 | 46 |
| $q_3 \subseteq q_4$ | 45.0 | 34 |
| $q_6 \subseteq q_5$ and $q_5 \nsubseteq q_6$ | 41.0 | 45 |
| $e_1$ | 66 | 27 |
| $e_2$ | 70 | 28 |
| $q_1$ | 55.0 | 36 |
| $q_2$ | 111.0 | 30 |

of reasoning problems such as the containment, emptiness and equivalence of a query. We now show experiments similar to those presented in article [7]. They are similar with respect to lean size. Where lean is the delimiter of the complexity of the algorithm. We used Xpath queries in non-abbreviated syntax. In Table 1, we observe the results obtained by the algorithm based on a depth-first search. The results presented by these authors are very similar to ours. Time is given in milliseconds. In each experiments the context $C$ is $\top$.

This algorithm was implemented in Java language. The experiments were executed on the computer with the following features: Windows 8 operating system, AMD processor A6 2.7GHz., 8Gb of RAM.

This algorithm is found online at the following url[1].

## 6 Conclusion and Future Work

In the current paper, we proposed a satisfiability algorithm for the $\mu$-calculus for trees with converse modalities. In contrast with known satisfiability algorithms, our proposal is based on a depth-first search algorithm. The algorithm is showed correct and optimal. Practical experiments in the setting of XPath reasoning are also described. Experiments show competitive results with respect to the other known implementation [7].

We believe the $\mu$-calculus with converse modalities can be used as a reasoning framework in context-aware systems [2, 3]. We are also interested in the development and efficient implementation of satisfiability algorithms for decidable extensions of the $\mu$-calculus, such as counting [1].

[1] $https://148.226.81.4:8181/AlgoritmoWeb/$

## References

1. **Bárcenas, E. & Lavalle, J. (2014).** Global numerical constraints on trees. *Logical Methods in Computer Science*, Vol. 10, No. 2.
2. **Benítez-Guerrero, E. (2010).** Context-aware mobile information systems: Data management issues and opportunities. **Arabnia, H. R., Hashemi, R. R., Vert, G., Chennamaneni, A., & Solo, A. M. G.**, editors, *International Conference on Information & Knowledge Engineering, IKE*, pp. 127–133.
3. **Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D. (2010).** A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, Vol. 6, No. 2, pp. 161–180.
4. **Bonatti, P. A., Lutz, C., Murano, A., & Vardi, M. Y. (2008).** The complexity of enriched mu-calculi. *Logical Methods in Computer Science*, Vol. 4, No. 3.
5. **Calvanese, D., De Giacomo, G., Lenzerini, M., & Vardi, M. Y. (2010).** Node selection query languages for trees. **Fox, M. & Poole, D.**, editors, *AAAI*, AAAI Press.
6. **Fischer, M. J. & Ladner, R. E. (1977).** Propositional modal logic of programs (extended abstract). **Hopcroft, J. E., Friedman, E. P., & Harrison, M. A.**, editors, *ACM Symposium on Theory of Computing*, ACM, pp. 286–294.
7. **Genevès, P., Layaïda, N., Schmitt, A., & Gesbert, N. (2015).** Efficiently deciding $\mu$-calculus with converse over finite trees. *ACM Trans. Comput. Log.*, Vol. 16, No. 2, pp. 16.
8. **Henriksen, J. G., Jensen, J. L., Jørgensen, M. E., Klarlund, N., Paige, R., Rauhe, T., & Sandholm, A. (1995).** Mona: Monadic second-order logic in practice. **Brinksma, E., Cleaveland, R., Larsen, K. G., Margaria, T., & Steffen, B.**, editors, *TACAS*, volume 1019 of *Lecture Notes in Computer Science*, Springer, pp. 89–110.

9. **Janin, D. & Walukiewicz, I. (1996).** On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. **Montanari, U. & Sassone, V.**, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, Springer, pp. 263–277.
10. **Lenzi, G. (2005).** The modal $\mu$-calculus: a survey. *TASK Quarterly*, Vol. 9, No. 3, pp. 293–316.
11. **Limón, Y., Bárcenas, E., Benítez-Guerrero, E., & Medina, M. A. (2017).** Depth-first search satisfiability of the $\mu$-calculus with converse over trees. *2017 International Conference on Electronics, Communications and Computers, CONIELECOMP*, IEEE.
12. **Limón, Y., Bárcenas, E., Benítez-Guerrero, E., & Mezura-Godoy, C. (2015).** Towards a reasoning model for context-aware systems: Modal logic and the tree model property. *Research in Computing Science*, Vol. 99, pp. 9–18.
13. **Tanabe, Y., Takahashi, K., & Hagiya, M. (2008).** A decision procedure for alternation-free modal $\mu$-calculi. **Areces, C. & Goldblatt, R.**, editors, *Advances in Modal Logic*, College Publications, pp. 341–362.
14. **Tarski, A. (1955).** A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, Vol. 5, No. 2, pp. 285–309.
15. **ten Cate, B. & Marx, M. (2009).** Axiomatizing the logical core of XPath 2.0. *Theory Comput. Syst.*, Vol. 44, No. 4, pp. 561–589.

*Article received on 03/08/2016; accepted on 12/10/2016.*
*Corresponding author is Yensen Limón.*