TREE LOGIC WITH RECURSION AND MODEL CHECKING ALGORITHM

Taolue Chen Tingting Han Jian Lu
State Key Laboratory of Novel Software Technology, Nanjing University,
Nanjing, Jiangsu, P.R.China 210093
email: ctl@ics.nju.edu.cn

ABSTRACT
Semi-structured data plays an increasingly important role in the exchange of information between globally distributed applications, which invokes renewed interests in typed programming languages that can manipulate tree-like data structures. Tree logic, inherited from ambient logic, is introduced as the formal foundation of related programming language and type systems. In this paper, we introduce recursion into such logic system, which can describe the tree data more clearly and concisely. By making a distinction between proposition and predicate, a concise semantics interpretation for our modal logic is given. We also develop a model checking algorithm for the logic without $\triangleright$ operator. The correctness of the algorithm is shown. Such work can be seen as the basis of the semi-structured data processing language and more flexible type system.

KEY WORDS
Semi-structured data, Tree logic, Fixpoint, Model Checking Algorithm

1 Introduction

Semi-structured data plays an important role in the exchange of information between globally distributed applications: examples include BibTex files and XML documents. Due to the growing popularity of semi-structured data, and particularly XML, there are renewed interests in typed programming languages that can manipulate tree-like data structures.

In general, we are going to have some tree-like data *t*, and some description language *T* that can flexibly describe the shape of the data. What we are interested in is the description languages which are so flexible that they are akin to logics rather than to type systems. We refer the reader to [2] for more descriptions. Generally speaking, the key problem is to find rich description languages and satisfaction and validity algorithms admitted by them. In the research community, it is well recognized that modal logic is an excellent candidate of such description language and thus in essence, such problems can be reduced to corresponding model checking problem, which is the main focus of this paper.

These problems have been widely studied by some researchers. For data model, the research community mostly agree on defining semi-structured data using trees with 'graphical' links or labelled directed graphs. For the description language, a logic that can be used as a rich description language for tree-like data has been provided. It merges as an application of the novel area of spatial logics used for describing data and network structures. In this paper, we call this logic **Tree Logic**. Many researches have focused on such a modal logic system. Indeed, tree logic is a sublogic of Ambient Logic [4] for ambient calculus due to Cardelli and Gordon, or spatial logic [1] due to Caires et al. Some detailed comparison is deferred to Section 4.

With the semi-structured data models and associated languages being investigated, the need for manipulating private data elements is becoming aware. Such private resources can be modelled using names and name hiding notions arising from the π-calculus [7] : during data manipulation, the identity of a private name is not important as long as the distinctions between it and other (public or private) names are preserved. Such work has been initialized by Cardelli et al in [5], where the simple tree model (such as XML) is extended in a general and orthogonal way with a hiding operator. Besides that, in logic, some modal operators, inspired by spatial logics of concurrency devised to cope with π-calculus restriction and scope extrusion, are introduced. However, so far there still lacks a satisfactory approach to introduce recursion into such logics, due to subtle interactions between recursion and first-order quantification. The recursion is important and useful, it can describe the tree data more clearly and concisely. We will give a illuminating example in Section 2. The standard approach to introducing recursion into a modal logic is via fixpoint, as in μ-calculus, however such work is not trivial since the rich modalities, such as ®, $\oslash$, especially the first order quantification И is introduced in order to manipulate hidden labels. To deal with such problems, we make a distinction between proposition and predicate, thus the possible interactions between recursion and first-order quantification can be solved based on the above work, a concise semantics interpretation for our modal logic is given. The main contribution of this paper lies in the model checking algorithm for the logic. We devote to presenting such an algorithm because it is the pivot of semi-data related language and corresponding type system. The correctness of the algorithm is shown. Note that due to space restriction, most of proofs in this paper are omitted, we refer the interested readers to our technical report [3].

The rest of the paper is organized as follows: In Section 2, the data model and tree logic with recursion is introduced, and the semantics is presented, some useful properties are also discussed in this section. The model checking algorithm is presented and its correctness is shown in Section 3. The paper is concluded with Section 4 where related work is also discussed.

2 Data Model and Tree Logic

2.1 Data Model

Let $l, m, n, ...$ ranged over by $\mathcal{N}$, which is a countable infinite set of names. The data model, which essentially is an edge-labelled finite tree with restriction name, is defined by BNF as follows:

$$P, Q ::= 0 \mid P|Q \mid m[P] \mid (\nu n)P$$

we refer reader to [5] for the intended meaning of these operators. As in common process calculi, $(\nu n)P$ introduces the distinction of bound names and free names. In common, we use $fn(P)$ and $bn(P)$ to denote the set of free names and bound names respectively appearing in tree *P*. And we identify α-equivalent trees, i.e. trees that are different only in renaming of bound names.

As usually, the structural congruence, denoted by $\equiv$, is defined as usual. We refer the readers to [5] or [3] for details.

The following result is well-known for ambient calculus and can be easily adapted to our data model.

Lemma 1 *The following properties hold:*

(i) $(\nu n)P \equiv 0$ *iff* $P \equiv 0$.

(ii) For different name m, n, $(\nu n)P \equiv m[Q]$ *iff there exists tree* R, *s.t.* $P \equiv m[R]$ *and* $Q \equiv (\nu n)R$.

(iii) $(\nu n)P \equiv Q_1 \mid Q_2$ *iff there exists tree* R_1, R_2, *s.t.* $Q_1 \equiv (\nu n)R_1$ *and* $Q_2 \equiv R_2$ *and* $n \notin fn(Q_2)$ *or* $Q_1 \equiv R_1$ *and* $Q_2 \equiv (\nu n)R_2$ *and* $n \notin fn(Q_1)$.

A substitution $\{m_1/n_1, \cdots m_l/n_l\}$ is a function from $\mathcal{N}$ to $\mathcal{N}$ that maps n_i onto m_i for $i \in \{1, \cdots, l\}$ and n onto itself for $n \notin \{n_1, \cdots, n_l\}$. Substitutions are usually denoted by σ. The empty substitution, that is the identity function on $\mathcal{N}$, is written as []. The result of applying σ to P is denoted by $P\sigma$. In the below, by α-conversion it is assumed that a substitution σ acts as an identity on the bound names of the process and keeps the separation between bound and free names. We follow this convention in the below and will use it implicitly in the proof. If $\mathcal{T}$ is a set of trees and σ a substitution, $\mathcal{T}\sigma$ is defined as $\{P\sigma \mid P \in \mathcal{T}\}$.

Substitution that just interchange a pair of names, which is called **transposition** and ranged by τ, will plays a special role in technical developments to follow. More precisely, the transposition of *n* and *m*, written as $\{m \leftrightarrow n\}$, denoted the substitution $\sigma : \{m, n\} \to \{n, m\}$. It turns out that transpositions are a useful tool in proving properties concerning fresh names.

2.2 Tree Logic with Recursion

We assume a countable infinite set $\mathcal{V}$ of *name variables* which is ranged over by $x, y, z, \cdots$, such that $\mathcal{V} \cap \mathcal{N} = \emptyset$. And we assume a countably infinite set $\mathcal{X}$ of *predicate variables*, ranged over by $X, Y, Z, \cdots$. The syntax of the formula is defined by BNF as follows:

$$\begin{array}{rcl} A, B & ::= & T \mid \neg A \mid A \vee B \mid 0 \mid A|B \mid A \triangleright B \mid \eta[A] \mid \\ & & A@\eta \mid \eta® A \mid A \oslash \eta \mid И x.A \mid \forall x.A \mid \Lambda(\tilde{\eta}) \\ \Lambda & ::= & X \mid \lambda\tilde{x}.A \mid \nu X.\Lambda \end{array}$$

where, $\eta \in \mathcal{N} \cup \mathcal{V}$.

In formulas of the form $\forall x.A$, $И x.A$, $\lambda\tilde{x}.A$ and $\nu X.\Lambda$, the distinguished occurrences of x and X are binding, with the scope of propositions A or predicate Λ. We define on formulas the relation $\equiv_\alpha$ of α-congruence in the standard way, that is, as the least congruence identifying formulas modulo renaming of bound (name and predicate) variables. We will consider formulas always modulo α-congruence. Note that for a formula, the notion of name substitution is extended to the function from $\mathcal{N} \cup \mathcal{V}$ to $\mathcal{N}$, i.e. we allow the name variables to be replaced by names.

For any formula A, we introduce the following sets in the common way, that is, the names in A, denoted by $n(A)$, the free name variables in A, denoted by $fv(A)$, and the free predicate variables in A, denoted $fpv(A)$. Since their definitions are rather standard, and we omit the formal presentation.

Note that for convenience, we identify β-equivalence formulas, that is, $(\lambda\tilde{x}.A)(\tilde{\eta})$ and $A[\tilde{\eta}/\tilde{x}]$. A formula A is called name-closed if $fv(A) = \emptyset$ and is called predicated-closed if $fpv(A) = \emptyset$. A formula is closed if it has neither free name variables nor free predicate variables.

In the tree logic, besides the unary operator $\neq$, the operator $\triangleright$ may also convey the same 'negative' effect. Formally, for any formula A, we define $\neg -$ and $- \triangleright A$ as two negative operators. We say that a predicate variable X is positive (resp. negative) in A if it is under an even (resp. odd) number of negative operators. Note that a variable X can be both positive and negative in a formula A. We say that a formula A is monotonic in X whenever every occurrence of X in A is positive, otherwise we say A is anti-monotonic in X.

A fixpoint predicate $\nu X.\Lambda$ is *well-formed* if Λ is well-formed and $n(\Lambda) \cup fv(\Lambda) = \emptyset$ and monotonic in X. Note that we require that Λ has no free name, thus $n(\Lambda(\tilde{\eta}))$ and $fv(\Lambda(\tilde{\eta}))$ are totally determined by the actual parameter $\tilde{\eta}$. Also, all free occurrences of X in Λ must occur just at positive position, which is used to ensure monotonicity of the denotation mapping associated with fixpoint formulas. A formula is well-formed if every fixpoint subformula in it is well-formed. In the sequel, we only consider well-formed formulas.

For application, especially some interesting examples of our logic, we refer the reader to [3].

2.3 Semantics

The semantics of formula is defined by assigning to each formula A a set of trees $[\![A]\!]$, namely the set of all trees that satisfy the property denoted by A. Since A may contain free name variables and free occurrences of predicate variables, its denotation depends on the denotation of such variables, which is given by a valuation (name valuation and predicate valuation). A name valuation ρ is a mapping from $\mathcal{V} \cup \mathcal{N}$ to $\mathcal{N}$ which is identity on $\mathcal{N}$. We define $\rho[n/x]$ as $\rho[n/x](y) =$ *if* $x = y$ *then* n *else* $\rho(y)$. A predicate valuation ξ assigns to every predicate variable of arity k a function $\mathcal{N}^k \to \wp(\mathcal{P})$, that is $\xi : \mathcal{X} \to (\mathcal{N}^k \to \wp(\mathcal{P}))$. As usual, the relation $\subseteq$ can be extended point-wise to functional space as follows: for two function $f^{(k)}, g^{(k)} : \mathcal{N}^k \to \wp(\mathcal{P})$, define $f^{(k)} \sqsubseteq g^{(k)}$ *iff* $f(\tilde{n}) \sqsubseteq g(\tilde{n})$ for any $\tilde{n} \in \mathcal{N}^k$. Thus, the functional space $\mathcal{N}^k \to \wp(P)$ forms a complete lattice w.r.t. $\sqsubseteq$. The denotation of formulas is defined inductively in Figure 1.

The semantics defined in Figure 1 is presented in the style of denotation, indeed, it can also be presented by satisfaction relation. We write $P \models_{\rho,\xi} A$ whenever $P \in [\![A]\!]_{\rho,\xi}$: this means that P satisfies formula A under name valuation ρ and predicate valuation ξ. Note that for a name-closed formula A, $[\![A]\!]_{\rho,\xi}$ does not depend on ρ and can be denoted by $[\![A]\!]_{\rho}$; and if A is closed, then $[\![A]\!]_{\rho,\xi}$ depends on neither ρ nor ξ and can be denoted by $[\![A]\!]$.

In the below, we devote to showing that the denotation map is well-defined. In particular, we show the semantics of the fixpoint operation is the intended one, i.e. $\nu X.\Lambda$ indeed denotes the greatest fixpoint.

As in the case of first-order logic, the following lemma which relates substitutions and valuations is common, and will be used implicitly.

Lemma 2 *The following properties hold:*

(i) $[\![A[n/x]]\!]_{\rho,\xi}$=$[\![A]\!]_{\rho[n/x],\xi}$.

(ii) $[\![\Lambda[F/X]]\!]_{\rho,\xi} = [\![\Lambda]\!]_{\rho,\xi[\xi(F)/X]}$.

Since we consider formulas up to α-congruence, we start by verifying that the denotation map is well-defined on the corresponding equivalence classes.

Lemma 3 *For any name evaluation ρ and predicate evaluation ξ, if $A \equiv_\alpha B$, then $[\![A]\!]_{\rho,\xi}$=$[\![B]\!]_{\rho,\xi}$.*

The following lemma shows the monotonicity of denotation semantics.

Lemma 4 *Let $F, G : \mathcal{N}^k \to \wp(\mathcal{P})$ and $F \sqsubseteq G$, then the following properties hold:*

(i) If A and Λ are monotonic in X, then

- $[\![A]\!]_{\rho,\xi[F/X]} \subseteq [\![A]\!]_{\rho,\xi[G/X]}$.
- $[\![\Lambda]\!]_{\rho,\xi[F/X]} \sqsubseteq [\![\Lambda]\!]_{\rho,\xi[G/X]}$

(ii) If A and Λ are anti-monotonic in X, then

- $[\![A]\!]_{\rho,\xi[G/X]} \subseteq [\![A]\!]_{\rho,\xi[F/X]}$.
- $[\![\Lambda]\!]_{\rho,\xi[G/X]} \sqsubseteq [\![\Lambda]\!]_{\rho,\xi[F/X]}$

By the above lemma, it is easy to see that the functional $\lambda\Psi.[\![\Lambda]\!]_{\rho,\xi[\Psi/X]}$ is a monotonic operator over the complete lattice $\mathcal{N}^k \to \wp(\mathcal{P})$ w.r.t. $\sqsubseteq$, since Λ is monotonic in X. By Tarski-Knaster theorem, we have:

Lemma 5 *Let Λ be monotonic in X, and for any name evaluation ρ and predicate evaluation ξ, then*

$$[\![\nu X.\Lambda]\!]_{\rho,\xi} = \mathit{gfix}(\lambda\Psi.[\![\Lambda]\!]_{\rho,\xi[\Psi/X]})$$

Where $\mathit{gfix}(\lambda\Psi.[\![\Lambda]\!]_{\rho,\xi[\Psi/X]})$ denotes the greatest fixpoint of the functional $\lambda\Psi.[\![\Lambda]\!]_{\rho,\xi[\Psi/X]}$.

For spatial logic, the properties concerning fresh names are important, especially when the modal operators which are used to deal with restriction, such as η®A, $A\oslash\eta$, are introduced. Now, we devote to establishing some important results. Following [1], we use transposition as a useful tool to give some concise proof of properties concerning fresh names. The following definition extends the notion of transposition to predicate.

Definition 1 *Let τ be a transposition. A function $f : \mathcal{N} \to \wp(\mathcal{P})$ is τ-preserving if $(f(n))\tau = f(n\tau)$ for any n. A valuation ξ is τ-preserving if $\xi(X)$ is τ-preserving for any X.*

Lemma 6 *Given a transposition τ and a function $f : \mathcal{N} \to \wp(\mathcal{P})$, define $f^\tau : \mathcal{N} \to \wp(\mathcal{P})$ as $f^\tau(n) = f(n) \cup (f(n\tau))\tau$ for any n, then the following properties hold:*

(i) f^τ is τ-preserving.

(ii) If $f \sqsubseteq g$ and g is τ-preserving, then $f^\tau \sqsubseteq g$.

The intuition of the following lemma is obvious. Although the proof is rather long, it needs no new techniques, only case analysis and mutual induction on the structure of A and Λ. Due to space restriction, we omit the details.

Lemma 7 *Suppose ξ is τ-preserving, then the following properties hold:*

(i) $([\![A]\!]_{\rho;\xi})\tau = [\![A\tau]\!]_{\rho;\xi}$.

(ii) $[\![\Lambda]\!]_{\rho;\xi}$ *is τ-preserving.*

Obviously, freshness plays a central role in our logic system and maybe is the most subtle operator, especially for the fresh name quantification. A fundamental consequence of above lemma is the following characterization of fresh name quantification. As in [1], the semantics definition of it is stated in 'existential' style, indeed, it also can be stated in 'universal' style, that is, if some property holds of a fresh name, it holds of all fresh names, which makes clear the universal/existential ambivalence of freshness.

$$
\begin{array}{rcl}
[\![T]\!]_{\rho;\xi} & = & \mathcal{P} \\
[\![\neg A]\!]_{\rho;\xi} & = & \mathcal{P}\backslash[\![A]\!]_{\rho;\xi} \\
[\![A \vee B]\!]_{\rho;\xi} & = & [\![A]\!]_{\rho;\xi} \cup [\![B]\!]_{\rho;\xi} \\
[\![0]\!]_{\rho;\xi} & = & \{P|P \equiv 0\} \\
[\![A|B]\!]_{\rho;\xi} & = & \{P|P \equiv P_1|P_2 \wedge P_1 \in [\![A]\!]_{\rho;\xi} \wedge P_2 \in [\![B]\!]_{\rho;\xi}\} \\
[\![A \triangleright B]\!]_{\rho;\xi} & = & \{P|Q \in [\![A]\!]_{\rho;\xi} \Rightarrow Q|P \in [\![B]\!]_{\rho;\xi}\} \\
[\![\eta[A]]\!]_{\rho;\xi} & = & \{P|\exists Q.P \equiv \rho(\eta)[Q] \wedge Q \in [\![A]\!]_{\rho;\xi}\} \\
[\![A@\eta]\!]_{\rho;\xi} & = & \{P|\rho(\eta)[P] \in [\![A]\!]_{\rho;\xi}\} \\
[\![\eta®A]\!]_{\rho;\xi} & = & \{P|\exists Q.P \equiv (\nu\rho(\eta))Q \wedge Q \in [\![A]\!]_{\rho;\xi}\} \\
[\![A \oslash \eta]\!]_{\rho;\xi} & = & \{P|(\nu\rho(\eta))P \in [\![A]\!]_{\rho;\xi}\} \\
[\![И x.A]\!]_{\rho;\xi} & = & \cup_{n \notin fn(A)}\{P \mid P \in [\![A]\!]_{\rho[n/x];\xi} \wedge n \notin fn(P)\} \\
[\![\forall x.A]\!]_{\rho;\xi} & = & \cap_{n \in \mathcal{N}}\{[\![A]\!]_{\rho[n/x];\xi}\} \\
[\![\Lambda(\tilde{\eta})]\!]_{\rho;\xi} & = & [\![\Lambda]\!]_{\rho;\xi}(\rho(\tilde{\eta})) \\
[\![X]\!]_{\rho;\xi} & = & \xi(X) \\
[\![\lambda\tilde{x}.A]\!]_{\rho;\xi} & = & \lambda\tilde{z}.[\![A]\!]_{\rho[\tilde{z}/\tilde{x}];\xi} \\
[\![\nu X.\Lambda]\!]_{\rho;\xi} & = & \sqcup\{F : \mathcal{N}^k \rightarrow \wp(\mathcal{P})|F \sqsubseteq [\![\Lambda]\!]_{\rho;\xi[F/X]}\}
\end{array}
$$

Figure 1. Interpretation of Formula

Lemma 8 *The following statements are equivalent:*

(i) $P \in [\![И x.A]\!]_{\rho,\xi}$.

(ii) There exists a name $n \notin fn(P) \cup fn(A)$, *s.t.* $P \in [\![A]\!]_{\rho[n/x],\xi}$.

(iii) For every name $n \notin fn(P) \cup fn(A)$, $P \in [\![A]\!]_{\rho[n/x],\xi}$.

3 Model Checking Algorithm

In this section, we devote to providing a model checking algorithm for the logic presented in this paper. Note that we have investigated the problem of model checking tree against formulas that may contain composition adjunct ($\triangleright$). It is now a rather standard result ([6]) that such a problem is undecidable, which might result from the coexist of the existential quantification ($\exists$) and the composition adjunct ($\triangleright$). A novel result of ours lies in that we prove that even the logic contains **only** fresh name quantification (but no existential quantification!) and the composition adjunct, the model checking problem for logic formulas is undecidable all the same. Due to space restriction, the proof is not presented here, and we refer the interested reader to [3]. Under such circumstance, we have to turn to design the model checking algorithm for formula **without** $\triangleright$.

3.1 Algorithm

Since our logic system subsumes the recursion (via fixpoint) constructor, one of the notable features of such algorithm is the mechanism used to keep track of unfolding fixpoint formulae. We adopt the latter of the methods, due to Winskel [8], and generalize it to the predicate case. In our algorithm, the tag sets will contain pairs $(\tilde{n}, P)$ of name vector and the tree. Formally, let $T = \{(\tilde{n_1}, P_1), \ldots, (\tilde{n_l}, P_l)\}$, where, $\tilde{n}_i$ $(1 \leq i \leq l)$ are vectors of the same length, say k and for $\forall i, j, i \neq j$, we have $\tilde{n}_i \neq \tilde{n}_j$. For any tag set T, we use λT to denote a function $\mathcal{N}^k \rightarrow \wp(\mathcal{P})$ defined as follows:

$$(\lambda T)(\tilde{n}) = \begin{cases} \{P\} & \text{if } (\tilde{n}, P) \in T \\ \emptyset & \text{if o.w.} \end{cases}$$

Now, the fixpoint predicate $\nu X.\Lambda$ can be generalized to $\nu X.[T]\Lambda$, note that the X must have the same arity as T and the usage of T lies in recording which points of the model have been visited before thus is only a bookkeeping device. The definition of $n(\nu X.[T]\Lambda)$, $fv(\nu X.[T]\Lambda)$ and $fpv(\nu X.[T]\Lambda)$ are the same as the corresponding definition for $\nu X.\Lambda$.

The denotation of $\nu X.[T]\Lambda$ is a simple extension for $[\![\nu X.\Lambda]\!]_{\rho;\xi}$ as follows:

$$[\![\nu X.[T]\Lambda]\!]_{\rho;\xi} = \sqcup\{F : \mathcal{N}^k \rightarrow \wp(\mathcal{P}) \mid F \sqsubseteq ([\![\Lambda]\!]_{\rho;\xi[F/X]} \sqcup \lambda T)\}$$

It is easy to see that the functional $\lambda\Psi.([\![\Lambda]\!]_{\rho,\xi[\Psi/X]} \sqcup \lambda T)$ is also a monotonic operator over the complete lattice $\mathcal{N}^k \rightarrow \wp(\mathcal{P})$ w.r.t. $\sqsubseteq$, since Λ is monotonic in X. Thus, a little generalization for Lemma 5 is valid all the same, and we use gfix$(\lambda\Psi.[\![\Lambda]\!]_{\rho,\xi[\Psi/X]} \sqcup \lambda T)$ to denote the greatest fixpoint of the functional $\lambda\Psi.[\![\Lambda]\!]_{\rho,\xi[\Psi/X]} \sqcup \lambda T$.

There now follows a technical Lemma which is a generalization of the so called Reduction Lemma of [8], the essence of the tag set method.

Lemma 9 *Let $L = \mathcal{N}^k \rightarrow \wp(\mathcal{P})$ be a complete lattice w.r.t. $\sqsubseteq$ and $\phi : L \rightarrow L$ be a monotonic functional. Then for any $f \in L$,*

$$f \sqsubseteq \textit{gfix}(\lambda\Psi.\phi(\Psi)) \quad \textit{iff} \quad f \sqsubseteq \phi(\textit{gfix}(\lambda\Psi.\phi(\Psi) \sqcup f))$$

So, using Lemma 9, the following lemma can be easily proved.

Lemma 10 *If $(\tilde{n}, P) \notin T$, then*

$$P \in [\![\nu X.[T]\Lambda]\!]_{\rho;\xi}(\tilde{n}) \textit{ iff } P \in [\![\Lambda[\nu X.[T \cup \{(\tilde{n}, P)\}]\Lambda/X]]\!]_{\rho;\xi}(\tilde{n})$$

To deal with name restriction, as in [6], we fix the representation of the tree: using α-renaming of restricted names and the rules (Str ResPar) and (Str ResAmb) of the congruence relation, we group together all name-restriction operators by transforming every tree to one of the form $(\nu n_1) \ldots (\nu n_k)P$ and separate bounded names by the following function *sep*. Note that all bounded names are renamed apart so that they are different.

Definition 2

$$\begin{cases} sep(0) \overset{def}{=} \langle \emptyset, P \rangle & \textit{if } P \equiv 0 \\ sep((\nu n)P) \overset{def}{=} \langle N \cup \{n\}, P' \rangle & \textit{if } sep(P) = \langle N, P' \rangle \\ sep(n[P]) \overset{def}{=} \langle N, n[P'] \rangle & \textit{if } sep(P) = \langle N, P' \rangle \\ sep(P|Q) \overset{def}{=} \langle N \cup N', P'|Q' \rangle & \textit{if } sep(P) = \langle N, P' \rangle \\ & \textit{and } sep(Q) = \langle N', Q' \rangle \end{cases}$$

Now, we are ready to present our model-checking algorithm. It is an extension of the algorithms from [6]. It is well known from the result of [6], for any tree P, the sets $\{P \mid P \equiv 0\}$, $\{(Q, R) \mid P \equiv Q|R\}$ and $\{(n, Q) \mid P \equiv n[Q]\}$ are decidable. For notation, we use $\dot{\cup}$ for disjoint union, that is, $A = B\dot{\cup}C$ if $A = B \cup C \wedge B \cap C = \emptyset$. We recalled that all bound names in the trees are renamed apart so that they are all different from each other and different from all free names occurring in the trees and the formulas. Since $\mathcal{N}$ is countable, we can assume it is ordered. For a set of names V, function new(V) returns the least name in $\mathcal{N} \backslash V$. The algorithm is presented in Figure 2.

Now, we devote to proving the correctness of our algorithm. To establish the termination property of the algorithm, we need to bound on the number of names for model checking process. First, recall that since we adopt the α-equivalence for formula, we can assume that both bound names in P and the bound name variables in a formula A are different. Then we write N_P for the number of names (including free and bound names) contained in the tree P and N_A for the number of names and name variables contained in A. Note that names in tag set of the formula are not included, since it only contributes as a bookkeeping. The following lemma is important, by which we can conclude that provided that each term only appears once in each tag set (just as in our algorithm), the size of tag set is bounded since the tree P we consider is finite.

Lemma 11 *For each recursive call of check, with caller parameter (N, P, A) and the callee parameter (N',P',A'), $N_{P'} + N_{A'} \leq N_P + N_A$.*

We now use this fact to give a well-founded ordering to formulae. We write $A \ll_{\mathcal{P}} A'$ iff A' is not a fixpoint formula and A is a proper sub-formula of A', otherwise A is the form $\Lambda[\nu X.[T \cup \{\tilde{n}, P\}]\Lambda/X](\tilde{n})$ and A' is $\nu X.[T]\Lambda(\tilde{n})$ where $(\tilde{n}, P) \notin T$ and T contains only nodes from $\mathcal{P}$. We aim to show that the transitive closure $\ll_{\mathcal{P}}^{+}$ of this relation is a well-founded order whenever $\mathcal{P}$ is finite.

Lemma 12 *For any tree P, $\ll_{\mathcal{P}}^{+}$ is well-founded order.*

Lemma 13 *Let ρ be name evaluation and ξ be predicate evaluation for $\forall x.A$, and assume $n \notin fn(P) \cup n(A)$, then $P \in [\![A]\!]_{\rho,\xi}$ iff $P \in \cap_{k \in fn(P) \cup n(A) \cup \{n\}} [\![A[k/x]]\!]_{\rho,\xi}$.*

Theorem 1 *For any tree P and closed $\triangleright$-free formula A, the following properties hold:*

(i) check$(sep(P), A)$ terminates;

(ii) check$(sep(P), A) = true$ iff $P \in [\![A]\!]$.

4 Conclusion

In this section, we conclude our work and discuss the related work. This paper deals with semi-structured data model and related logic system, i.e. tree logic system. We extend existing work such as [5] with recursion, which is important as we have pointed out in Section 2. Because of the subtle interactions between recursion and first-order quantification, especially the 'fresh' quantification И, such task is challenging and in which one of our contribution lies. We solve such a problem by making a distinction between proposition and predicate. A concise semantics interpretation for the modal logic formula is given. Based on it, since as we point out in the Introduction, model-checking algorithm plays a curial role in the research of corresponding programming language and type system, we focus on devising such an algorithm. Unfortunately, it can be shown that model checking the full logic system is not decidable. Alternatively, we present a model checking algorithm for $\triangleright$-free sublogic system. We adapt the well-known Winskel's tag set method to predicate case to deal with fixpoint operator, note that our tag set construction is different from Winskel's. The correctness of the algorithm is shown.

There are some publications on tree logic, or more generally, ambient logic or spatial logic. The ambient logic has been developed step by step for a few years. And the the most comprehensive version might be [4]. A spatial logic for an asynchronous π-calculus was introduced and studied in [1], which has both fresh name quantification and recursion. The tree logic can be seen as the adaption of above work to the research of semi-structured data processing language and related type systems. Its development

$$
\begin{aligned}
&\text{check}(N,P,T) \overset{def}{=} true;\\
&\text{check}(N,P,\neg A) \overset{def}{=} \neg\text{check}(N,P,A);\\
&\text{check}(N,P,A\vee B) \overset{def}{=} \text{check}(N,P,A)\vee\text{check}(N,P,B)\\
&\text{check}(N,P,0) \overset{def}{=} \begin{cases} \textit{true} & \text{if } P\equiv 0\\ \textit{false} & \text{o.w.}\end{cases}\\
&\text{check}(N,P,A|B) \overset{def}{=} \bigvee_{N=N_1\dot{\cup}N_2}\bigvee_{P\equiv P_1|P_2}\text{check}(N_1,P_1,A)\wedge\text{check}(N_2,P_2,A)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\wedge fn(P_1)\cap N_2=\emptyset\wedge fn(P_2)\cap N_1=\emptyset;\\
&\text{check}(N,P,n[A]) \overset{def}{=} n\notin N\wedge P\equiv n[Q]\vee check(N,Q,A);\\
&\text{check}(N,P,A@n) \overset{def}{=} \text{check}(N,n[P],A)\\
&\text{check}(N,P,n®A) \overset{def}{=} \bigvee_{m\in N}\text{check}(N\backslash\{m\},P[n/m],A)\\
&\qquad\qquad\qquad\qquad\vee(n\notin fn(P)\wedge\text{check}(N,P,A));\\
&\text{check}(N,P,A\oslash n) \overset{def}{=} \text{check}(N\cup\{n\},P,A);\\
&\text{check}(N,P,\text{И}x.A) \overset{def}{=} \text{check}(N,P,A[\text{new}(fn(N,P)\cup fn(A))/x]);\\
&\text{check}(N,P,\forall x.A) \overset{def}{=} \bigwedge_{n\in fn(N,P)\cup fn(A)}\text{check}(N,P,A[n/x])\\
&\qquad\qquad\qquad\qquad\qquad\qquad\wedge\text{check}(N,P,A[\text{new}(fn(N,P)\cup fn(A))/x]);\\
&\text{check}(N,P,(\nu X.[T]\Lambda)(\tilde{n})) = \begin{cases} \textit{true} & if(\tilde{n},P)\in T\\ \text{check}(N,P,\Lambda[\nu X.[T\cup\{(\tilde{n},P)\}]\Lambda/X](\tilde{n})) & \text{o.w.}\end{cases}
\end{aligned}
$$

Figure 2. The Algorithm

follows similar lines. [2] has a good introduction. However, our work follows [5], in which hidden information is studied. However, in [5], the transposition is explicitly in the data model and logic system while we follow the more traditional approach and transposition is only a proof tool. Now, we will do some comments on [1] since both of [1] and our work deal with recursion. Comparing to [1], besides the difference in the data (process) model, the syntax and the semantics are also very different. [1] does not make a distinction between proposition and predicate, however, it conveys difficulties when interpreting the fresh name quantification. As a remedy, the notion of *PSets* is introduced. We refer the reader to [1] for more details. The advantage of our solution lies in that the semantics of our logic is clearer and more concise. What's more, it is more favorable (at least) for model checking purpose. However, some useful tools, such as transposition, comes from [1]. It is worth pointing out that we believe our method can also be applied to ambient calculus and spatial logic (with recursive), we leave it as our future work.

There are several directions for further research. First of all, how to improve efficiency of our algorithm is an interesting problem. At the same time, the tree logic lacks so called somewhere modality $\diamondsuit$, we think it is important for the description of the static structure of the tree, which is another direction of our further research.

Acknowledgements The work in this paper is partially support by NNSFC (60273034, 60233010), 863 Program (2002AA116010) and 973 Program of China (2002CB312002).

References

[1] L.Caires, L.Cardelli. A Spatical Logic for Concurrency (Part I). Proc. TACS'2001, LNCS 2215, pp.1-30, Springer, 2001.

[2] C.Calcagno, L.Cardelli, A.D.Goron. Deciding Validity in a Spatial Logic for Trees. Proc. TLDI'03, ACM Press, pp. 62-73, 2003.

[3] T.Chen, T.Han, J.Lu. Tree Logic with Recursion and Model Checking Algorithm. Technical Report of State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, P.R.China, 2004.

[4] L.Cardelli, A.D.Gordon. Ambient Logic. Mathematical Structures in Computer Science. To appear.

[5] L.Cardelli, P.Gardner, G.Ghelli. Manipulating Trees with Hidden Labels. Proc. FOSSACS'03, LNCS 2620, Springer, 2003.

[6] W.Charatonik, J.-M.Talbot. The Decidability of Model Checking Mobile Ambient. Proc. CSL'01. LNCS 2142, pp.339-354, Springer, 2001.

[7] R.Milner, J.Parrow, D.Walker. A Calculus of Mobile Process, part I/II. Journal of Information and Computation, 100:1-77, Sept.1992.

[8] G.Winskel. A Note on Model Checking the Modal μ-calculus. Theoretical Computer Science 83:157-167, 1991.