CTL Model Checking for Labelled Tree Queries *

Sylvain Hallé, Roger Villemaire, Omar Cherkaoui
Université du Québec à Montréal
C. P. 8888, Succ. Centre-Ville
Montréal, Canada H3C 3P8
halle.sylvain@info.uqam.ca

Abstract

Querying and efficiently validating properties on labelled tree structures has become an important part of research in numerous domains. In this paper, we show how a fragment of XPath called Configuration Logic (CL) can be embedded into Computation Tree Logic. This framework embeds into CTL a larger subset of XPath than previous work and in particular allows universally and existentially quantified variables in formulas. Finally, we show how the variable binding mechanism of CL can be seen as a branching-time equivalent of the "freeze" quantifier.

1 Introduction and Related Work

The representation of data in the form of tree-like structures is a natural choice for numerous applications. Arborescent, or so-called *semi-structured* data has been used in areas as diverse as mobile ambients [10], database systems [1] and network equipment configuration [19]. The widespread use of the Extensible Markup Language (XML) [7] has further confirmed the importance of labelled trees as a mean of representing information, forming among other things the basis for web services standards.

Complementary to the representation of data into structures is the need to query those structures, either to validate constraints on them, to express patterns or to extract parts respecting some selection criterion. To this end, various languages have been proposed with different levels of expressivity and fields of application, such as the Tree Query Logic (TQL) [9], XML Schema [17], Schematron/Schemapath [14], XPath [12], XQuery [5], and Configuration Logic [24].

In [2, 3], a method for validating formulas on tree structures by means of model checking has been presented. More precisely, the authors show how the formulas of the logical fragment of Core XPath [18], called Simple XPath, can be translated into Computation Tree Logic (CTL) [13]. By this framework, the problem of checking a Simple XPath formula on a given XML tree amounts to checking a CTL formula on a suitably constructed Kripke structure whose states and transitions reflect the nodes and links of the original tree, a result already hinted back in [18, 22].

In this paper, we demonstrate that a similar reduction can be done for a larger subset of XPath called Configuration Logic (CL). CL has been presented in [24] as a formalism tailored for validating logical properties on the configuration of network routers expressed as XML parameter-value hierarchies. The major difference between CL and Simple XPath is in the use of variables: in Simple XPath, all references to the node labels of the parsed tree are constant, whereas CL allows the use of existentially and universally quantified variables to stand for node labels and be tested for equality. The construction we describe is inspired in part from the trace semantics of Core XPath described in [21], but adapted to a branching-logic setting.

The main contribution of our work is twofold. First, we show in section 2 that the use of variables in CL is closely related, and actually extends, the use of the "freeze" quantifier already developed for linear time logics [4, 15] to branching time logics. Second, in section 3, we use this approach to embed CL formulas into CTL. Section 4 concludes and indicates further avenues of investigation.

2 Configuration Logic

In this section, we briefly recall the basic structure of Configuration Logic (CL) and its intended purpose. We further show how the variable quantification mechanism of CL extends the notion of a "freeze" quantifier.

*We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada on this research.

2.1 Overview of CL

Simply put, CL is a logic over tree structures whose nodes are name-value pairs. It has been introduced in [24] in the context of configuration management of network routers.

In the CL framework, a *configuration* is a forest such as the one shown in Figure 1. This particular kind of structure has been introduced as a way of representing the configuration parameters of network routers in a hierarchical fashion that mirrors the organisation of their command line interface into modes, submodes, interfaces, and so on. Clearly, a configuration can be encoded as a particular kind of XML document, and this document can be readily exchanged by standard configuration protocols such as Netconf [16, 20].

Each node is formed of two parts: a *name* and a *value*, represented in the form "name = value". In typical network configurations, examples of names are `interface`, `router`, `ip-address`. To simplify things, we will top each forest with an additional source node connected to the roots of all trees and consider from now on that a configuration is a tree. For example, Figure 1 shows a configuration formed of two trees whose respective roots, a $= 1$ (node 1) and a $= 6$ (node 2), are linked to a source node.

Definition 1. *A* configuration *is a structure of the form* $\langle V, N, \tilde{R}_1, ..., \tilde{R}_n \rangle$ *where:*

- V *is a set, whose elements are called* values.
- N *is a set of words closed under prefix, on the alphabet formed of* $(p = v)$*, with* p *a name and* $v \in V$*. The elements of* N *are called* nodes.
- $\tilde{R}_1, \ldots, \tilde{R}_n$ *are relations on* V *(i.e. subsets of* $V^{arity(R_1)}, \ldots, V^{arity(R_n)}$ *respectively).*

In this paper, the only relation we consider is equality.

Named paths The succession of name-value pairs from the source node to an arbitrary node is called a *named path*; two nodes are considered identical if they have the same named path. For example, the following expression, when referring to Figure 1, corresponds to a traversal of the configuration that starts at the source node and goes all the way down to node 10.

$$\text{a} = 6\,,\ \text{b} = 6\,,\ \text{e} = 7 \tag{1}$$

A named path can have one or more variables in place of the value part of some nodes. In this case, depending on the values taken by those variables, the named path will designate different nodes. For example, the named path

$$\text{a} = 6\,,\ \text{d} = x \tag{2}$$

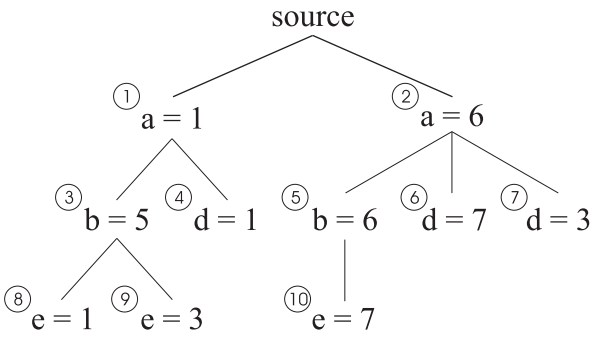

Figure 1. A sample configuration composed of two trees linked by a source node. Names and values are abstract.

designates node 6 if $x = 7$, node 7 if $x = 3$, and no node of the configuration otherwise. Variables standing for the name of nodes are not authorised.

Finally, the $*$ symbol is a shortcut that can be replaced by any number of nodes in a named path. For example, the named path

$$*\,,\ \text{d} = 3 \tag{3}$$

designates a node "d = 3" at any depth in the tree; in the case of Figure 1, this expression corresponds to two named paths for the nodes 4 and 7. This shortcut symbol has a tricky semantics and is not part of the original definitions in [24], as no real-world network property uses it. It is only included here for the sake of later comparison with Simple XPath.

Quantifiers CL is the formalism tailored for expressing logical properties on the values occuring in the nodes of configurations. It is composed of the traditional Boolean connectives and allows existential ($\langle\,\rangle$) and universal ($[\,]$) quantification on the "value" part of "name = value" pairs. Quantification in CL resembles in some way to the restricted "next" operator in sub-LTL [23] which does not consider all possible future states, but only a subset of them that has a specific label.

$$\langle \overline{p}\,;\, n = x \rangle\, \varphi \tag{4}$$

Equation (4) shows the form of an existential quantifier where $\overline{p}$ is a (possibly empty) named path, n is a name and x is a variable free in φ. Note that in this quantification, only x is considered bound; the other variables possibly occurring in $\overline{p}$ are considered free. Therefore, as stated above, the actual admissible values for x depend on how the named path that precedes it is valuated. Semantically, the previous quantification means that there exists a value c of x for

which the node designated by the named path $\overline{p}$, $n = c$ exists, such that $\varphi[x/c]$ is true.

$$[\overline{p};\, n = x]\ \varphi \tag{5}$$

Universal quantification, shown in (5), likewise asserts that all values c for which the node designated by the named path $\overline{p}$, $n = c$ exists, are such that $\varphi[x/c]$ is true.

Formally, the semantics of CL have been stated in [24] as the following.

Definition 2. *Let $\mathcal{C} = \langle V, N, \tilde{R}_1, ..., \tilde{R}_n\rangle$ be a configuration and ρ be a valuation for this configuration. We say that $\mathcal{C}, \rho$ satisfies a configuration logic formula φ (in notation $\mathcal{C}, \rho \models \varphi$), if recursively:*

- $\mathcal{C}, \rho \models R_i(\bar{x})$ *if* $\tilde{R}_i(\rho(\bar{x}))$ *holds*
- $\mathcal{C}, \rho \models \varphi \wedge \psi$ *if* $\mathcal{C}, \rho \models \varphi$ *and* $\mathcal{C}, \rho \models \psi$
- $\mathcal{C}, \rho \models \varphi \vee \psi$ *if* $\mathcal{C}, \rho \models \varphi$ *or* $\mathcal{C}, \rho \models \psi$
- $\mathcal{C}, \rho \models \neg\varphi$ *if* $\mathcal{C}, \rho \not\models \varphi$
- $\mathcal{C}, \rho \models \langle \bar{p} = \bar{x}; p = x\rangle\varphi$ *if there exists a* $v \in V$ *such that* $(\bar{p} = \rho(\bar{x}))(p = v) \in N$ *and* $\mathcal{C}, \rho[x/v] \models \varphi$
- $\mathcal{C}, \rho \models [\bar{p} = \bar{x}; p = x]\varphi$ *if for all* $v \in V$ *such that* $(\bar{p} = \rho(\bar{x}))(p = v) \in N$ *it holds that* $\mathcal{C}, \rho[x/v] \models \varphi$

We suppose that formulas are *well-named* –that is, each variable is quantified only once. It can be easily shown that every CL formula can be transformed to an equivalent well-named formula. As previously explained, the original semantics of CL can be extended by allowing the condition $(\bar{p} = \rho(\bar{x}))(p = v) \in N$ to include $*$ symbols that stand for any number of nodes.

Sentences A sentence is a closed CL formula. For example, the formula

$$\begin{array}{l}\langle\,;\, \mathrm{a} = x_1\rangle \\ \quad \langle \mathrm{a} = x_1\,;\, \mathrm{b} = x_2\rangle \\ \quad\quad x_1 \neq x_2\end{array} \tag{6}$$

states that there exists a child of the source node with name "a", which itself has a child with name "b" and a different value. One can see that this property is true for the tree of Figure 1 by considering nodes 1 and 3. The named path figuring in a quantifier is important, as it constrains the admissible values of a variable. For example, the following formula differs only from (6) in the named path of the second quantifier.

$$\begin{array}{l}\langle\,;\, \mathrm{a} = x_1\rangle \\ \quad \langle\,;\, \mathrm{b} = x_2\rangle \\ \quad\quad x_1 \neq x_2\end{array} \tag{7}$$

It states that there exists a child of the source node with name "a", and another child of the source node with name "b", and that both have different values; this formula is false on the tree of Figure 1.

CL also allows to express constraints on values in more than one branch. As a more complex example, consider the following formula:

$$\begin{array}{l}[\,;\, \mathrm{a} = x_1] \\ \quad \langle \mathrm{a} = x_1\,;\, \mathrm{b} = x_2\rangle \\ \quad\quad \langle \mathrm{a} = x_1\,;\, \mathrm{d} = x_3\rangle \\ \quad\quad\quad x_2 \neq x_3\end{array} \tag{8}$$

This formula states that every child of the source with name "a" has at least two children: one of name "b", and one of name "d" that have different values. For example, in Figure 1, the first node of name "a" under the source is node 1; it has a "d" child in node 4, whose value is different from the "b" node 3. One can find a similar pairing (nodes 5 and 6) under node 2, and therefore (8) is true on the configuration of Figure 1. Remark that in this case, the values that are ultimately compared (x_2 and x_3) are taken along two different branches: this is what gained CL the appellation of a *multi-site* modal logic.

The reader is directed to [24] for a more detailed presentation of CL, and to [19] for further examples of applications to network constraints.

2.2 CL versus Simple XPath

We now proceed to sketch the differences between CL and Simple XPath as presented in [2, 18].

First, while CL is a *logic* whose sentences are either true or false for a given configuration, Simple XPath is a language aimed at extracting parts of an XML document according to some criteria. Therefore, XPath expressions are not sentences that are true or false, but queries that return a set of nodes called an *answer set*. In [3], a Simple XPath query φ is converted to a sentence by interpreting it as the assertion "the answer set of φ contains the root of the tree", which is either true or false.

Second, one must remark that CL separates a node between name ane value; Simple XPath does not make this distinction. However, generic XML trees can be transformed into configurations by taking each label as the value of the node, and by appending the same dummy name, say "n", to each node.

Apart from that, one can see that the semantics of Simple XPath, given in [3], is very similar to CL. Actually, the interpretations of all predicates and Boolean connectives coincide with CL's. A Simple XPath "sentence" is a CL formula where equality testing only occurs between a variable and a constant, but never between two variables. For example, consider the following XPath query, called Q1 in [3]:

$$/[\text{child::site}/\text{child::regions}/\text{child::africa}/ \text{child::item}/\text{child::description}/ \text{child::parlist}/\text{child::listitem}/\text{child::text}] \quad (9)$$

This formula can be translated to the following CL formula:

$$\begin{array}{l} \langle\,;\, \mathrm{n} = x_1\rangle \\ \quad \langle \mathrm{n} = x_1\,;\, \mathrm{n} = x_2\rangle \\ \quad \ldots \\ \quad \langle \mathrm{n} = x_1\,,\, \mathrm{n} = x_2\,,\, \mathrm{n} = x_3\,,\, \mathrm{n} = x_4\,, \\ \qquad \mathrm{n} = x_5\,,\, \mathrm{n} = x_6\,,\, \mathrm{n} = x_7\,,\, \mathrm{n} = x_8\rangle \\ \qquad\quad x_1 = \text{site} \wedge x_2 = \text{regions} \wedge \ldots \\ \qquad\qquad \wedge\, x_7 = \text{listitem} \wedge x_8 = \text{text} \end{array} \quad (10)$$

As another example, query Q3, stated in formula (11), becomes (12) in CL.

$$/\,[\text{descendant::item}/\text{descendant::text}] \quad (11)$$

$$\begin{array}{l} \langle *\,;\, \mathrm{n} = x_1\rangle \\ \quad \langle *\,,\, \mathrm{n} = x_1\,;\, *\,;\, \mathrm{n} = x_2\rangle \\ \qquad x_1 = \text{item} \wedge x_2 = \text{text} \end{array} \quad (12)$$

Conversely though, there exist CL formulas that cannot be expressed in Simple XPath. Formulas (6) and (8) are two examples of properties that cannot be expressed in Simple XPath, as they compare two quantified node labels, feature that Simple XPath does not allow.

There is one feature of Simple XPath that is not included into CL: the support for *relative* queries, i.e. queries φ that are evaluated at every node of a tree and whose answer set contains the nodes for which the query evaluates to true; these queries are identified in XPath by the // symbol. However, we can use the same procedure as in [3] to handle them —that is, to root the formula successively in every node of the tree, and to return the nodes for which it is true. This fact allows us to say the translation presented here actually extends previous work.

2.3 CL as a Freeze Logic

The quantification process in CL bears some resemblance with the Timed Propositional Temporal Logic (TPTL) [4], and in particular with its concept of "freeze" quantifier. TPTL can be summarised as a Linear Temporal Logic (LTL) with an additional quantifier, $x.$, that allows to retain a value in a variable x at some specific instant in the evolution of a system to be compared at a later time with another value. In the particular case of TPTL, one is interested in freezing the value of a global clock. For example, the formula

$$\mathbf{G}\, x.(p \rightarrow \mathbf{F}\, y.(q \wedge y \leq x + 10)) \quad (13)$$

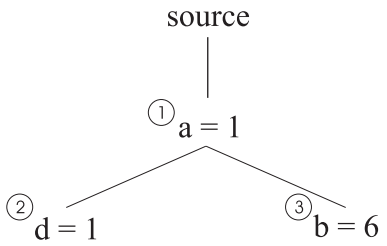

Figure 2. A small configuration forest

states that at every state where p holds, there exists a future state where q holds within 10 clock ticks. When the **G** operator is evaluated, the value of the global clock where p holds is stored in x; when the **F** operator is evaluated, the value of the global clock where q holds is stored in y.

The quantification process in CL can be interpreted in a similar way. Instead of a global clock ticking at each state change, we consider the *value* part of the name-value pair encountered at each node along a name path. The named path inside a quantifier starts at the root of the tree and descends down a specific branch; at the end of the path, the value in that particular node is fetched and stored into the quantified variable. It can then be recalled further in the formula and be used either to orient the descent down a branch, or be compared for equality with another value.

For example, consider the CL formula (6) validated against the tree of Figure 2. The first quantifier, $\langle \mathrm{a} = x_1\rangle$, starts from the source node of the configuration and descends into a node with name "a"; this is represented by arrow 1 in Figure 3. It freezes the value of that node into variable x_1. The second quantifier, $\langle \mathrm{a} = x_1\,;\, \mathrm{b} = x_2\rangle$, then restarts from the source node (arrow 2), goes down the node "a = x_1" (arrow 3), and then down a node of name "b" (arrow 4); it freezes the value of that node in x_2. Finally, the last part of the formula returns back to the source node of the configuration (arrow 5) and checks that $x_1 = x_2$.

Therefore, the quantifiers in a CL sentence describe multiple passes through the tree that freeze the value of one more variable on each pass. It is therefore more than a mere CTL with freeze, which would only be able to compare values that have been frozen along the same path. Figure 3 shows only one possible traversal of the configuration depicted in Figure 2.

In the previous example, there was only one way of traversing the tree on each pass; however, in most trees, there are multiple possible passes. This would be the case in Figure 1. Depending on whether the quantifiers in a formula are existential or universal, the CL formula asserts something about all possible traversals or only some of them.

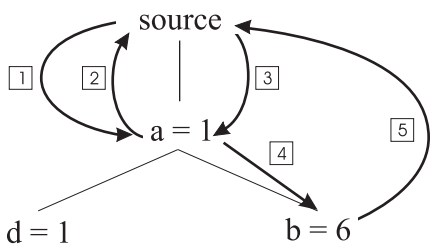

Figure 3. Quantification in CL is a traversal of the tree ended by the freezing of a node's value into a variable

3 Embedding CL into CTL

The observation that variable quantification in CL can be seen as a special type of freeze quantification will be used to perform our embedding of CL into CTL. We proceed in two steps: first, we show how to convert a configuration T and a CL formula φ to a Kripke structure $K_{T,\varphi}$; then, we show how a CL formula φ can be translated to a CTL formula φ' and show that $T \models_{\text{CL}} \varphi$ if and only if $K_{T,\varphi} \models_{\text{CTL}} \varphi'$, thereby reducing the problem of CL model checking to CTL model checking.

The resulting Kripke structure depends on φ only for the number of distinct quantified variables that appear in it. Therefore, for a fixed number of quantified variables, the translation of T is the same for all formulas. In the following, we will simply denote it by K_T by assuming all considered formulas have less than k variables for some $k > 0$.

3.1 From a Configuration to a Kripke Structure

The Kripke structure K_T is built in such a way that each state of the system is the image of some node of T.

The conversion procedure we show takes a configuration T and a CL formula φ and generates a Kripke structure $K_T = (S, I, R, L)$, where S is the set of states, I is the set of initial states, $R \subseteq S \times S$ is a total transition relation and L is a labelling function $S \to 2^{AP}$, for AP a set of atomic propositions. For the sake of clarity, we will abuse notation and use state variables that take their values in arbitrary discrete sets instead of Boolean; the set AP is therefore the set of atomic propositions that encode into Boolean variables the discrete values occurring in the structure. This operation is identical as in [2, 3].

Actually, each node of T will have many images in K_T; there are as many copies as there are possible combinations of values for the quantified variables appearing in the formula.

Set of states Each state of K_T (with the exception of a source and a sink node) is intended to mimick one, and only one, node of the original tree T. Let N be the set of all names appearing in T, and V be the set of all values appearing in T. For the tree of Figure 2, we have $N = \{a, b, d\}$ and $V = \{1, 6\}$. From these sets, create the sets N' and V' by adding a special, unused symbol # that will stand for "undefined". A state of S is uniquely identified by $k + 2$ state variables:

- $\alpha \in N'$ contains the name of the node in the original tree
- $\beta \in V'$ contains the value of the node in the original tree
- $x_1, \ldots, x_k \in V'$ contain the values of the quantified variables that are frozen when a formula is evaluated; they can hold either a node value, or the "undefined" symbol #.

The set S of states of K_T is the subset of $N' \times V' \times {V'}^k$ such that $s = (\alpha, \beta, x_1, \ldots x_k) \in S$ if and only if there exists a node in T labelled $\alpha = \beta$. By convention, the special source node of every configuration has both name and value equal to #. For example, if T is the tree of Figure 2 and φ is formula 6, then the state where $\alpha = a$, $\beta = 1$, $x_1 = x_2 = \#$ is an element of S that is associated with node 1.

One can remark that for each node in T, there are multiple states in S that are associated to it, namely one for each possible valuation of the $x_1, \ldots, x_k$. We will call a *copy* of T the subset of S in which the x_i are valuated in the same way. There are $|V'|^k$ copies of T in S. The set I contains only one initial state: it is the source state of the copy of T where all x_i are undefined, i.e. $(\#, \#, \#, \ldots, \#)$.

Transition relation Next, we define the transition relation R of that Kripke structure. There are two kinds of transitions in R. Let $s_1 = (\alpha_1, \beta_1, x_{1,1}, x_{1,2}, \ldots, x_{1,n})$ and $(\alpha_2, \beta_2, x_{2,1}, x_{2,2}, \ldots, x_{2,n})$ be two states in S.

The first set of transitions are *tree* transitions: they correspond to a descent into the original tree. The tree transitions in R link all the states inside each copy of T according to the original tree structure. In such transitions, no quantified variable changes its value. Formally, this represents the transitions (s_1, s_2) in R such that s_2 is a child of s_1 in the original tree and $x_{1,i} = x_{2,i}$ for all $1 \leq i \leq k$. In Figure 3, the arrows 1, 3 and 4 are represented in K_T by tree transitions.

The second set of transitions are the *freeze* transitions exposed in section 2.3. They correspond to the arrows 2 and 5 of Figure 3, when the value of a node is frozen into a variable and the pass into the tree is ended by a return

to the source node. In K_T, these transitions always lead to the source state of a copy of the original tree where all variables keep their values, except for one x_i that switches from # to some defined value. The value that this x_i takes is the value (β) of the node from which the switch has been made, therefore depicting the freezing of that value into x_i. Formally, this represents the transitions (s_1, s_2) in T such that there exists a $1 \leq j \leq n$ such that $x_{1,i} = x_{2,i}$ for all $i \neq j$, $\beta_1 = v$ for some $v \in V$, $\alpha_2 = \beta_2 = \#$, $x_{1,j} = \#$ and $x_{2,j} = v$.

As has been explained in section 2.3, a quantified CL formula describes multiple passes down T that each freeze a node value into some variable. A trace in K_T represents just that: a sequence of descents down T along some named paths; at the end of each named path, a variable x_i is quantified, and the value v of the node at this point is frozen in x_i. This freezing is done by switching to the copy of T where x_i has value v. The traversal of T caused by the next quantifier then restarts from its source node, which is the image of T's source state in the copy where x_i has been frozen by the previous quantifier.

Optimisations It is possible to further optimise the structure. First, one can assume that the variables are always quantified in the order $x_1, x_2, \ldots, x_n$. Moreover, every formula that does not respect this condition can be transformed by a simple renaming of its variables to a formula that complies with this restriction. Doing so reduces the size of the structure and the number of transitions, as only one variable at a time can change from # to some value.

Second, the copies of T where all variables are defined can be trimmed of everything but their source state. This is possible because at that point in the evaluation of a formula, no new variable can be quantified, and therefore no further pass down the tree is possible. Hence, the nodes of these copies of T are useless, since they can never be accessed.

The resulting Kripke structure obtained from Figure 2 and formula (6) is shown in Figure 4. Solid arrows represent tree transitions; dotted arrows are the freeze transitions. The optimisations discussed above have been applied to the graph for the sake of clarity; however, the remaining of the discussion will suppose a complete structure as defined previously.

3.2 From CL to CTL

From the Kripke structure K_T constructed in the previous section, it becomes straightforward to translate CL's semantics into CTL. We proceed as in [3] and define a linear embedding ω of CL into CTL formulas, with ϵ as the empty string. Let φ and ψ be CL formulas, c be a constant in V, m and n be names in N, and the x_i be quantified variables. Translating the Boolean connectives and the ground equality testings is direct:

$$\omega(x_i = c) = x_i = c \tag{14}$$
$$\omega(x_i = x_j) = x_i = x_j \tag{15}$$
$$\omega(\varphi \wedge \psi) = \omega(\varphi) \wedge \omega(\psi) \tag{16}$$
$$\omega(\varphi \vee \psi) = \omega(\varphi) \vee \omega(\psi) \tag{17}$$
$$\omega(\neg\varphi) = \neg\omega(\varphi) \tag{18}$$

The translation of named paths and quantifiers is slightly more complicated. It proceeds by nesting one element of the named path at a time until reaching the semicolon; as a consequence of the semantics of CL, this process is identical in both existential and universal quantifiers. Then, the quantified part is translated. Here as before, we designate by $\overline{p}$ a (possibly empty) named path as defined in section 2.1.

$$\begin{aligned}\omega(\langle n = x_i\,,\,\overline{p}\,;\,m = x_i\rangle\varphi) = \\ \mathbf{EX}\,(\alpha = n \wedge \beta = x_i \\ \wedge\, \omega(\langle\overline{p}\,;\,m = x_i\rangle\varphi))\end{aligned} \tag{19}$$

$$\begin{aligned}\omega([n = x_i\,,\,\overline{p}\,;\,m = x_i]\,\varphi) = \\ \mathbf{EX}\,(\alpha = n \wedge \beta = x_i \\ \wedge\, \omega([\overline{p}\,;\,m = x_i]\,\varphi))\end{aligned} \tag{20}$$

$$\begin{aligned}\omega(\langle *\,,\,\overline{p}\,;\,m = x_i\rangle\varphi) = \\ \mathbf{EF}\,\omega(\langle\overline{p}\,;\,m = x_i\rangle\varphi)\end{aligned} \tag{21}$$

$$\begin{aligned}\omega([*\,,\,\overline{p}\,;\,m = x_i]\,\varphi) = \\ \mathbf{EF}\,\omega([\overline{p}\,;\,m = x_i]\,\varphi)\end{aligned} \tag{22}$$

$$\begin{aligned}\omega(\langle\,;\,n = x_i\rangle\varphi) = \\ \mathbf{EX}\,((\alpha = n \wedge x_i = \#) \\ \wedge\, \mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi)))\end{aligned} \tag{23}$$

$$\begin{aligned}\omega([\,;\,n = x_i]\,\varphi) = \\ \mathbf{AX}\,((\alpha = n \wedge x_i = \#) \\ \rightarrow \mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi)))\end{aligned} \tag{24}$$

For example, formula (6) is translated to the following CTL expression:

$$\begin{aligned}&\mathbf{EX}\,((\alpha = \mathsf{a} \wedge x_1 = \#) \wedge \\ &\quad\mathbf{EX}\,(x_1 \neq \# \wedge \\ &\qquad\mathbf{EX}\,(\alpha = \mathsf{a} \wedge \beta = x_1 \wedge \\ &\qquad\quad\mathbf{EX}\,((\alpha = \mathsf{b} \wedge x_2 = \#) \wedge \\ &\qquad\qquad\mathbf{EX}\,(x_2 \neq \# \wedge \\ &\qquad\qquad\quad x_1 \neq x_2)))))\end{aligned} \tag{25}$$

The trace that makes this formula true on K_T is shown in bold arrows in Figure 4. Note that this trace represents two passes in the tree where appropriate node values are frozen into variables x_1 and x_2, as explained in section 2.3. The trace in K_T is nothing but the succession of states, including the values of frozen variables, that occur when one traverses the original tree as shown in Figure 3

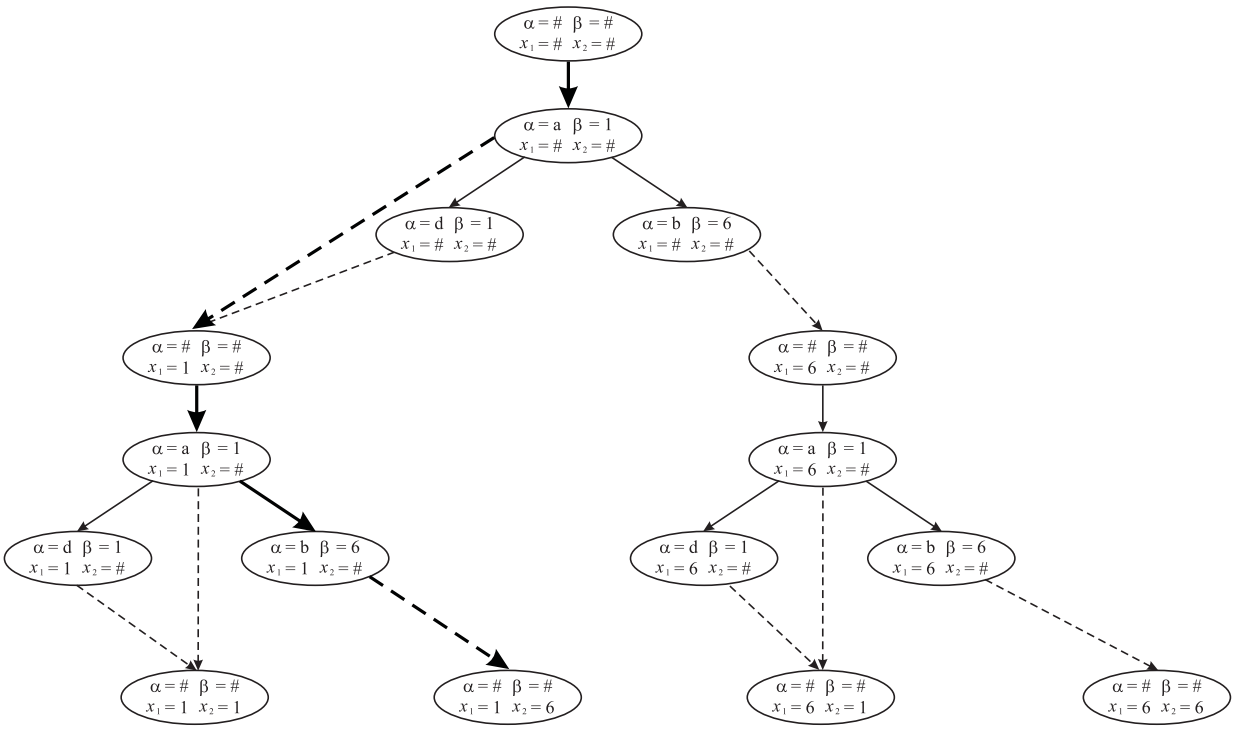

Figure 4. The resulting transition system K_T obtained from Figure 2 and formula (6).

Theorem 1. *Let T be a configuration, φ be a CL formula and ω be the embedding defined previously. Let K_T be the Kripke structure built as shown in the previous section. Then $T \models_{CL} \varphi$ if and only if $K_T \models_{CTL} \omega(\varphi)$.*

Proof. To demonstrate the equivalence of this embedding, we need to introduce the notion of *restriction* of a Kripke structure K_T to some valuation ρ. This restriction, noted $K|_\rho$, is the Kripke structure obtained from K by keeping only the states where the values of the x_i agree with ρ. More precisely, we have that $\rho(x_i) = v \leftrightarrow K|_\rho \models_{CTL} x_i = v$ for $v \in V$. Constants are mapped identically. The initial state of $K|_\rho$ is the state $\alpha = \#, \beta = \#$ with the smallest set of defined variables that agree with ρ. One can easily show that, from the construction of K, this state is unique.

The proof is then done by structural induction. The base case is ground equality testing. $T, \rho \models_{CL} x_i = x_j$ if and only if $\rho(x_i) = \rho(x_j)$, by Definition 2. But this is true in turn if and only if both $K|_\rho \models_{CTL} x_i = \rho(x_i)$ and $K|_\rho \models_{CTL} x_j = \rho(x_j)$, which is equivalent to $K|_\rho \models_{CTL} x_i = x_j$. A similar reasoning can be made in the case of the comparison of a variable with a constant.

For the induction step, we check the cases one by one.

$T, \rho \models_{CL} \varphi \wedge \psi$ if and only if $T, \rho \models_{CL} \varphi$ and $T, \rho \models_{\text{CL}} \psi$, by Definition 2. By the induction hypothesis, these two assertions are equivalent to $K|_\rho \models_{\text{CTL}} \omega(\varphi)$ and $K|_\rho \models_{\text{CTL}} \omega(\psi)$, which by (16) is equal to $K|_\rho \models_{\text{CTL}} \omega(\varphi \wedge \psi)$. A similar reasoning can be done for disjunction and negation.

The case of quantification is more complex. As stated in Definition 2, $T, \rho \models_{\text{CL}} \langle \overline{p}\,;\, n = x_i \rangle \varphi$ if and only if

1. there exists a $v \in V$ such that $\rho(\overline{p}\,,\, n = v)$ is a node of T and
2. $T, \rho[x_i/v] \models_{\text{CL}} \varphi$.

The first part is true if and only if there exists a path from the initial state of $K|_\rho$ that matches $\rho(\overline{p}\,,\, n = v)$; but this is direct from the translation of named paths in (19)-(22) and the way $K|_\rho$ has been constructed in Section 3.1.

The second part is true if and only if there exists a way to extend ρ to ρ' in such a way that $\rho = \rho'$ except for $\rho'(x_i) = v$. By the way K is built, this happens if there exists a way to transit from the current copy of T to a copy where x_i is no longer undefined (#), but rather takes value v (the same v as the value of the node at the end of the named path); in such a transition, all x_j ($j \neq i$) keep their already defined values, except for x_i which switches from # to v. Moreover, such a transition leads to a copy of the source node of T; what is accessible from this state is nothing but $K|_{\rho'}$. Then, by induction hypothesis, we have that $T, \rho' \models_{\text{CL}} \varphi$ if and only if $K|_{\rho'} \models_{\text{CTL}} \omega(\varphi)$. A similar reasoning can be made for the case of the universal quantifier. □

3.3 Theoretical Consequences

The embedding described in Section 3.2 is linear; that is, if we denote by $|\varphi|$ the length of a CL formula φ, then $|\omega(\varphi)| \in O(|\varphi|)$. It suffices to remark that each translation rule consumes at least one symbol of the original CL formula and contributes a fixed number of symbols in the resulting CTL formula. Moreover, as explained in section 3.1, K_T has $|V'|^k$ copies of T, with k the maximum number of quantified variables occurring in φ; but clearly, $|V| \leq |T|$, as there cannot be more different values as there are nodes in T, and therefore $|K_T| \in O(|T|^k)$.

Since the problem of validating a CL formula can be reduced to the problem of model checking a particular CTL formula, all theoretical results proved for CTL also apply to this particular context of CL.

Complexity of model checking Model checking CTL formulas is decidable and has time complexity of $O(|\varphi| \times |K|)$ [13], where $|K|$ is the size (vertices + edges) of the Kripke structure. This gives us that the model checking of CL is decidable and has time complexity of $O(|\varphi| \times |T|^k)$. This means that model checking is polynomial in the size of the tree and the size of the formula, but exponential in the number of different quantified variables appearing in φ. The optimisations suggested in section 3.1 do not change this complexity.

This result places CL in comparison with other tree languages and logics mentioned in Section 1. It can be shown that CL is a fragment of TQL. [8] has shown that the model checking, validity and satisfiability problems for closed formulas in TQL with no quantifiers is decidable. Moreover, [6] demonstrated that model checking in TQL is PSPACE-hard; this result also applies to CL by using the same argument. This polynomial space bound is effectively achieved when using on-the-fly model checking methods.

In turn, as section 2.2 showed it, Simple XPath is a fragment of CL. It has been demonstrated in [2] that Simple XPath model checking is equivalent, in terms of time complexity, to CTL model checking ($O(|\varphi| \times |T|)$), a result already proved in a different way in [18]. Our result is accordingly more complex, as the logic presented here is richer; however, it is interesting to remark that it preserves polynomiality in terms of formula length and tree size.

Choice of the approach These complexity results also validate the choice of approach used for embedding CL into CTL. In the method presented here, the Kripke structure is responsible for handling the values of the quantified variables by containing one copy of the original configuration for each possible assignment of the x_i and by linking these copies by freeze transitions. A different approach would have been to create a Kripke structure K_T containing a single copy of T without taking into account the x_i as state variables. The handling of the quantified variables would be transferred instead to the formula by translating the quantifiers in the classical fashion:

$$\omega(\langle \overline{p}\,;\, n = x_i \rangle\, \varphi) = \bigvee_{v \in V(\overline{p})} \varphi\,[x_i/v] \tag{26}$$

$$\omega([\,\overline{p}\,;\, n = x_i]\, \varphi) = \bigwedge_{v \in V(\overline{p})} \varphi\,[x_i/v] \tag{27}$$

However, because of the semantics of CL, special care should be taken to consider for v only values that are at the end of the named path $\overline{p}$, and not all values of V; this is represented in the previous formulas by $V(\overline{p})$. Thus, instead of making multiple copies of the tree in K_T according to each possible value of the x_i, one makes multiple copies of the original CL formula φ. This would lead to a Kripke structure of size $O(|T|)$, and to a formula of length $O(|\varphi|^k)$. The complexity of model checking CL formulas in such a framework would therefore be of $O(|\varphi|^k \times |T|)$ instead of $O(|\varphi| \times |T|^k)$ as is shown here.

This alternate approach has several drawbacks. First, although the translation of a tree into a Kripke structure is straightforward, the translation of a CL formula into CTL becomes tightly coupled with the tree on which it has to be checked. This is because the translation of the quantifiers shown in (26)-(27) depends on the values occurring in the tree at the end of some specific named paths —therefore, restricting possible values to some predetermined set does not solve the problem, since the structure of each tree still has to be taken into account.

In practice, however, it is far more frequent to check the same formula on many configurations than the opposite. Therefore, the approach presented in this paper, where the translation of formulas does not depend in any way on the tree, is more efficient. Moreover, we have shown in section 3.1 that given a fixed maximum number of quantified variables, the translation of a configuration into a Kripke structure is also independent of any formula.

Second, on a more technical aspect, standard model checkers such as NuSMV [11] can easily handle systems with very large state spaces and reasonably short temporal formulas, but are far less efficient for checking exponentially long formulas on relatively small systems. It is therefore natural to choose an approach where the exponential is placed on system size rather than formula length.

4 Conclusion

In this paper, we have briefly recalled the origins of Configuration Logic (CL) and stated its differences with respect to Simple XPath. We have shown how a fragment

of XPath called Configuration Logic (CL) can be embedded into Computation Tree Logic. We have also shown that this framework embeds into CTL a larger subset of XPath than a previous work and in particular allows universally and existentially quantified variables in formulas. Finally, we have also demonstrated how the variable binding mechanism of CL can be seen as a branching-time equivalent of the "freeze" quantifier. This embedding of CL into CTL opens the way to the reverse process of using decidability and model existence of CTL to prove decidability and model existence of CL, a property that is highly desirable in the context of network configurations and that would make CL one of the few decidable tree logics around.

References

[1] S. Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.

[2] L. Afanasiev. XML query evaluation via CTL model checking. In *ESSLLI*, pages 1–12, 2004.

[3] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke. CTL model checking for processing simple XPath queries. In *TIME*, pages 117–124. IEEE Computer Society, 2004.

[4] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[5] S. Boag, D. Chamberlin, and M. F. Fern. XQuery 1.0: An XML query language, W3C working draft, 2005.

[6] I. Boneva and J.-M. Talbot. On complexity of model-checking for the TQL logic. In *TCS 2004*, 2004.

[7] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeay, and J. Cowan. Extensible markup language (XML) 1.1, 2004.

[8] C. Calcagno, L. Cardelli, and A. D. Gordon. Deciding validity in a spatial logic for trees. In *TLDI*, 2003.

[9] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[10] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *FoSSaCS*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[11] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV: An opensouce tool for symbolic model checking. In *CAV 2002*, pages 359–364, 2002.

[12] J. Clark and S. DeRose. XML path language (XPath) version 1.0, W3C recommendation, 1999.

[13] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[14] C. S. Coen, P. Marinelli, and F. Vitali. Schemapath, a minimal extension to XML Schema for conditional constraints. In *WWW*, pages 164–174. ACM, 2004.

[15] S. Demri, R. Lazić, and D. Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. In *TIME*, pages 113–121, 2005.

[16] R. Enns. Netconf configuration protocol, IETF working draft, 2005.

[17] D. C. Fallside and P. Walmsley. XML schema part 0: Primer second edition, W3C recommendation, 2004.

[18] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. In *VLDB*, pages 95–106, 2002.

[19] S. Hallé, R. Deca, O. Cherkaoui, and R. Villemaire. Automated validation of service configuration on network devices. In J. B. Vicente and D. Hutchison, editors, *MMNS*, volume 3271 of *Lecture Notes in Computer Science*, pages 176–188. Springer, 2004.

[20] S. Hallé, R. Deca, O. Cherkaoui, R. Villemaire, and D. Puche. A formal validation model for the netconf protocol. In *DSOM*, pages 147–158, 2005.

[21] P. H. Hartel. A trace semantics for positive core XPath. In *TIME*, pages 103–112, 2005.

[22] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *PODS*, pages 65–76, 2002.

[23] A. Muscholl and I. Walukiewicz. An NP-complete fragment of LTL. In C. Calude, E. Calude, and M. J. Dinneen, editors, *Developments in Language Theory*, volume 3340 of *Lecture Notes in Computer Science*, pages 334–344. Springer, 2004.

[24] R. Villemaire, S. Hallé, and O. Cherkaoui. Configuration logic: A multi-site modal logic. In *TIME*, pages 131–137, 2005.