Forest expressions

Mikołaj Bojańczyk

Warsaw University

Abstract. We define regular expressions for unranked trees (actually, ordered sequences of unranked trees, called forests). These are compared to existing regular expressions for trees. On the negative side, our expressions have complementation, and do not define all regular languages. On the positive side, our expressions do not use variables, and have a syntax very similar to that of regular expressions for word languages.
We examine the expressive power of these expressions, especially from a logical point of view. The class of languages defined corresponds to a form of chain logic [5, 6]. Furthermore, the star-free expressions coincide with first-order logic. Finally, we show that a concatenation hierarchy inside the expressions corresponds to the quantifier prefix hierarchy for first-order logic, generalizing a result of Wolfgang Thomas [4].

1 Introduction

We define a new type of regular expressions for forests, i.e. ordered sequences of unranked trees. Like word regular expressions and unlike the known tree regular expressions, our expressions do not use variables. Similar to CTL*, we have two sorts of expressions: one describing forests, and one describing contexts (forests with a hole, which can be used for substitution). The two sorts are defined by mutual recursion, and each allows concatenation, star, and boolean operations (including complementation and intersection). Forest expressions do not capture all regular languages of unranked forests (or even trees), but they do have a characterization in terms of logic:

Theorem 11
Forest expressions have the same expressive power as extended chain logic. Star-free forest expressions have the same expressive power as first-order logic.

In the above theorem, the models for the logic are forests. The signature has label tests, and two partial orders: vertical and horizontal. The vertical order corresponds to descendants, and the horizontal order is used to say when one sibling is to the left of another. Extended chain logic is a fragment of MSO, where set quantification is restricted to two types of sets: chains (sets linearly ordered by the vertical order) and sibling sets (sets linearly ordered by the horizontal order). Over binary trees, the second quantification is superfluous, and the logic collapses to chain logic as defined by Thomas in [5]. The second statement in the above theorem can be seen as a tree extension of [3], where it is shown that star-free word expressions have the same expressive power as first-order logic.

Furthermore, we show that the correspondence between the quantifier alternation hierarchy, and the concatenation hierarchy, which has been shown for words by Thomas in [4], also works with our expressions:

Theorem 12
A forest language is on level $n^{1}/_{2}$ if and only if it is defined by a sentence in Σ_n.

This result is the second main contribution of this paper.

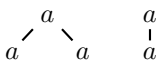
2 The regular expressions

In the formal syntax of expressions, there will be two sorts of expressions, one for forests and one contexts. Before we present this syntax in Definition 21, we will gradually introduce the operations allowed in the expressions.

We use $+$ to denote concatenation of forests (sequences of trees); while $\vee$ is used in the regular expressions for language union (likewise $\wedge, \neg$ for intersection and complement). For instance, both the expressions

$$a(a+a) \vee a(a) \qquad a((a+a) \vee a)$$

denote the same language containing two trees:



Concatenation of trees can be iterated using the Kleene star; for instance the expression

$$a((a \vee b)^*)$$

denotes all trees of depth at most two that have a in the root and a, b in the leaves. If we want to define languages of unbounded tree depth, we must also have some form of vertical iteration. Here, we use iteration of contexts, i.e. forests with a single hole. The hole is denoted by $\Box$, and is used for substitution. For instance, the set of a-labeled trees that have a single leaf is defined by the expression:

$$(a\Box)^* a \ .$$

Context composition is written multiplicatively, for instance $(aa\Box)^* a$ denotes trees with a single path of odd length.

The kind of a subexpression can be determined by the way it is used in the larger expression: if it is an argument of $+$ then it must be a forest expression, if it is an argument of the multiplicative context concatenation, then it must be a context expression. The only possible ambiguity is when the expression does not contain forest concatenation or context composition, eg. $a \vee (\neg b)$; by convention we assume the expression describes forests and not contexts.

Even with the two Kleene stars (for forests and contexts); our expressions cannot define trees with a large amount of branching. In particular, the set of

all trees cannot be defined. Therefore, as in star-free word languages, the key to success is judicious use of complementation. As usual with complementation, it is important that an alphabet A is specified beforehand; let us fix $A = \{a, b\}$ for the next several examples. For instance, $a(\neg\emptyset)$ defines the set of all trees over A that have a in the root (by convention, $\emptyset$ is the empty forest – and not context – language).

We have already discussed concatenation and Kleene star for forests and contexts, and the boolean operations. We have also implicitly embedded forests within contexts, for instance in $(a\Box)^*a$ we have embedded the forest a in the context $(a\Box)^*$. The last remaining operation is one that embeds a forest within a context; we use $+$ to add a context to a forest, with the result being a context. For instance,

$$\neg\emptyset + \Box + \neg\emptyset$$

denotes all contexts where the hole is at the root. In particular,

$$\big((a\Box \ \vee \ b\Box \ \vee \ (\neg\emptyset + \Box + \neg\emptyset)\big)^*$$

denotes all contexts. An equivalent expression would be $(\Box \wedge \ \neg\Box)$. Using this expression and complementation, we can define all contexts where only the letter a occurs.

Below is the formal definition of the expressions:

Definition 21 The syntax of forest and context expressions over an alphabet A is defined by the following grammar:

$$\mathcal{F} \to \emptyset \mid \epsilon \mid a \mid \mathcal{F} + \mathcal{F} \mid \mathcal{F}^* \mid \mathcal{C}\mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \neg\mathcal{F}$$
$$\mathcal{C} \to \Box \mid a\Box \mid \mathcal{C}\mathcal{C} \mid \mathcal{F} + \mathcal{C} \mid \mathcal{C} + \mathcal{F} \mid \mathcal{C} \vee \mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \mid \neg\mathcal{C}$$

In the above, a stands for any letter in the alphabet A. An expression is called *star-free* if it does not use the productions $\mathcal{F}^*$ or $\mathcal{C}^*$.

The semantics have already been defined in the discussion leading to the above definition. We only omitted ϵ, which stands for the empty forest. To avoid excess parentheses, we assume the following binding precedence order: context composition, $+$, $\neg$, $\wedge$ and finally $\vee$.

Note that Definition 21 contains some redundant rules. For instance, $F \wedge G$ is equivalent to $((F + \Box) \wedge (G + \Box))\epsilon$. However, we choose to keep all the rules to have a more convenient notation.

Before continuing with our discussion of these expressions, we would like to comment on complexity issues. Emptiness of a forest expression is EXPTIME hard; by encoding computations of an alternating polynomial space Turing machine. A matching EXPTIME upper bound can be found by compiling an expression into a bottom-up automaton with an exponential state space. Model-checking, on the other hand, is polynomial: one can check in polynomial time if a given forest belongs to a given forest expression by using a dynamic algorithm.

2.1 Comparison with existing expressions

The regular expressions usually considered for trees, see eg. Chapter 2 of [2], are different from our expressions in that:

1. They use arbitrarily many types of hole $\Box_1, \ldots, \Box_n$ (instead of one $\Box$).
2. The contexts can have holes of different types, and with multiple copies.
3. There is no negation or intersection.

Another difference is that we consider unranked trees instead of ranked trees, but the definitions from [2] can be adapted to the unranked setting, with similar results. The differences 1,2 can be seen as advantages of our expressions, while the difference 3 is a disadvantage. Another disadvantage is that our expressions cannot define all regular languages, as opposed to the standard expressions. For instance, the language "positive boolean expressions that evaluate to true" is not definable via our forest expressions (otherwise, this language would be definable in chain logic, which it is not), while it can be done using even with one type of hole, but with multiple copies of this hole. We present this expression below, to give the reader a taste of the expressions with many holes. We use the alphabet $\{OR, AND, 0, 1\}$ here to avoid confusion with $\vee, \wedge$ found in the syntax of expressions. The following expression describes all forests of properly formed expressions, possibly with some holes left:

$$E \quad = \quad (OR\Box \ \vee \ AND\Box \ \vee \ 0 \vee 1 \ \vee \ \Box + \Box)^* .$$

The following expression denotes contexts that take 1 to 1 (i.e. will give a result of 1 if all holes are substituted by 1):

$$F^* \qquad \text{where } F = AND(\Box^*) \ \vee \ OR((E \vee \Box)^* + \Box + (E \vee \Box)^*) .$$

Every true positive boolean expression can be decomposed as F^*1.

It should be remarked here that our forest expressions are not meant to replace the standard regular expressions with many holes. We only claim that they have a convenient syntax and, most importantly, disallowing multiple holes gives a close correspondence to logic, as witnessed by Theorems 11 and 12.

3 Forest automata

We denote forests by letters s, t, with forest concatenation denoted by $s + t$. We denote contexts by letters p, q, r, with context composition written multiplicatively as pq. The result of placing a forest s in the hole of a context p is denoted by ps. We use the standard relationships of nodes in a forest/context: descendant, ancestor, child, parent, sibling, leftmost sibling, rightmost sibling, left neighbor (closest sibling to the left), right neighbor.

Recall that a monoid is a set H along with an associative operation, which we denote here by $+$ to underline the connection with forest concatenation. Furthermore, each monoid has a unit element $\epsilon \in H$, which satisfies $\epsilon + h =$

$h + \epsilon = h$ for all $h \in H$. We will denote monoids by F, G, H and their elements by f, g, h (there are two justifications here: first, we want to avoid confusion with trees s, t, u; second we want to be consistent with notation in forest algebra, see [1]). A *monoid forest automaton* is a deterministic word automaton where the states have a monoid structure. In other words, the automaton has two components: a finite monoid H and a function $\delta : A \times H \to H$, where A is the label alphabet. The automaton assigns to each forest t a unique element $\mathcal{A}(t)$ of H; called the *value* of the forest. The empty forest is assigned the unit of the monoid, while for larger forests we have:

$$\mathcal{A}(t + s) = \mathcal{A}(t) + \mathcal{A}(t) \qquad \mathcal{A}(a(t)) = \delta(a, \mathcal{A}(t)) \ .$$

In the above, the left + is forest concatenation, while the right + is the monoid operation. Thanks to associativity, the above definition is nonambiguous. The automaton recognizes a language L if membership in L only depends on the value of a forest. From now on, we will only be using monoid forest automata; and we will simply call them forest automata.

Let $\mathcal{A} = (H, \delta)$ be a forest automaton, and let $g, h \in H$. We say a context p takes g to h if there is some forest s with value g such that ps has value h. Note that in the above we could have equivalently written "for any forest with value s".

Definition 31 We say a forest automaton $\mathcal{A} = (H, \delta)$ is *represented by expressions* if for all possible values $g, h \in H$

- There is an forest expression L_h describing forests with value h.
- There is a context expression L_g^h describing contexts p that take h to g.

A forest language L is represented by expressions if some forest automaton recognizing L is. Clearly, a language represented by expressions can be defined by a forest expression: it suffices to take the union of L_m for all values m of forests in the language. Furthermore, the proof of Theorem 11 also gives the converse: if a language is defined by an expression, then its also represented by expressions.

4 Equivalence with first-order logic

It is fairly easy to show the left to right inclusions from Theorem 11: extended chain logic and first-order logic can capture forest expressions and star-free expressions, respectively. We only give the proof of the more difficult right to left inclusions here.

Proposition 41 Forest languages definable in first-order logic can be represented by expressions.

The proof of this proposition is rather standard and proceeds by induction on the formula of first-order logic. Later on in the paper, we present a stronger result, however we include the below proof since it is significantly shorter.

To go through the induction step, we need to prove the statement also for formulas with free variables. A formula

$$\varphi(x_1, \ldots, x_n)$$

with free variables $x_1, \ldots, x_n$ can be seen as a forest language $L(\varphi)$ over an extended alphabet $A \times \{0,1\}^n$:

$$t \in L(\varphi) \qquad \text{iff} \qquad t, \nu \models \varphi \ ,$$

where ν is the valuation that assigns to x_i the unique node with 1 on the i-th $\{0,1\}$ coordinate. (If the forest has 0 or at least 2 nodes with 1 on a given coordinate, the valuation ν is undefined and therefore $t, \nu \models \varphi$ cannot hold.)

The proof of Proposition 41 is a rather standard induction on the size of the formula φ. The only nontrivial step is when passing from φ to $\exists x.\varphi$ (we eliminate $\forall$ using negation). Let then $\mathcal{A} = (H, \delta)$ be an automaton recognizing a forest language L over an alphabet $A \times \{0,1\}$. We need to show that if $\mathcal{A}$ is represented by star-free expressions, then so is an automaton recognizing the language

$$K = \{t : t[x] \in L \text{ for some node } x \text{ of } t\} \ .$$

In the above, $t[x]$ is the forest over over $A \times \{0,1\}$ obtained from t by adding label 1 on the second coordinate of the node x, and label 0 for the remaining nodes. We first define a forest automaton $\mathcal{B}$ that recognizes the language K; then we will show star-free expressions that represent this automaton. The value under $\mathcal{B}$ of a forest in t is a pair (h, G). The first coordinate is the value (in $\mathcal{A}$) of $t[\emptyset]$, while the second coordinate is the set of values (in $\mathcal{A}$) of forests $t[x]$ for all possible nodes x. The reader will easily fill in the monoid operation on states of $\mathcal{B}$, and its transition function.

We need to show that the automaton $\mathcal{B}$ is represented by star-free expressions. First, we will define some auxiliary star-free expressions. We begin by describing, for every $g, h \in H$, the forests and contexts where the node x is not present:

- K_h is a star-free forest expression describing forests t over A where $t[\emptyset]$ has value h.
- K_g^h is a star-free context expression describing contexts p over A where $p[\emptyset]$ takes g to h.

These star-free expressions are easily obtained from the star-free expressions representing the automaton $\mathcal{A}$, by writing $a \in A$ instead of $(a,0), (a,1)$ and then intersecting with a language that forbids labels of the form $(a,1)$. Next, we write for each $g, h \in H$ star-free expressions that describe forests and context where the node x is present. These are defined by distinguishing the node x:

– L_h is star-free forest expression describing forests t over A where $t[x]$ has value h, for some node x of t. This is the union of star-free expressions

$$K_g^h a K_f$$

over $a \in A$ and $f, g \in H$ that satisfy $\delta((a,1), f) = g$

– L_g^h is a star-free context expression describing contexts p over A where $p[x](t)$ takes g to h. This is the union of star-free expressions

$$K_f^h a K_g^{f'}$$

over $a \in A$ and $f, f' \in H$ that satisfy $\delta((a,1), f') = f$.

Once the auxiliary star-free expressions have been found, the star-free expressions defining the automaton $\mathcal{B}$ can be easily obtained by using boolean combinations. The forests that have value (h, G) are described by the star-free expression:

$$L_h \quad \cap \quad \bigcap_{g \in G} K_g \quad \setminus \quad \bigcup_{g \notin G} K_g \ ,$$

while the contexts that take the automaton from value (f, F) to value (h, G) are described by the star-free expression:

$$L_f^h \quad \cap \quad \bigcap_{g \in G} \big(K_f^g \cup \bigcup_{g' \in F} L_{g'}^g\big) \quad \setminus \quad \bigcup_{g \notin G} \big(K_f^g \cup \bigcup_{g' \in F} L_{g'}^g\big) \ .$$

5 Equivalence with chain logic

Proposition 51 Forest languages definable in extended chain logic can be represented by expressions.

The proof is similar to the one in the previous section. We only comment on chain quantification, denoted here by $\exists^c X\ \varphi$; quantification over sibling sets is done in a similar manner.

Similar to the notation in Section 4, if t is a forest over A, and X is a set of nodes in X, then $t[X]$ is the forest over over $A \times \{0,1\}$ obtained from t by adding label 1 on the second coordinate of nodes in X, and label 0 for the remaining nodes. If L is a forest language over $A \times \{0,1\}$, the *chain projection* is defined to be the set of forests t such that $t[X]$ belongs to L for some chain X. The language defined by $\exists^c X \varphi$ is the chain projection of the language defined by φ.

The inductive step in Proposition 51 that goes from φ to $\exists^c \varphi$ will follow once we show that languages represented by expressions are closed under chain projection. The proof of this closure is similar to the one in Section 4. We only sketch a key step, leaving the remaining details to the reader. In the following, $\mathcal{A} = (H, \delta)$ is a forest automaton over the alphabet $A \times \{0,1\}$ that is represented by expressions.

A set of nodes X in a context p is called a p-chain if all of its elements are ancestors of the hole in p. We say a context over A *chains* $g \in H$ *to* $h \in H$ if there is some p-chain X such that the context $p[X]$ takes g to h. The following statement is the key step in showing closure under chain projection, the rest of the proof is left to the reader:

Lemma 1. *For every $g, h \in H$, the set of contexts over A that chains g to h is described by context expression.*

Proof
The proof proceeds in two steps.

In the first step, we write a forest expression for "small" contexts that chain g to h. We will denote this language by K_g^h. A "small" context is of the form:

$$s + a\Box + t \ ,$$

where $a \in A$ is a label and s, t are forests over A. This context chains g to h if and only if for some $i = 0, 1$ we have

$$f + \delta((a, i), g) + f' = h \ , \tag{1}$$

where $f, f' \in H$ are the values under $\mathcal{A}$ of the trees $s[\emptyset], t[\emptyset]$. Recall that $\delta : (A \times \{0, 1\}) \times H \to H$ is the transition function in the automaton A; therefore the parameter i specifies if the chain contains the a node or not. Therefore, the set of "small" contexts that chain g to h is the (finite) union of context expressions

$$L_f + a\Box + L_{f'}$$

such that for some $i = 0, 1$, condition (1) is satisfied. Here L_f (likewise for $L_{f'}$) is the language of forests s such that $s[\emptyset]$ has value f in $\mathcal{A}$; this language has a forest expression by assumption on $\mathcal{A}$ being fully defined by forest expressions.

We now proceed to the second step. We will only describe an expression for contexts where the hole has no siblings; the more general case can be easily obtained using techniques as above. Contexts that chain g to h and have no siblings of the hole can be described by composing small contexts:

$$\bigcup \{K_{f_{k-1}}^{f_k} K_{f_{k-2}}^{f_{k-1}} \cdots K_{f_2}^{f_3} K_{f_1}^{f_2} : g = f_1, \ldots, h = f_k\} \ .$$

Although the above is an infinite expression, it can be easily rendered finite using the Kleene star. □

6 Concatenation hierarchy

In this section, we show that our expressions admit a hierarchy similar to the concatenation (Straubing) hierarchy for word languages. Also similar to the word case, this hierarchy coincides with the quantifier hierarchy of first-order logic with order.

In the context of forest and context expressions, the name concatenation hierarchy would be confusing, since we have two types of concatenation: forest concatenation and context composition. To make things even more confusing, the role of language concatenation in the word concatenation hierarchy is played here by context composition, and not forest concatenation. Therefore, below we refrain from using the name concatenation hierarchy.

The definition below is for a hierarchy of context languages. The hierarchy is extended to forest languages by substituting the empty forest ϵ for the hole: a forest language L is said to be on level n (or $n1/2$) if there is a context language K on level n (or $n1/2$) such that $L = \{p\epsilon : p \in K\}$.

We fix an alphabet A.

- Level 0 of the hierarchy contains two context languages: the set of all contexts over A, and the empty set.
- Level $n1/2$ of the hierarchy is defined as follows.
  1. Every context language on level n is on level $n1/2$.
  2. Context languages on level $n1/2$ are closed under finite union.
  3. If $a \in A$, K is a context language on level $n1/2$ and L is a context language on level n, then LaK is a context language on level $n1/2$.
  4. If L is a forest language of level $n1/2$, and K is a context language of level $n1/2$, then $K + L$ and $L + K$ are context languages of level $n1/2$.
- Level $n + 1$ is the boolean closure of level $n1/2$.

The above definition is the same as for word languages, except for clause 4, which introduces branching into the expressions. Clearly each language in the hierarchy—both for forests and contexts–can be described by a star-free expression. The hierarchy for words can be recovered by taking context languages, and looking at the word that labels the path from root to hole.

Recall that Σ_n is the class of existential first-order formulas—possibly with free variables—whose quantifier prefix has $n - 1$ alternations. For instance, Σ_2 are the formulas with quantifier prefix $\exists^*\forall^*$. Over words, a result of Thomas [4] shows that Σ_n defines the same languages as level $n1/2$ (the signature contains the linear order $<$ on word positions, but not the successor relation $x = y + 1$).

We want to have the same result. However, we need to be careful about the picking the correct signature. We cannot have only the descendant order on tree nodes, since this would give commutative languages; while the forest concatenation $K + L$ gives non-commutative languages already on level $1/2$. Furthermore, the hierarchy is sensitive to slight changes in the signature. For instance, the language "the root has label a" will be on level Σ_1 if the signature contains the "root" unary relation; however if the "root" relation is dropped this language moves up to level Σ_2.

We chose two relations: the lexicographic order $\sqsubseteq$ and the greatest common ancestor $gca(x, y) = z$. The descendant order $x \leq y$ can be defined in terms of gca without using quantifiers, since y is a descendant of x if and only if $gca(x, y) = x$. However, to define gca in terms of the descendant, we need a universal quantifier, to enforce minimality. Therefore, the quantifier alternation

hierarchies are different for the signatures $\{gca, \sqsubseteq\}$ and $\{\leq, \sqsubseteq\}$. From now on, we will be using the signature $\{gca, \sqsubseteq\}$ when talking about the quantifier hierarchy.

We repeat the statement of our result relating the hierarchies in expressions and logic:

Theorem 12
A forest language is on level $n^{1}/_{2}$ if and only if it is defined by a sentence in Σ_n.

We present the two implications in the next two sections. A consequence of the above theorem is that expressions on level $n^{1}/_{2}$ are closed under intersection, which is not immediately apparent from the syntax. Another consequence is Proposition 41, since every first-order formula can be found on some level Σ_n.

It will be convenient to have a notion of formulas describing contexts. Here a formula for contexts has the same syntax as a formula for forests; except that there is a free variable x, which corresponds to the hole. The formula describes those contexts where it holds under the valuation that maps x to the hole. For instance, the formula $\forall y\ x = y$ is true only in the empty context, which maps every forest to itself. The hierarchy Σ_n is defined for context formulas in the same way as for forest formulas. The correspondence in Theorem 12 also extends to contexts.

Level $n^{1}/_{2}$ is definable in Σ_{n+1} The proof is by induction, first on n and then on the size of the expression. The induction base of $n = 0$ is fairly simple, and we omit the details.

The key step is to show how to construct formulas for the steps LaK and $L + K$ in the definition of level $n^{1}/_{2}$.

The step for LaK is a simple relativization technique, and works the same way as for words. Let then $a \in A$, and let L, K be context languages of levels n and $n^{1}/_{2}$ respectively. We will write a formula of Σ_{n+1} that defines the context language LaK. By induction assumption, there are formulas $\varphi(x) \in \Sigma_n, \psi(x) \in \Sigma_{n+1}$ for the languages L, K. The language LaK is defined by the formula

$$\exists y \quad a(y) \ \wedge\ \varphi'(y) \wedge\ \psi'(x) \qquad \in \Sigma_{n+1}\ .$$

Here φ' is obtained from φ by relativizing every quantifier $\exists z$ to $\exists z \not\geq y$; similarly for ψ' and $z > y$. Note that the descendant order $\leq$ is definable in terms of gca without quantifiers, so this relativization does not change the position in the hierarchy.

The case of $L + K$ requires a little more attention. Let then L be a forest language defined by Σ_{n+1} formula φ, and let K be a context language defined by a Σ_{n+1} formula ψ. We will write a formula for $L + K$ (the case of $K + L$ is done the same way), and we also want to use relativization. However, the node that divides L from K must be on the root: we use the leftmost root node x of K here (this node is lexicographically the first one in K). The formula is:

$$\exists x \qquad (\forall y\ gca(x, y) = y \Rightarrow x = y) \ \wedge\ \varphi' \ \wedge\ \psi' \qquad \in \Sigma_{n+1}\ .$$

The first conjunct says that x is one of the roots; the universal quantifier in this conjunct does not increase the hierarchy level of the larger formula, since we are working at or above level Σ_2 (recall that the base case of $n = 0$ was done separately). The relativized formulas are defined as in the previous case, only this time we use the lexicographic order.

Formulas in Σ_{n+1} are captured by level $n+1/2$ As a warm-up, we do the case of $n = 0$. Let then

$$\varphi = \exists x_1, \ldots, x_k \psi$$

be a formula, with φ quantifier-free. By choice of signature, φ can only say what is the lexicographic order on the nodes $x_1, \ldots, x_k$, and which quantified nodes are greatest common ancestors of other quantified nodes. The basic idea is that in a forest of the form $pa(s + t)$, with p a context, a a label, and s, t forests, the greatest common ancestor of a node from s and a node from t must be the node a. The lexicographic order can also be expressed, since the expressions are ordered. We only present the construction on an example:

$$\exists x, y, z \quad a(x) \wedge b(y) \wedge c(z) \wedge gca(x, y) = z$$

Since the labels a, b, c are different, the nodes x, y, z are different. The appropriate expression is:

$$Kc((KaK\epsilon) + (KbK\epsilon)) \qquad \cup \qquad Kc((KbK\epsilon) + (KaK\epsilon)) \ .$$

In the above, K is the level 0 expression describing all contexts.

We would like to continue along these lines for the classes Σ_2, Σ_3, etc. Unfortunately we run across a technical problem, which does not occur for words, and makes the proof rather tedious. If we have nodes $x_1, \ldots, x_k$ in a word w (say, ordered from left to right), then these partition the word into words of the form $w[0..x_1], w[x_1 + 1..x_2], \ldots, w[x_{k-1} + 1..x_k]$. Unfortunately, this is no longer the case for forests.

A first obstacle is how to use two nodes x, y to cut out a piece from a larger forest. If we want to cut out a forest, it would be reasonable to expect the two nodes x, y to be siblings; but the sibling relationship is not quantifier-free in our signature. A second problem is that a given set of nodes may not partition a forest: for this the set must satisfy certain closure properties (such as containing greatest common ancestors).

We write $x \preceq y$ to denote that x is a sibling to the right of y; this formula needs a universal quantifier (to say that all ancestors of x are also ancestors of y) and therefore is a negation of a Σ_1 formula. Note that this is *not* the lexicographic order. A formula $x \preceq y$ is called a *forest link*. The ideas that the nodes x, y can be used to *induce* a subforest. If $x = y$ then the induced subforest contains proper descendants z of x; if $x \prec y$ then the induced subforest contains nodes z that lexicographically between x, y, but are not descendants of x. A formula $x < y$ is called a *context link*. Similarly to forest links, the nodes x, y

induce a context: the nodes of this context are nodes z with $x < z$ and $y \not\leq z$, and the hole is in y. We will use letters α, β to refer to links, be they context or forest. In either case, the property that z belongs to to the induced forest/context of a link α is expressed by a quantifier-free formula in x, y, z.

Types

Let X be set of variables. A *pre-type over* X is a conjunction, which for every variables $x, y, z \in X$ specifies the label of this variable, and which of the following hold, and which do not hold: x is a leftmost sibling, x is a rightmost sibling, $x \leq y$, $x \preceq y$, x is a child of y, x is the left neighbor of y, z is the greatest common ancestor of x, y. Since a pre-type specifies all this information, it makes sense to write "a node $x \in \tau$ has a child in τ"—likewise for the other relations described in a pre-type—with the meaning being that $x \in X$ and for some node $y \in X$, the pre-type τ specifies that y is the child of x. Note that a pre-type is a boolean combination of Σ_1 formulas, since universal quantification is necessary to talk about left/right siblings, etc.

Take a link α—either a forest link $x \preceq y$ or a context link $x < y$. A *type local to* α is a type $\tau \ni x, y$ where every variable $z \in \tau$ distinct from x, y is in the (forest/context) induced by α. Furthermore, we require the saturation properties listed below; these ensure that the nodes induced by α can be partitioned into links (both context and forest) given by variables from τ.

- If $z_1, z_2 \in \tau$ are distinct siblings, but not siblings of x and y (possible only when α is a forest link), then the parent of z_1, z_2 belongs to τ.
- If $z \in \tau$ has its parent in τ, then its leftmost, rightmost siblings are in τ.
- If $z_1, z_2 \in \tau$ are not related by the descendant relation, then there are siblings $z_1', z_2' \in \tau$ that are ancestors of z_1, z_2, respectively.

A consequence of the properties above is that nodes described by a type τ are closed under greatest common ancestor, as long as this greatest common ancestor stays within the link α.

Below we define the *joints* of type τ. The idea is that these are context/forest links that are closest to each other and contain some nodes in their induced forest/context.

- Let $x, y \in \tau$ be such that y is a descendant of x, and minimal for this property. If y is not a child of x, then $x < y$ is called a *(context) joint of* τ.
- There are two kinds of *forest joints*. If $x, y \in \tau$ are distinct siblings, with no sibling from τ strictly between them, then $x \prec y$ is a forest joint of τ. Also, if $x \in \tau$ has no proper descendants in τ, then $x \preceq x$ is a joint of τ.

The saturation properties in the definition of a type are chosen so that the joints partition a type, i.e.:

Lemma 2. *Let τ be a type local to a link α. The type τ entails that any node in the (context/forest) induced by α is either one of the nodes from τ, or in exactly one (context/forest) induced by a joint of τ.*

Proof
Using the saturation properties. We consider here the case when $\alpha = x \preceq y$. Let then z be a node not in the forest induced by the link $x \preceq y$. Assume furthermore that z is not equal to one of the nodes from τ. Consider two cases:

- There are siblings of z in τ. We claim that there are nodes $z_1, z_2 \in \tau$ with $z_1 \prec z \prec z_2$. If $x \prec z \prec y$ then we are done. Otherwise, since z has siblings in τ, then the parent of z belongs to τ thanks to the saturation properties. This entails that also the leftmost and rightmost siblings of z belong to τ. Let then $z_1, z_2 \in \tau$ be such that $z_1 \prec z \prec z_2$, and chosen closest to z for this property. Therefore, $z_1 \preceq z_2$ is a joint of τ, and its induced forest contains z.
- There are no siblings of z in τ. First we claim that z has an ancestor in τ. If x is an ancestor of z, we are done. Otherwise, x and z are incomparable by the descendant relation, and therefore by the saturation properties, τ must contain siblings that are ancestors of x, z, which concludes the claim. Let z' be the ancestor in τ that is closest to z. If z has a descendant z'' in τ, then z belongs to the context induced by $z' < z''$, if z'' is taken closest to z. Otherwise, we show that z' has no descendants in τ, and hence z belongs to the forest induced by the link $z' \preceq z'$. Assume for the sake of contradiction that τ contains a descendant z'' of z' that is not a descendant of z. By definition of z', z'' must be incomparable with z. Using the last saturation property, we get a node in τ that is strictly between z' and z, a contradiction.

□

Let X be a set of variables, and let α be a link (using variables from X). A type *local to α generated by X* is defined to be any type τ local to α over variables $Y \supseteq X$ that is minimal for this property: i.e. there is no type σ local to α over variables Z, with $X \subseteq Z \subsetneq Y$ such that τ and σ are consistent. If τ is generated, then $Y \setminus X$ is called the set of *auxiliary* variables of τ, and is denoted by Y_τ. The idea is that the variables in Y_τ are used to add the saturation properties required in a type. Since the saturation properties require adding at most a linear number of additional nodes, there are finitely many nonequivalent types generated by a given set X and a link α; furthermore any two nonequivalent ones are inconsistent.

Normal form

A formula is said to be *local* to a link α if its quantified and free variables are relativized to nodes induced by the link α. Let α be a link. We define two types of normal form: $(n-1/2, \alpha)$-normal form, which corresponds to Σ_n formulas local to α, and (n, α)-normal form, which corresponds to boolean combinations of the former. We define $(0, \alpha)$-normal form to be all quantifier-free formulas local to α. For $n \geq 1$, a formula is said to be in $(n-1/2, \alpha)$ *normal form* if it is a positive boolean combination of formulas $\exists x_1, \ldots, x_k \psi$, with ψ in $(n-1, \alpha)$ normal form. A formula—with free variables X—is in (n, α)-*normal form* if it is of the form $\exists y_1, \ldots, y_k \ \tau \wedge \psi$, where

- τ is a type local to α generated by X with auxiliary variables $y_1, \ldots, y_k$. To abbreviate the notation, we write $\exists y_1, \ldots, y_k$ as $\exists Y_\tau$.

– For some joint β of τ, the formula ψ is either in $(n - 1/2, \beta)$-normal form, or a negation thereof.

Lemma 3. *For $n \geq 1$, every formula in Σ_n local to α is equivalent to a formula in $(n - 1/2, \alpha)$ normal form.*

Proof
The proof is a by induction on n. The only nontrivial part is showing that positive boolean combinations of formulas in (n, α)-normal form are closed under negation. By De Morgan laws, we only need to do the negation of a single formula:

$$\neg(\exists Y_\tau\ \tau \wedge \psi) \qquad \Longleftrightarrow \qquad (\exists Y_\tau\ \tau \wedge \neg\psi)\ \vee\ \bigvee_\sigma (\exists Y_\sigma\ \sigma \wedge \mathit{true})\ ,$$

where the disjunction above ranges over the—finitely many—types σ generated by X that are inconsistent with τ.

The case of existential quantification, which corresponds to going from (n, α) normal form to $(n + 1/2, \alpha)$-normal form, is a straightforward consequence of Lemma 2. □

From normal form to expressions

We will now rewrite a formula in $(n1/2, \alpha)$ normal form into an expression on level $n1/2$. The expression will be a context or forest expression, depending on the kind of link α. It will be convenient to use an intermediate form, where logical formulas and regular expressions can be mixed. For this purpose we extend first-order logic with the following type of formulas as follows. Let $\alpha = x < y$ be a context link. If K is a context expression, then $K_\alpha(x, y)$ is a formula. This formula holds if K contains the context induced by α. The definition for a forest link $x \preceq y$ and a forest expression L is analogous. Together with Lemma 3, the following result concludes the "if" part of Theorem 12.

Lemma 4. *Let $n \geq 0$. Let $\alpha = x < y$ be a context link, and let φ be a formula in $(n1/2, \alpha)$ normal form. There is a context expression K of level $n1/2$ such that φ and $K_\alpha(x, y)$ are equivalent. Likewise for forests.*

Proof (Sketch)
Induction on n, with plenty case analysis. The base case $n = 0$ is done as in the beginning of this section. We only do the induction step for a context link $x < y$, leaving the proof for forests to the reader. Let then $n \geq 1$. Since the expressions are closed under union, and types are either equal or inconsistent, we only need consider a formula φ be of the form:

$$\varphi = \exists x_1, \ldots, x_k\ \tau\ \wedge\ \psi\ ,$$

with τ a type over variables $x_1, \ldots, x_k, x, y$ and ψ a conjunction of formulas in $(n - 1/2, \beta)$ normal form (or their negations), with β ranging over joints of τ.

Using the induction assumption, we may replace ψ be a conjunction $\bigwedge_\beta K_\beta$, where β ranges over joints of τ, and each K_β is a forest or context expression

of level n, depending on the kind of β. (Recall that expressions of level n are closed under boolean combinations, so we need not bother with the negations in conjuncts in ψ.)

The proof then proceeds via a second induction, this time on the number of variables k. Recall the link $\alpha = x < y$ to which the formula φ is local. The proof is a case analysis. We only consider here the case when τ contains the parent of y, but not the grandparent of y. Let then $x_1 \in \tau$ be the parent of y, and let $x_2 < x_1$ be the closest ancestor of x_1 that is in τ. We leave to the reader the special cases when $x_1 = x$, and therefore x_2 is undefined, and also when y is either a leftmost or rightmost sibling. Let then

$$z_1 \prec \cdots \prec z_i \prec y \prec z'_1 \prec \cdots \prec z'_j \tag{2}$$

be all the distinct siblings of y given in the type τ. Using the induction assumption, we may assume that $i, j = 1$, and that the links $z_1 \preceq z_1$, $z_1 \prec y$, $y \prec z'_1$, and $z'_1 \preceq z'_1$ are described by expressions L, M, L', M' respectively. Finally, let K be the context expression describing the joint $x_2 < x_1$, and let a be the label of the node x_1, which can be read from τ. We can now lose all the variables from (2), by describing the context link $x_2 < y$ via the expression

$$Ka(L + M + \Box + L' + M') \ .$$

The above is a level $n1/2$ expression, since $n \geq 1$ and $\Box$—an expression that describes only the empty context $\Box$—is a level $1\frac{1}{2}$ context expression. $\Box$

References

1. M. Bojańczyk and I. Walukiewicz. Forest algebras (unpublished manuscript). http://hal.archives-ouvertes.fr/ccsd-00105796.
2. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2002. release October, 1rst 2002.
3. R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
4. W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25:360–375, 1982.
5. W. Thomas. Logical aspects in the study of tree languages. In *Colloquium on Trees and Algebra in Programming*, pages 31–50, 1984.
6. W. Thomas. On chain logic, path logic, and first-order logic over infinite trees. In *Logic in Computer Science*, pages 245–256, 1987.