Marrying Words and Trees

Rajeev Alur
University of Pennsylvania

ABSTRACT

Traditionally, data that has both linear and hierarchical structure, such as annotated linguistic data, is modeled using ordered trees and queried using tree automata. In this paper, we argue that nested words and automata over nested words offer a better way to capture and process the dual structure. Nested words generalize both words and ordered trees, and allow both word and tree operations. We study various classes of automata over nested words, and show that while they enjoy expressiveness and succinctness benefits over word and tree automata, their analysis complexity and closure properties are analogous to the corresponding word and tree special cases. In particular, we show that finite-state nested word automata can be exponentially more succinct than tree automata, and pushdown nested word automata include the two incomparable classes of context-free word languages and context-free tree languages.

Categories and Subject Descriptors

F.4.3 [**Theory of computation**]: Formal languages; H.2.3 [**Database management**]: Query languages

General Terms

Theory

Keywords

Tree automata, Pushdown automata, Nested words, XML, Query languages

1. INTRODUCTION

Linearly structured data is usually modeled as words, and queried using word automata and related specification languages such as regular expressions. In many applications including executions of structured programs, annotated linguistic data, and primary/secondary bonds in genomic sequences, the data has a hierarchical structure in addition to the linear order. For example, in natural language processing, the sentence is a linear sequence of words, and parsing into syntactic categories imparts the hierarchical structure. Even though the only logical structure on data is hierarchical, sometimes linear sequencing is added either for storage or for stream processing. For example, in SAX representation of XML data, the document is a linear sequence of text characters, along with a hierarchically nested matching of open-tags with closing tags.

Data with dual linear-hierarchical structure is traditionally modeled using binary, and more generally, using ordered unranked, trees, and queried using tree automata (see [17, 13, 18] for recent surveys on applications of unranked trees and tree automata to XML processing). In ordered trees, nodes with the same parent are linearly ordered, and the classical tree traversals such as infix (or depth-first left-to-right) can be used to define an implicit ordering of all nodes. While ordering only the siblings is natural in many database applications (indeed, this is the DOM representation of XML), the lack of explcit ordering of all nodes do have some consequences. First, tree-based approach implicitly assumes that the input linear data can be parsed into a tree, and thus, one cannot represent and process data that may not parse correctly. Word operations such as prefixes, suffixes, and concatenation, while natural for document processing, do not have analogous tree operations. Second, tree automata can naturally express constraints on the sequence of labels along a hierarchical path, and also along the left-to-right siblings, but they have difficulty to capture constraints that refer to the global linear order. For example, the query that patterns $p_1, \dots p_n$ appear in the document in that order (that is, the regular expression $\Sigma^* p_1 \Sigma^* \dots p_n \Sigma^*$ over the linear order) compiles into a deterministic word automaton of linear size, but standard deterministic bottom-up tree automaton for this query must be of size exponential in n. This exponential gap is unsurprising in retrospect since complexity of a bottom-up tree automaton is related to the index of the congruence induced by the query, while complexity of a word automaton is related to the index of the right-congruence induced by the query. This deficiency shows up more dramatically if we consider pushdown acceptors: a query such as "the document contains an equal number of occurrences of patterns p and q" is a context-free word language but is not a context-free tree language.

In this paper, we show that the model of *nested words*, recently proposed in the context of specification and verification of structured programs [4], allows a better integration of the two orderings. A nested word consists of a sequence of linearly ordered positions, augmented with hierarchical edges connecting calls to returns (or open-tags to close-tags). The edges create a properly nested hierarchical structure, while allowing some of the edges to be pending. This nesting structure can be uniquely represented by a sequence specifying the types of positions (calls, returns, and internals). Words are nested words where all positions are internals. Ordered trees can be interpreted as nested words using the following traversal: to process an a-labeled node, first print an a-labeled call, process all children in order, and print an a-labeled return. Note that this is a combination of top-down and bottom-up traversals, and each node is processed twice. Word operations such as prefixes, suffixes, concatenation, reversal, as well as tree operations, can be defined easily on nested words. Binary trees, ranked trees, unranked trees, forests, and documents that do not parse correctly, all can be represented with equal ease. Finally, since the SAX representation of XML documents already contain tags that specify the position type, they can be interpreted as nested words without any preprocessing.

A nested-word automaton is similar to a classical finite-state word automaton, and reads the input from left to right according to the linear sequence, processing each symbol based to its type [4]. At a call, it can propagate states along both linear and hierarchical outgoing edges, and at a return, the new state is determined based on states labeling both the linear and hierarchical incoming edges. The resulting class of *regular* languages of nested words seems to have all the appealing theoretical properties that the classes of classical regular word and tree languages enjoy. Deterministic nested word automata are as expressive as their nondeterministic counterparts. The class is closed under boolean operations as well as a variety of word and tree operations. Decision problems such as membership, emptiness, language inclusion, and language equivalence are all decidable, typically with the same complexity as the corresponding problem for tree automata. Finally, the notion of regularity can be characterized in multiple equivalent ways, and in particular, using definability in monadic second order logic.

In order to study the relationship of nested word automata to various kinds of word and tree automata, we define restricted classes of nested word automata and study the impact of these restrictions on expressiveness and succinctness. *Flat* automata do not propagate information along the hierarchical edges at calls, and correspond to classical word automata accepting the weaker class of regular word languages. *Bottom-up* automata, on the other hand, do not propagate information along the linear edges at calls. Over the subclass of nested words corresponding to ordered trees, these automata correspond exactly to bottom-up tree automata for binary trees and stepwise bottom-up tree automata [5] for unranked trees. We show that there is an exponential price in terms of succinctness due to this restriction. Our definition of *joinless* automata avoids a nontrivial join of information along the linear and hierarchical edges at returns, and this concept is a generalization of the classical top-down tree automata. While deterministic joinless automata are strictly less expressive, nondeterministic ones can accept all regular languages of nested words. The succinctness gap between nested word automata and traditional tree automata holds even if we restrict attention to paths (that is, unary trees): nested word automata are exponentially more succinct than both bottom-up and top-down automata.

We also introduce and study *pushdown nested word automata* by adding a stack to the finite-state control of nondeterministic joinless automata. We show that both pushdown word automata and pushdown tree automata are special cases, but pushdown nested word automata are strictly more expressive than both. In terms of complexity of analysis problems, they are similar to pushdown tree automata: membership is NP-complete and emptiness is EXPTIME-complete. Our decision procedure for emptiness generalizes the corresponding checks for pushdown word automata and for pushdown tree automata.

Related Work

Languages of words with well-bracketed structure have been studied as Dyck languages and *parenthesis* languages, and shown to have some special properties compared to context-free languages (for example, decidable equivalence problem) [16, 9]. The new insight is that the matching among left and right parantheses can be considered to be an explicit component of the input structure, and this leads to a robust notion of regular languages using finite-state acceptors. This was first captured using *visibly pushdown automata* [3], and later, a cleaner and improved reformulation using nested word automata [4]. There is a lot of recent work on visibly pushdown automata and/or nested word automata (see [14, 2, 1, 10]). However, none of this addresses succinctness compared to tree automata. Recent work proposes the notion of XVPA for processing XML schema [11]. These automata, however, are a particular form of bottom-up automata studied in this paper, and thus, do not process linear structure well.

There is a rich literature on tree automata, and we used [18, 6] for our research. Besides classical top-down and bottom-up automata over binary trees, stepwise bottom-up tree automata for processing unranked ordered trees [15, 5] and pushdown tree automata [8, 12] are the most relevant to this paper. Deterministic word automata have been also used for stream processing of XML documents [7], where the authors argue, with experimental supporting data, that finite-state word automata may be good enough given that hierarchical depth of documents is small.

2. LINEAR HIERARCHICAL MODELS

2.1 Nested Words

Given a linear sequence, we add hierarchical structure using edges that are well nested (that is, they do not cross). We will use edges starting at $-\infty$ and edges ending at $+\infty$ to model "pending" edges. Assume that $-\infty < i < +\infty$ for every integer i.

A *matching relation* $\rightsquigarrow$ of length ℓ, for $\ell \geq 0$, is a subset of $\{-\infty, 1, 2, \dots \ell\} \times \{1, 2, \dots \ell, +\infty\}$ such that

1. if $i \rightsquigarrow j$ then $i < j$;
2. if $i \rightsquigarrow j$ and $i \rightsquigarrow j'$ and $i \neq -\infty$ then $j = j'$, and if $i \rightsquigarrow j$ and $i' \rightsquigarrow j$ and $j \neq +\infty$ then $i = i'$
3. if $i \rightsquigarrow j$ and $i' \rightsquigarrow j'$ then it is not the case that $i < i' \leq j < j'$

When $i \rightsquigarrow j$ holds for $1 \leq i < j \leq \ell$, the position i is a *call* with the position j being its *return-successor*, and the position j is a *return* with i being its *call-predecessor*. When $-\infty \rightsquigarrow j$ holds, the position j is a return, but without

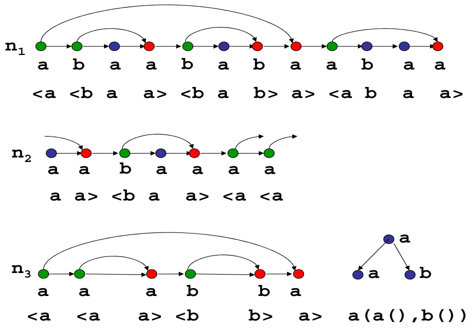

Figure 1: Sample nested words

a matching call, and when $i \rightsquigarrow +\infty$ holds, the position i is a call, but without a matching return. Our definition requires that a position has at most one return-successor and at most one call-predecessor, and a position cannot be both a call and a return. A position i that is neither a call or a return is called *internal*.

A *nested word* n over an alphabet Σ is a pair $(a_1 \ldots a_\ell, \rightsquigarrow)$, for $\ell \geq 0$, such that a_i, for each $1 \leq i \leq \ell$, is a symbol in Σ, and $\rightsquigarrow$ is a matching relation of length ℓ. Let us denote the set of nested words over Σ as $NW(\Sigma)$. A language of nested words over Σ is a subset of $NW(\Sigma)$.

A nested word n with matching relation $\rightsquigarrow$ is said to be *well-matched* if there is no position i such that $-\infty \rightsquigarrow i$ or $i \rightsquigarrow +\infty$. Thus, in a well-matched nested word, every call has a return-successor and every return has a call-predecessor. We will use $WNW(\Sigma) \subseteq NW(\Sigma)$ to denote the set of all well-matched nested words over Σ. A nested word n of length ℓ is said to be *rooted* if $1 \rightsquigarrow \ell$ holds. Observe that a rooted word must be well-matched.

While the length of a nested word captures its linear complexity, its (nesting) depth captures its hierarchical complexity. The *depth* of a nested word is the maximum number d such that there exist call positions $i_1, \ldots i_d$ with corresponding return-successors $j_1, \ldots j_d$ such that $i_1 < i_2 < \cdots < i_d < j_d \cdots < j_2 < j_1$.

It is convenient to define a notion of *call-parent* for all positions. Let n be a nested word of length ℓ. The call-parent of position 1 is defined to be 0. For $1 \leq i < \ell$, if i is a call then the call-parent of $i+1$ is i; if i is an internal then the call-parent of $i+1$ is same as the call-parent of i; if i is a return such that $j \rightsquigarrow i$, then if $j = -\infty$ then call-parent of $i+1$ is 0, otherwise it is same as the call-parent of j. Thus, if i is at top-level then its call-parent is 0 otherwise it is the smallest call position whose return-successor is after i.

Figure 1 shows three nested words over the alphabet $\{a, b\}$. The nested word n_1 is well-matched, and has length 12 and depth 2. The nested word n_2 has an unmatched return and two unmatched calls, and the nested word n_3 is rooted.

2.2 Relation to Words

Nested words over Σ can be encoded by words in a natural way by using the tags $\langle$ and $\rangle$ to denote calls and returns, respectively. We assume that $\langle$ and $\rangle$ are special symbols that do not appear in the alphabet Σ. Then, given an alphabet Σ, define the *tagged alphabet* $\hat{\Sigma}$ to be the set that contains the symbols $\langle a$, a, and $a\rangle$ for each $a \in \Sigma$. Formally, we define the mapping $nw\_w : NW(\Sigma) \mapsto \hat{\Sigma}^*$ as follows: given a nested word $n = (a_1, \ldots a_\ell, \rightsquigarrow)$ of length ℓ over Σ, $nw\_w(n)$ is a word $b_1, \ldots b_\ell$ over $\hat{\Sigma}$ such that for each $1 \leq i \leq \ell$, $b_i = a_i$ if i is an internal, $b_i = \langle a_i$ if i is a call, and $b_i = a_i\rangle$ if i is a return.

Figure 1 also shows the tagged words corresponding to each nested word. Since we allow unmatched calls and returns, every word over the tagged alphabet $\hat{\Sigma}$ corresponds to a nested word: the transformation $nw\_w : NW(\Sigma) \mapsto W(\hat{\Sigma})$ is a bijection. The inverse of nw_w is a transformation function that maps words over $\hat{\Sigma}$ to nested words, and will be denoted $w\_nw : W(\hat{\Sigma}) \mapsto NW(\Sigma)$. This one-to-one correspondence shows that there are exactly 3^ℓ distinct matching relations of length ℓ, and the number of nested words of length ℓ is $3^\ell|\Sigma|^\ell$. Observe that if w is a word over Σ, then $w\_nw(w)$ is the corresponding nested word with the empty matching relation.

We will also consider a mapping that views a word as a sequence of symbols along a hierarchical path. More precisely, consider the transformation function $path : \Sigma^* \mapsto NW(\Sigma)$ such that $path(a_1 \ldots a_\ell)$ is $w\_nw(\langle a_1 \ldots \langle a_\ell a_\ell\rangle \ldots a_1\rangle)$. Note that for a word w, $path(w)$ is rooted and has depth $|w|$.

2.3 Relation to Ordered Trees

Ordered trees can be interpreted as nested words. In this representation, it does not really matter whether the tree is binary, ranked, or unranked.

The set $OT(\Sigma)$ of ordered trees over an alphabet Σ is defined inductively:

1. $\varepsilon \in OT(\Sigma)$: this is the empty tree;
2. if $t_1, \ldots t_n \in OT(\Sigma)$, for $n \geq 0$, with each $t_i \neq \varepsilon$, and $a \in \Sigma$, then $a(t_1, \ldots t_n) \in OT(\Sigma)$: this represents the tree whose root is labeled a, and has n children $t_1 \ldots t_n$ in that order.

Consider the transformation $t\_w : OT(\Sigma) \mapsto W(\hat{\Sigma})$ that encodes an ordered tree over Σ as a word over $\hat{\Sigma}$: $t\_w(\varepsilon) = \varepsilon$; and $t\_w(a(t_1, \ldots t_n)) = \langle a\, t\_w(t_1) \cdots t\_w(t_n)\, a\rangle$. This transformation can be viewed as a traversal of the tree, where processing an a-labeled node corresponds to first printing an a-labeled call, followed by processing all the children in order, and then printing an a-labeled return. Note that each node is visited and copied twice. An a-labeled leaf corresponds to the word $\langle aa\rangle$, we will use $\langle a\rangle$ as its abbreviation.

The transformation $t\_nw : OT(\Sigma) \mapsto NW(\Sigma)$ is the functional composition of t_w and w_nw. However, not all nested words correspond to ordered trees. A nested word $n = (a_1 \ldots a_\ell, \rightsquigarrow)$ is said to be a *tree word* if (1) it is rooted, (2) has no internals, and (3) for all $i \rightsquigarrow j$, $a_i = a_j$. The first condition ensures that the nested word is well-matched and has a single root. The latter two conditions ensure one-to-one correspondence between matching calls and returns. Let $TW(\Sigma) \subseteq NW(\Sigma)$ denote the set of all tree words over Σ. Then, the transformation $t\_nw : OT(\Sigma) \mapsto NW(\Sigma)$ is a bijection between $OT(\Sigma)$ and $TW(\Sigma)$. The inverse of t_nw then is a transformation function that maps tree words to ordered trees, and will be denoted $nw\_t : TW(\Sigma) \mapsto OT(\Sigma)$.

In Figure 1, only the nested word n_3 is a tree word, and corresponds to the binary tree shown there.

2.4 Operations on Nested Words

Due to the correspondence between nested words and tagged words, every classical operation on words and languages of nested words can be defined for nested words and languages of nested words. Operations on ordered trees and tree languages can be lifted to nested words and their languages. We list a few operations below.

Concatenation: Concatenation of two nested words n and n' is the nested word $w\_nw(nw\_w(n)nw\_w(n'))$. Notice that the matching relation of the concatenation can connect unmatched calls of the first with the unmatched returns of the latter.

Subwords, prefixes, and suffixes: Given a nested word $n = w\_nw(b_1 \ldots b_\ell)$, its *subword* from position i to j, denoted $n[i,j]$, is the nested word $w\_nw(b_i \ldots b_j)$, provided $1 \leq i \leq j \leq \ell$, and the empty nested-word otherwise. Note that if $i \rightsquigarrow j$ in a nested word, then in the subword that starts before i and ends before j, this hierarchical edge will change to $i \rightsquigarrow +\infty$; and in the subword that starts after i and ends after j, this hierarchical edge will change to $-\infty \rightsquigarrow j$; Subwords of the form $n[1,j]$ are prefixes of n and subwords of the form $n[i,\ell]$ are suffixes of n. Note that for $1 \leq i \leq \ell$, concatenting the prefix $n[1,i]$ and the suffix $n[i+1,\ell]$ gives back n.

Reverse: Reverse of a nested word n is defined to be $w\_nw(b_\ell \ldots b_2 b_1)$, where for each $1 \leq i \leq \ell$, $b_i = a_i$ if i is an internal, $b_i = \langle a_i$ if i is a return, and $b_i = a_i\rangle$ if i is a call. That is, to reverse a nested word, we reverse the underlying word as well as all the hierarchical edges.

Insertion: If n is a nested word, $a \in \Sigma$, and n' is a well-matched nested word, then the result of inserting n' into n after every a-labeled position, written $Insert(n, a, n')$ is defined to be n if $a_i \neq a$ for all $1 \leq i \leq \ell$, and the concatenation of $n[1,i]$ and n' and $Insert(n[i+1,\ell], a, n')$ if $a_i = a$ such that $a_j \neq a$, for $1 \leq j < i$. Note that insertion of a tree word into another tree word is same as tree insertion. Other tree operations such as subtree deletion and substitution can be defined similarly.

3. FINITE-STATE AUTOMATA

3.1 Nested Word Automata

Now we define finite-state acceptors over nested words that can process both linear and hierarchical structure.

A *nested word automaton* (NWA) A over an alphabet Σ is a structure $(Q, q_0, F, \delta_c, \delta_i, \delta_r)$ consisting of

- a finite set Q of states,
- an initial state $q_0 \in Q$,
- a set of final states $F \subseteq Q$,
- a call-transition function $\delta_c : Q \times \Sigma \mapsto Q \times Q$,
- an internal-transition function $\delta_i : Q \times \Sigma \mapsto Q$, and
- a return-transition function $\delta_r : Q \times Q \times \Sigma \mapsto Q$.

The automaton A starts in the initial state, and reads the nested word from left to right. The state is propagated along the linear edges as in case of a standard word automaton. However, at a call, the nested word automaton can propagate a state along the outgoing hierarchical edge also. At a return, the new state is determined based on the states propagated along the linear as well as the hierarchical incoming edges. Formally, a *run* r of the automaton A over a nested word $n = (a_1 \ldots a_\ell, \rightsquigarrow)$ is a sequence $q_0, \ldots, q_\ell$ of states corresponding to linear edges, and a sequence q_{ij}, for $i \rightsquigarrow j$, of states corresponding to hierarchical edges, such that for each position $1 \leq i \leq \ell$,

- if i is a call with $i \rightsquigarrow j$, then $\delta_c(q_{i-1}, a_i) = (q_i, q_{ij})$;
- if i is an internal, then $\delta_i(q_{i-1}, a_i) = q_i$;
- if i is a return such that $j \rightsquigarrow i$, then $\delta_r(q_{i-1}, q_{ji}, a_i) = q_i$, where if $j = -\infty$ then $q_{ji} = q_0$.

One can view nested word automata as graph automata over the acyclic graph of linear and hierarchical edges (see [19]): a run is a labeling of the edges such that the states on the outgoing edges at a node are determined by the states on the incoming edges and the symbol labeling the node. For the hierarchical edges of the form $-\infty \rightsquigarrow i$, the corresponding state is the initial state q_0. Verify that for a given nested word n, the automaton has precisely one run over n. The automaton A accepts the nested word n if in this run, $q_\ell \in F$. The language $L(A)$ of a nested-word automaton A is the set of nested words it accepts.

There is a close similarity to tree automata: at calls, the automaton behaves like a top-down tree automaton forking states, and at returns, it acts like a bottom-up tree automaton joining states.

3.2 Regular Languages

A language L of nested words over Σ is *regular* if there exists a nested word automaton A over Σ such that $L = L(A)$. We recall the main properties of this class [4].

Closure: The class of regular languages of nested words is (effectively) closed under union, intersection, complementation, concatenation, and Kleene-$*$. If L is a regular language of nested words then all the following languages are regular: the set of all prefixes of all the words in L; the set of all suffixes of all the words in L; the set of reversals of all the words in L.

Regular languages are closed under tree-like operations that use hierarchical structure. For example, if L and L' are regular languages of nested words then the set of all nested words of the form $Insert(n, a, n')$ for $n \in L$ and $n' \in L'$, is a regular language.

Logic-based characterization: The classical correspondence between monadic second order logic and finite recognizability for words and trees continues to hold for nested words. The *monadic second-order logic of nested words* is given by the syntax:

$$\phi := Q_a(x) \mid x \leq y \mid x \rightsquigarrow y \mid \phi \vee \phi \mid \neg\phi \mid \exists x.\phi \mid \exists X.\phi,$$

where $a \in \Sigma$, x, y are first-order variables, and X is a second order variable. The semantics is defined over nested words in a natural way. A language L of nested words over Σ is regular iff there is an MSO sentence ϕ over Σ such that L is the set of all nested words that satisfy ϕ.

Nondeterministic Automata: A nondeterministic NWA A has finite set Q of states, initial states $Q_0 \subseteq Q$, a set $F \subseteq Q$ of final states, a call-transition relation $\delta_c \subseteq Q \times \Sigma \times Q \times Q$, an internal-transition relation $\delta_i \subseteq Q \times \Sigma \times Q$, and a return-transition relation $\delta_r \subseteq Q \times Q \times \Sigma \times Q$. The notion of a run over a nested word and the language $L(A)$ is defined in the obvious way. Nondeterministic nested word automata are no more expressive than the deterministic ones: given a nondeterministic automaton A with s states, one can effectively construct a deterministic NWA B with 2^{s^2} states such that $L(B) = L(A)$.

Decision Problems: Given a nested word automaton A and a nested word n, the membership problem (is n in $L(A)$?) can be solved in linear time. The space required is proportional to the depth of n since one needs to remember the labeling of pending hierarchical edges at every position. If A is nondeterministic, membership problem can be solved in time $O(|A|^3\ell)$ using dynamic programming similar to the one used for membership for pushdown word automata.

The emptiness problem for nested word automata(is $L(A)$ empty?) can be solved in cubic time using techniques similar to the ones used for pushdown word automata or tree automata.

Problems such as language inclusion and language equivalence are decidable. These problems can be solved using constructions for complementation and language intersection, and emptiness test. If one of the automata is nondeterministic, then this would require determinization, and both language inclusion and equivalence are EXPTIME-complete for nondeterministic NWAs.

Weak Automata: Note that the call-transition function δ_c of a nested word automaton A has two components that specify, respectively, the states to be propagated along the linear and the hierarchical edges. We will refer to these two components as δ_c^l and δ_c^h. That is, $\delta_c(q,a) = (\delta_c^l(q,a), \delta_c^h(q,a))$. In terms of expressiveness, it suffices if the hierarchical component simply propagates the current state. A nested word automaton A with call-transition function δ_c is said to be *weak* if for all states q and symbols a, $\delta_c^h(q,a) = q$. Weak NWAs can capture all regular languages:

THEOREM 1. *Given a nested word automaton A with s states over Σ, one can effectively construct a weak NWA B with $s|\Sigma|$ states such that $L(B) = L(A)$.*

PROOF. Consider an NWA $A = (Q, q_0, F, \delta_c, \delta_i, \delta_r)$. The weak automaton B remembers, in addition to the state of A, the symbol labeling the call-parent of the current position so that it can be retrieved at a return and the hierarchical component of the call-transition function of A can be applied. The desired automaton B is $(Q \times \Sigma, (q_0, a_0), F \times \Sigma, \delta_c', \delta_i', \delta_r')$ (here a_0 is some arbitrarily chosen symbol in Σ). The transitions are defined by: $\delta_i'((q,a),b) = (\delta_i(q,b),a)$; $\delta_c'((q,a),b) = ((\delta_c^l(q,b),b),(q,a))$; and $\delta_r'((q,a),(q',b),c) = (\delta_r(q,\delta_c^h(q',a),c),b)$. $\square$

3.3 Relation to Word Languages

A nested word automaton $A = (Q, q_0, F, \delta_c, \delta_i, \delta_r)$ is said to be *flat* if for all $a \in \Sigma$ and $q \in Q$, $\delta_c^h(q,a) = q_0$. Thus, in a run of a flat automaton over a nested word, all the hierarchical edges are labeled with the initial state, and hence, there is no information propagated across these edges to the returns. Consequently, a flat NWA is equivalent to a classical finite-state word automaton. The only difference is that such an automaton updates its state not only based on its current state and the symbol being read, but also based on whether the current position is a call, internal, or return. Conversely, a classical word automaton $A = (Q, q_0, F, \delta)$ over $\hat{\Sigma}$ can be interpreted as a flat NWA $A' = (Q, q_0, F, \delta_c, \delta_i, \delta_r)$, where for every $q, q' \in Q$ and $a \in \Sigma$, $\delta_c(q,a) = (\delta(q,\langle a), q_0)$, $\delta_i(q,a) = \delta(q,a)$, and $\delta_r(q,q',a) = \delta(q,a\rangle)$. This is summarized in the following:

THEOREM 2. *A nested word language L over Σ is accepted by a flat nested word automaton with s states iff the corresponding language $nw\_w(L)$ of tagged words over $\hat{\Sigma}$ is accepted by a deterministic word automaton with s states.*

In general, due to the ability to pass information across hierarchical edges, for a regular language L nested words, $nw\_w(L)$ need not be a regular word language. In particular, the set $nw\_w(WNW(\Sigma))$ is not regular. One can interpret the nested word automaton as a pushdown word automaton that is required to push while reading a call and pop while reading a return. The height of the stack is determined by the input word, and equals the depth of the prefix read. Such restricted form of pushdown automata are called *visibly pushdown automata* [3].

When a language L of nested words is accepted by a flat NWA, then using the classical algorithms for minimizing deterministic word automata, one can construct a minimal (in terms of number of states) flat NWA accepting L. However, such minimal automata can be exponentially larger than NWAs that use the hierarchical edges for information propagation. This exponential price in succinctness is established by the following theorem.

THEOREM 3. *There exists a family L_s, $s \geq 1$, of regular word languages over $\hat{\Sigma}$ such that each $w\_nw(L_s)$ is accepted by a NWA with $O(s)$ states, but every word automaton accepting L_s must have 2^s states.*

PROOF. Let $\Sigma = \{a,b\}$. For $s \geq 1$, let $L_s = \{path(w) \mid w \in \Sigma^s\}$. To accept L_s, the automaton must check that the input word is of the form $\langle a_1 \langle a_2 \ldots \langle a_s a_s\rangle \ldots a_2\rangle a_1\rangle$. It is easy to check that L_s is regular, but a word automaton accepting L_s must have 2^s states (in fact, even nondeterminism won't help in this case).

The NWA simply needs a counter to ensure that the depth is s, and at each call, it passes the current symbol along the hierarchical edge. At a return, the symbol along the hierarchical edge must match the symbol being read. In fact, $s+2$ states suffice to accept L_s. $\square$

3.4 Bottom-up Automata

A nested word automaton $A = (Q, q_0, F, \delta_c, \delta_i, \delta_r)$ is said to be *bottom-up* iff the linear component of the call-transition function does not depend on the current state: $\delta_c^l(q,a) = \delta_c^l(q',a)$ for all $q, q' \in Q$ and $a \in \Sigma$. Consider the run of a bottom-up NWA A on a nested word n, let i be a call with return-successor j. Then, A processes the rooted subword $n[i,j]$ without using the prefix of n upto i. This does not limit expressiveness provided there are no unmatched calls. However, if $i \rightsquigarrow +\infty$, then the acceptance of n by A does not depend at all on the prefix $n[1,i-1]$, and this causes problems. In particular, for $\Sigma = \{a,b\}$, the language containing the single nested word $a\langle a$ can be accepted by a flat NWA, but not by a bottom-up NWA (if a bottom-up NWA accepts $a\langle a$, then it will also accept $n\langle a$, for every nested word n). To avoid this anomaly, we will assume that bottom-up automata process only well-matched words.

THEOREM 4. *Given an NWA A with s states, one can effectively construct a weak bottom-up NWA B with $s^s|\Sigma|$ states such that $L(A) \cap WNW(\Sigma) = L(B) \cap WNW(\Sigma)$.*

PROOF. Recall that we can transform any NWA into a weak one, and this transformation increases the number of states by a factor of $|\Sigma|$. Let $A = (Q, q_0, F, \delta_c^l, \delta_i, \delta_r)$ be a weak NWA. A state of B is a function $f : Q \mapsto Q$. The automaton B simulates the behavior of A in the following way. Consider a nested word n and a position i with call-parent j. The state of B before processing position i is f such that the subword $n[j, i-1]$ takes A from q to $f(q)$, for each $q \in Q$.

The initial state of B is the identity function. A state f is final if $f(q_0) \in F$. After reading an a-labeled call, the state of B is f such that $f(q) = \delta_c^l(q, a)$. While reading an a-labeled internal in state f, B updates its state to f' such that $f'(q) = \delta_i(f(q), a)$. While reading an a-labeled return in state f, if the state along the hierarchical edge is g, then B updates its state to f' such that $f'(q) = \delta_r(f(g(q)), g(q), a)$. □

A variety of definitions of bottom-up tree automata have been considered in the literature. All of these can be viewed as special cases of bottom-up NWAs. In particular, *bottom-up stepwise tree automata* are very similar and process the input in the same order [5, 15]. The only difference is that stepwise automata were defined to read only tree words, and process the symbol at a call when first encountered. That is, a stepwise bottom-up tree automaton is a weak bottom-up NWA on tree words with the restriction that $\delta_r : Q \times Q \times \Sigma \mapsto Q$ does not depend on its third argument.

LEMMA 1. *If $L \subseteq OT(\Sigma)$ is accepted by a stepwise bottom-up tree automaton with s states, then there exists a bottom-up NWA A with s states such that $nw\_t(L(A)) = L$.*

Since stepwise bottom-up tree automata accept all regular tree languages, it follows that NWAs can define all regular tree languages. Also, stepwise automata have been shown to be more succinct than many other classes of tree automata [15], so succinctness gap of NWAs with respect to bottom-up NWAs carries over to these classes. Over word encoding, the number of states of a minimal word automaton accepting a language L is the index of the corresponding right-congruence (two words u and v are equivalent iff for all suffixes w, $uw \in L$ iff $vw \in L$), while the number of states of a minimal bottom-up tree automaton accepting a language L is the index of the corresponding congruence (two well-matched words u and v are equivalent iff for all prefixes w and suffixes w', $wuw' \in L$ iff $wvw' \in L$). The right congruence induced by a language can have exponentially less number of classes than the congruence induced by L. This leads to the following exponential gap, shown using techniques developed for defining congruences for nested words [2]:

THEOREM 5. *There exists a family L_s, $s \geq 1$, of regular languages of tree words such that each L_s is accepted by a flat NWA with $O(s^2)$ states, but every bottom-up NWA accepting L_s must have 2^s states.*

PROOF. Let $\Sigma = \{a, b\}$. We will use L to denote the set $\{\langle a\rangle, \langle b\rangle\}$. For $s \geq 1$, consider the language L_s of tree words of the form $\langle a\langle b\rangle^m \langle a L^{i-1} \langle a\rangle L^{s-i} a\rangle a\rangle$, where $i = m \ mod \ s$.

First, we want to establish that there is a deterministic word automaton with $O(s^2)$ states accepting L_s. The automaton can compute the value of $i = m \ mod \ s$ after reading $\langle a\langle b\rangle^m \langle a$ by counting the number of repetitions of $\langle b\rangle$ modulo s using $O(s)$ states. Then, it must ensure that what follows is $L^{i-1}\langle a\rangle L^{s-i} a\rangle a\rangle$. For each value of i, this can be done using $O(s)$ states.

Let A be a bottom-up NWA accepting L_s. Let q be the unique state of A having read the prefix $\langle a\langle b\rangle^m \langle a$. This state q is independent of m since A is bottom up. The set L^s contains 2^s well-matched words. If A has less than 2^s states then there must exist two distinct words n and n' in L^s such that A goes to the same state q' after reading both n and n' starting in state q. Since n and n' are distinct, they must differ in some block. That is, there must exist $1 \leq i \leq s$ such that n is of the form $L^{i-1}\langle a\rangle L^{s-i}$ and n' is of the form $L^{i-1}\langle b\rangle L^{s-i}$. Now consider the words $\langle a\langle b\rangle^i \langle a\, n\, a\rangle a\rangle$ and $\langle a\langle b\rangle^i \langle a\, n'\, a\rangle a\rangle$. Only one of them is in L_s, but A will either accept both or reject both. □

3.5 Joinless Automata

A nested word automaton at a return position joins the information flowing along the linear edge and the hierarchical edge. In this section, we study the impact of disallowing such a join. A joinless automaton operates in two modes, linear and hierarchical. Initially it is in linear mode. At a call, it decides either to stay in the linear mode propagating only the dummy initial state along the hierarchical edge, or to enter the hierarchical mode and process the subword upto the matching return and the suffix after the return independently. At a return, if the automaton is in the linear mode, it checks that the state along the hierarchical edge is initial, and continues based on the current state. In a hierarchical mode the automaton behaves like a top-down tree automaton, and at a return, next state is based upon the state propagated along the hierarchical edge and the only information along the linear edge is whether the inside subword is accepted or not.

A *nondeterministic joinless nested word automaton* has finite set Q of states partitioned into Q_l and Q_h, a set $Q_0 \subseteq Q$ of initial states, a set $F \subseteq Q$ of final states, a call-transition relation $\delta_c \subseteq (Q_h \times \Sigma \times Q_h \times Q_h) \cup (Q_l \times \Sigma \times Q \times Q)$; an internal-transition relation $\delta_i \subseteq (Q_h \times \Sigma \times Q_h) \cup (Q_l \times \Sigma \times Q)$; a return-transition relation $\delta_r \subseteq (Q_h \times \Sigma \times Q_h) \cup (Q_l \times \Sigma \times Q)$. The automaton is deterministic if there is only one initial state and choice of at most one transition given the current state and symbol. A run of A on a nested word $n = (a_1, \ldots a_\ell, \rightsquigarrow)$ consists of states $q_0, q_1 \cdots q_\ell$ and q_{ij}, for $i \rightsquigarrow j$, (define $q_{-\infty j} = q_0$) such that $q_0 \in Q_0$, for each position $1 \leq i \leq \ell$, (1) if i is a call with $i \rightsquigarrow j$ then $(q_{i-1}, a_i, q_i, q_{ij}) \in \delta_c$; (2) if i is an internal then $(q_{i-1}, a_i, q_i) \in \delta_i$; (3) if i is a return with $j \rightsquigarrow i$ then either $q_{i-1} \in Q_l$ and $q_{ji} = q_0$ and $(q_{i-1}, a_i, q_i) \in \delta_r$, or $q_{i-1} \in Q_h \cap F$ and $(q_{ji}, a_i, q_i) \in \delta_r$.

Note that a flat automaton is joinless with $Q_l = Q$: all states are linear. We will call a joinless automaton *top-down* if Q_l is empty and all states are hierarchical. Over tree words, the standard definition of top-down tree automata is the same as our notion of top-down automata:

LEMMA 2. *If $L \subseteq OT(\Sigma)$, then L is accepted by a (non)deterministic top-down tree automaton with s states iff there exists a (non)deterministic top-down NWA A with s states such that $nw\_t(L(A)) = L$.*

This implies that the well-known expressiveness deficiency of deterministic top-down tree automata applies in case of nested words. Consider the requirement that the nested word contains some a-labeled symbol. This can be checked by a flat (and hence, deterministic joinless) automaton, but not by a top-down automaton. The requirement that the input word is a tree word can be checked by a deterministic top-down (and hence, deterministic joinless) automaton, but not by a flat automaton. Thus, expressiveness of deterministic top-down automata and flat automata is incomparable. The conjunction of these two requirements can be checked by an NWA but not by a deterministic joinless automaton:

THEOREM 6. *Deterministic joinless nested word automata are strictly less expressive than nested word automata.*

PROOF. Let $\Sigma = \{a, b\}$. Consider the language of tree words of the form $(\langle a \rangle^s \langle b \langle c \rangle b \rangle \langle c \rangle (a \rangle)^s$, for some $s \geq 0$ and $c \in \Sigma$. We can construct an NWA to accept this language. Let A be a deterministic joinless automaton with s states. Consider the run of the automaton on a string of s a-labeled calls. If the automaton stays in linear mode throughout this, then one can pump in this part without changing the labeling of hierarchical edges, and use that to show that A cannot accept the language correctly. Hence, the automaton must be in hierarchical mode at the end of this prefix. Consequently, after reading the b-labeled call, it forks off two independent copies, with state q_1 along linear edge and state q_2 along hierarchical edge. Now the obligation is that the next symbol read by q_1 must match the next symbol read by q_2. We can show that this cannot be enforced by considering accepting runs for different symbols and combining them. $\square$

Nondeterminism can be used to address this deficiency:

THEOREM 7. *Given a nondeterministic NWA A with s states, one can effectively construct a nondeterministic joinless NWA B with $O(s^2|\Sigma|)$ states such that $L(A) = L(B)$.*

PROOF. Let $A = (Q, Q_0, F, \delta_c, \delta_i, \delta_r)$ be an NWA. For each state q, B has a corresponding linear state. For every pair (q, q') of states of A, B has a hierarchical state meaning that the current state of A is q and there is an obligation that the state of A will be q' at the first unmatched return. We will also need auxiliary hierarchical states of the form (q, q', a) to label hierarchical edges to mean that the symbol at the return is guessed to be a.

The initial states are Q_0. Linear states in F and hierarchical states of the form (q, q) are accepting. For every internal transition (q, a, q') of A, B has a corresponding linear internal transition and for every q'', it also has a hierarchical internal transition $((q, q''), a, (q', q''))$. For every call transition (q, a, q_l, q_h), there is a linear call transition (q, a, q_l, q_0) denoting the guess that there is no matching return, and for every return transition (q_1, q_h, b, q_2), there is a linear return transition (q_1, b, q_2), and for every q', B has a hierarchical call transition $((q, q'), a, (q_l, q_1), (q_2, q', b))$. Note that here B is demanding a run from q_l to q_1 on the inside subword, and the accepting condition ensures that this obligation is met. The hierarchical return transitions of B are of the form $((q, q', a), a, (q, q'))$. $\square$

Note that a similar theorem *does not* hold for top-down automata. The reason is that in a top-down automaton information cannot flow outwards. Hence, if a position is an unmatched return, then even a nondeterministic top-down automaton won't be able to relate the subwords before and after this position.

3.6 Path Languages

The mix of top-down and bottom-up traversal in nested word automata can be better explained on unary trees. For a word language $L \subseteq \Sigma^*$, let $path(L) = \{path(w) \mid w \in L\}$ be the corresponding language of tree words. We call such languages *path languages*. Observe that for unary trees, the multitude of definitions of tree automata collapse to two: top-down and bottom-up. Top-down tree automata for $path(L)$ correspond to word automata accepting L, while bottom-up tree automata correspond to word automata processing the words in reverse. The following lemma follows from definitions:

LEMMA 3. *For a word language L, $nw\_t(path(L))$ is accepted by a deterministic top-down tree automaton with s states iff L is accepted by a deterministic word automaton with s states, and $nw\_t(path(L))$ is accepted by a deterministic bottom-up tree automaton with s states iff L^R, the reverse of L, is accepted by a deterministic word automaton with s states.*

It follows that $path(L)$ is a regular language of nested words iff L is a regular language of words. Also, for path languages, deterministic top-down and deterministic bottom-up automata can express all regular languages. Given that a word language L and its reverse can have exponentially different complexities in terms of the number of states of deterministic acceptors, we get

THEOREM 8. *There exists a family L_s, $s \geq 1$, of regular path languages such that each L_s is accepted by a NWA with $O(s)$ states, but every deterministic bottom-up or top-down NWA accepting L_s must have 2^s states.*

PROOF. For $\Sigma = \{a, b\}$, let L_s be $\Sigma^s a \Sigma^* a \Sigma^s$. An NWA with linear number of states can accept the corresponding path language: it needs to count s calls going down, count s returns on way back, and also make sure that the input word is indeed a path word by passing each call-symbol along the hierarchical edge. It is easy to see that L_s requires 2^s states for a DFA to enforce the constraint that $s+1$-th symbol from end is an a. Since L_s is its own reverse, from Lemma 5, the theorem follows. $\square$

4. PUSHDOWN AUTOMATA

In this section, we generalize the classical definitions of pushdown automata over words and trees to nested words. Note that pushdown automata over words can be used to implement finite-state NWAs where the stack stores the states labeling the pending hierarchical edges so that these states can be accessed at returns. In this section, we will use the pushdown store for labeling itself so that at a call, the automaton can fork off two runs each with its own stack.

4.1 Pushdown Nested Word Automata

Given that nondeterministic pushdown automata over words are more expressive than deterministic ones (and correspond to the well understood class of context-free languages), we will consider only nondeterministic automata. Furthermore, we will restrict attention to joinless automata for two reasons. First, for finite-state acceptors, nondeterministic joinless automata can specify all regular languages, and generalize both word automata and top-down tree automata. Second, in presence of stacks, while there is an obvious and natural generalization of the transition relation in the joinless case, it's not clear how to join two stacks. Consistent with the classical definitions of pushdown automata, we will allow ϵ-transitions, but for simplicity of presentation, we will assume that the transitions updating the stack are precisely the ϵ-transitions. It is easy to verify that this does not change the class of languages accepted.

A *pushdown nested word automaton* consists of

- a finite set Q of states, partitioned into linear states Q_l and hierarchical states Q_h,
- a set of initial states $Q_0 \subseteq Q$,
- a finite set Γ of stack symbols,
- a bottom symbol $\perp \in \Gamma$,
- a call-transition relation δ_c which is a subset of $Q_l \times \Sigma \times Q \times Q$ and $Q_h \times \Sigma \times Q_h \times Q_h$,
- an internal-transition relation δ_i which is a subset of $Q_l \times \Sigma \times Q$ and $Q_h \times \Sigma \times Q_h$,
- a return-transition relation δ_r which is a subset of $Q_l \times \Sigma \times Q$ and $Q_h \times \Sigma \times Q_h$,
- a push-transition relation $\delta_+ \subseteq Q \times Q \times \Gamma \setminus \{\perp\}$, and
- a pop-transition relation $\delta_- \subseteq Q \times \Gamma \times Q$.

A *configuration* is a pair consisting of the automaton state and a sequence of stack symbols. We will use $C = Q \times \Gamma^*$ to denote the set of all configurations. The initial configuration of the automaton is $(q_0, \perp)$ for some $q_0 \in Q_0$. The automaton either processes a position of the input word, and updates the state exactly as in case of joinless automata without updating the stack, or it executes a push or a pop transition updating the state and the stack without processing the input. Thus, the automaton can take a sequence of ϵ-steps along a linear edge. The automaton accepts by empty stack. We also assume that the unmatched return edges are labeled with some default configuration.

To define runs formally, we lift the transition relations of the automaton to relate configurations. The internal-transition relation δ_i defines the relation $\Delta_i \subseteq C \times \Sigma \times C$ as follows: $((q,\alpha), a, (q',\alpha))$ belongs to Δ_i iff $(q,a,q') \in \delta_i$. The relations $\Delta_c \subseteq C \times \Sigma \times C \times C$ and $\Delta_r \subseteq C \times \Sigma \times C$ are defined using δ_c and δ_r in a similar manner. The push transition relation δ_+ defines the relation $\Delta_+ \subseteq C \times C$: $((q,\alpha),(q',\gamma\alpha)$ belongs to Δ_+ iff $(q,q',\gamma) \in \delta_+$. Similarly, $((q,\gamma\alpha),(q',\alpha)) \in \Delta_-$ iff (q,γ,q') is a pop-transition.

A run of A on a nested word $n = (a_1, \dots a_\ell, \rightsquigarrow)$ is a sequence $c_0, c_0', c_1, c_1', \cdots c_\ell, c_\ell'$ of configurations corresponding to linear edges, and configurations c_{ij}, for $i \rightsquigarrow j$ (let $c_{-\infty j} = (q_0, \perp)$ for some $q_0 \in Q_0$), labeling hierarchical edges, such that, for each i, (1) (c_i, c_i') is in the reflexive-transitive closure of $\Delta_+ \cup \Delta_-$; (2) if i is a call with $i \rightsquigarrow j$ then $(c_{i-1}', a_i, c_i, c_{ij}) \in \Delta_c$; (3) if i is an internal then $(c_{i-1}', a_i, c_i) \in \Delta_i$; (4) if i is a return with $j \rightsquigarrow i$ then either (a) the state of configuration c_{i-1}' is in Q_l and the state of the configuration c_{ji} is q_0 and $(c_{i-1}', a_i, c_i) \in \Delta_r$, or (b) the state of configuration c_{i-1} is in Q_h and $(c_{ji}, a_i, c_i) \in \Delta_r$. For such a run, c_0 is the *start* configuration, c_ℓ' is the *end* configuration, and each c_{i-1}' such that its state is in Q_h and i is a return, is called a *leaf* configuration.

The run is *initialized* if the start configuration is $(q_0, \perp)$ for some $q_0 \in Q_0$. The run is *accepting* if the stack in end configuration as well as each leaf configuration is empty. The nested word n is accepted by A if there is an initialized accepting run of A on n, and the language $L(A)$ consists of all the nested words accepted by A. A language $L \subseteq NW(\Sigma)$ of nested words is said to be a *pushdown* language if there exists a pushdown nested word automaton A such that $L(A) = L$.

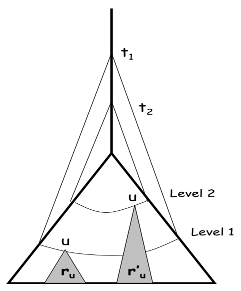

Figure 2: Pumping in trees

4.2 Expressiveness

A nondeterministic joinless NWA canbe easily changed to a pushdown NWA (by adding ϵ-transitions from final states that pop $\perp$ from the stack), and hence, pushdown NWAs can define all regular languages of nested words. A classical pushdown word automaton accepting by empty stack is also a special case where all states are linear. This implies that all context-free word languages are definable:

LEMMA 4. *If L is a context-free language over $\hat{\Sigma}$, then $w\_nw(L)$ is a pushdown language of nested words.*

Top-down pushdown tree automaton is also a special case where all states are hierarchical. This implies that context-free tree languages are definable:

LEMMA 5. *If $L \subseteq OT(\Sigma)$ is a context-free tree language, then $t\_nw(L)$ is a pushdown language of nested words.*

Context-free tree languages can be defined using nondeterministic top-down tree automata. Such automata, however, cannot simulate a linear stack-based information flow. A top-down automaton splits the run into two independent ones at a call position (i.e. a tree node). Nondeterminism can be used to exchange finite amount of information across the two runs (as in the proof of expressive completeness of joinless automata). However, this trick fails in case of pushdown automata.

THEOREM 9. *There exists a language L of nested words such that $nw\_w(L)$ is accepted by a pushdown word automaton, but $nw\_t(L)$ is not a context-free tree language.*

PROOF. Let $\Sigma = \{a,b\}$. Consider the language L of nested words n such that the number of a-labeled positions is equal to the number of b-labeled positions. This is a standard context-free word requirement. Suppose this language is accepted by a top-down pushdown tree automaton with s states. Consider a tree with a stem consisting of 2^s a-labeled nodes, followed by a full binary tree of depth s consisting of b-labeled nodes. Consider an accepting run of the automaton over this tree. Note that in the run of a pushdown tree automaton, a push-transition can get matched with multiple pop-transitions. See Figure 2. We can show that there must exist two positions t_1 and t_2 along the stem such that (1) the states labeling t_1 and t_2 are identical, (2) the pop-transitions matching the push at t_1 and t_2, respectively, are along the frontiers level 1 and 2, respectively and (3) the set of states labeling the positions at level 1 equals the set of states labeling level 2. Without loss of generality, we can assume that if a state appears multiple times at level 1, then the subtrees rooted at these multiple occurrences are identical. Same holds for level 2. For each $u \in U$, let r_u and r'_u denote the subtrees rooted at an occurrence of u at level 1 and 2, respectively. Now observe that we can delete the stem from t_1 to t_2, and replace each subtree r'_u by r_u, and this still gives us an accepting run. We can also duplicate the stem from t_1 to t_2, and replace each subtree r_u at level 1 by a copy of the subtree r'_u, while maintaining acceptance. Note that each pumping increases the depth of every leaf at least by 1, and hence, at least doubles the current number of b nodes, while the number of a nodes added is fixed (equal to the length of the stem from t_1 to t_2). This leads to a contradiction. □

4.3 Membership

The membership question is to decide, given a pushdown NWA A and a nested word n, whether $n \in L(A)$ holds. For pushdown word automata as well as for nondeterministic nested word automata, the membership problem is solvable in cubic time. For pushdown NWAs, the problem turns out to be NP-complete.

THEOREM 10. *The membership question for pushdown nested word automata is* NP*-complete.*

PROOF. For membership in NP, we need to show that if the nested word is accepted then it is accepted in a run in which the number of ϵ-transitions, and hence, the size of the stack can be polynomially bounded. This can be established easily.

For hardness, it suffices to consider unary alphabet. The proof is by reduction from satisfiability of CNF formulas. Given a formula over v variables and s clauses, consider the nested word $(\langle aa^v a\rangle)^s$. Initially, the automaton executes v ϵ-moves pushing either 0 or 1 onto its stack guessing a truth assignment to the variables. While reading a call, the automaton simply propagates the stack along the hierarchical edge. For the i-th copy of a^v enclosed between a call and a return, the automaton pops the stack and checks if the i-th clause is satisfied according to the popped assignment. The word is accepted if there exists an assignment satisfying all the clauses. □

Note that the NP-hardness is really due to the ability to propagate the same stack to distinct branches. Hence, membership problem is also NP-complete for pushdown tree automata.

4.4 Emptiness

Given a pushdown nested word automaton A, we want to decide if it accepts some word. Throughout let F denote the set of states from which $\perp$ can be popped, that is, $q \in F$ iff $(q, \perp, q') \in \delta_-$ for some q'.

Let's first briefly recall the emptiness check for pushdown word automata. The key to the procedure is computing the so-called "stackless summaries" of runs: we want to define a relation $R \subseteq Q \times Q$ such that $R(q, q')$ holds precisely when there is a word w and a run of the automaton over w starting in the configuration (q, ϵ) and ending in (q', ϵ). This would also imply a run from the start configuration (q, α) to the end configuration (q', α), for all possible stack contents α. The relation $R(q, q')$ can be defined using inductive rules. In particular, if $R(q, q')$ holds, and there is a push-transition from q_1 to q and a pop-transition from q' to q_2, both involving the same stack symbol γ, then one can infer $R(q_1, q_2)$. The relation R can be computed in polynomial-time. The language is empty iff $R(q_0, q_f)$ holds for some $q_f \in F$.

For pushdown (top-down) tree automata, the notion of summaries needs to be generalized. In particular, when a stack symbol γ is pushed, it can get popped along multiple branches. The definition of a stackless summary then is a relation $R \subseteq Q \times 2^Q$ such that $R(q, U)$ holds precisely when there is a tree t and a run of the automaton over t starting in the configuration (q, ϵ) at the root and each leaf configuration is of the form (q', ϵ) for some $q' \in U$. The push-pop rule for words generalizes to: $R(q', U')$ can be inferred from $R(q, U)$ if there is a stack symbol γ such that there is a transition from q' to q that pushes γ, and for each $u \in U$, there is some $u' \in U'$ such that there is a transition from u to u' popping γ. The number of summaries now is exponential, and so is the complexity of the emptiness test. The language is empty iff $R(q_0, U)$ holds for some $U \subseteq F$.

The summaries for pushdown nested word automata combine these two ideas. Our summaries will be of the form $R(q, U, q')$. Intuitively, q is the start state of the run, q' is the end state, and U is the set of states labeling the leaf configurations. The stack is empty in the start, end, and all leaf configurations. Formally, define $R \subseteq Q \times 2^{Q_h} \times Q$ such that $R(q, U, q')$ holds iff there is a nested word n and a run r of the automaton over n such that the start configuration is (q, ϵ), the end configuration is (q', ϵ), and each leaf configuration is of the form (u, ϵ), for some $u \in U$. Note that all states in U must be hierarchical, and if both q and q' are hierarchical states, then the word must be well-matched. The language of a pushdown word automaton A is empty iff $R(q_0, U, q_f)$ holds for some $U \subseteq F$ and $q_f \in F$. It follows that emptiness can be checked by computing the relation R. Let R be the smallest subset of $Q \times 2^{Q_h} \times Q$ that satisfies the following constraints:

Internal transitions If $(q, a, q') \in \delta_i$ then $(q, \emptyset, q') \in \mathsf{R}$.

Linear calls If $(q, a, q', q_0) \in \delta_c$ for $q \in Q_l$ then $(q, \emptyset, q') \in \mathsf{R}$.

Linear returns If $(q, a, q') \in \delta_r$ for $q \in Q_l$ then $(q, \emptyset, q') \in \mathsf{R}$.

Hierarchical call-returns If $(q, a, q_l, q_h) \in \delta_c$ with $q_l \in Q_h$ and $(q_h, b, q') \in \delta_r$ then $(q, \{q_l\}, q') \in \mathsf{R}$.

Push-pop transitions If $(q, U, q') \in \mathsf{R}$, $(q_1, q, \gamma) \in \delta_+$, $(q', \gamma, q_2) \in \delta_-$, and for each $u \in U$, $(u, \gamma, u') \in \delta_-$ for some $u' \in U'$. then $(q_1, U', q_2) \in \mathsf{R}$.

Linear concatenation If (q, U, q') and (q', U', q'') are in R then so is $(q, U \cup U', q'')$.

Hierarchical concatenation If (q, U, q') and (u, U', v) are in R for some $u \in U$, then so is $(q, (U \cup U' \cup \{v\}) \setminus \{u\}, q')$.

In the above rules, the internal push-pop rule is the generalization of the corresponding rules for matching pushes with pops in case of words and trees. In the hierarchical mode, the calls and returns must be processed jointly, and this is captured by the hierarchical call-returns rule. For concatenating summaries, we need to consider both linear concatenation at top-level and hierarchical concatenation at the leaf-level. For correctness, we need to show that the relation R capturing summaries of runs coincides with the relation R defined using above derivation rules. The desired relation R can be computed in exponential time. Putting all pieces together, and using the fact that emptiness for pushdown games (or pushdown tree automata) is EXPTIME-hard [12], we get

THEOREM 11. *The language emptiness problem for pushdown nested word automata is* EXPTIME*-complete.*

5. CONCLUSIONS

We have shown that nested words is a suitable model for data that has both linear and hierarchical structure. Both words and ordered trees are special cases of nested words, and nested words support both word and tree operations. We have shown that nested word automata combine left-to-right, top-down, and bottom-up traversals, and are exponentially more succinct than word automata as well as all varieties of tree automata. We have also introduced pushdown automata over nested words, studied their decision problems, and shown them to be more expressive than pushdown tree automata.

In terms of future work, on practical side, we need to explore if compiling existing XML query languages into nested word automata reduces query processing time. On theoretical side, many problems such as algorithms for edit distances, transducers, and temporal and first-order logics, seem worth investigating.

Acknowledgements: When I gave a talk on nested words at the Newton Institute Workshop on Games and Verification in June 2006, many people including Thomas Schwentick, Wolfgang Thomas, and Moshe Vardi, asked questions regarding the relationship between ordered trees and nested words. I thank them for initiating this research, and also the workshop organizers for a stimulating meeting. I would also like to thank Swarat Chaudhuri, Kousha Etessami, Neil Immerman, Leonid Libkin, P. Madhusudan, Benjamin Pierce, and Mahesh Viswanathan, for fruitful discussion regarding nested words. This research was partially supported by US National Science Foundation under grants CNS 0524059 and CPA 0541149.

6. REFERENCES

[1] R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. In *Proc. 18th International Conference on Computer-Aided Verification*, LNCS 4144, pages 329–342. Springer, 2006.

[2] R. Alur, V. Kumar, P. Madhusudan, and M. Viswanathan. Congruences for visibly pushdown languages. In *Automata, Languages and Programming: Proceedings of the 32nd ICALP*, LNCS 3580, pages 1102–1114. Springer, 2005.

[3] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 202–211, 2004.

[4] R. Alur and P. Madhusudan. Adding nesting structure to words. In *Developments in Language Theory*, LNCS 4036, pages 1–13, 2006.

[5] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

[6] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Draft, Available at `http://www.grappa.univ-lille3.fr/tata/`, 2002.

[7] T. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, pages 173–189. Springer, 2003.

[8] I. Guessarian. Pushdown tree automata. *Mathematical Systems Theory*, 16:237–264, 1983.

[9] D. Knuth. A characterization of parenthesis languages. *Information and Control*, 11(3):269–289, 1967.

[10] V. Kumar, P. Madhusudan, and M. Viswanathan. Minimization, learning, and conformance testing of Boolean programs. In *CONCUR'06: 17th International Conference on Concurrency Theory*, LNCS 4137, pages 203–217. Springer, 2006.

[11] V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown languages for streaming XML. In *Proceedings of the 16th International World Wide Web Conference*, 2007.

[12] O. Kupferman, N. Piterman, and M. Vardi. Pushdown specifications. In *Proc. 9th Intl. Conf. on Logics for Programming, Artificial Intelligence, and Reasoning*, LNCS 2514, pages 262–277. Springer, 2002.

[13] L. Libkin. Logics for unranked trees: An overview. In *Automata, Languages and Programming, 32nd International Colloquium, Proceedings*, LNCS 3580, pages 35–50. Springer, 2005.

[14] C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference*, LNCS 3328, pages 408–420. Springer, 2004.

[15] W. Martens and J. Niehren. Minimizing tree automata for unranked trees. In *Proceedings of the 10th International Symposium on Database Programming Languages*, pages 233–247, 2005.

[16] R. McNaughton. Parenthesis grammars. *Journal of the ACM*, 14(3):490–500, 1967.

[17] F. Neven. Automata, logic, and XML. In *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic, CSL 2002*, pages 2–26. Springer, 2002.

[18] T. Schwentick. Automata for XML – a survey. Technical report, University of Dortmund, 2004.

[19] W. Thomas. On logics, tilings, and automata. In *Automata, Languages and Programming, 18th Intl. Colloquium*, LNCS 510, pages 441–454, 1991.