Adding Nesting Structure to Words *

Rajeev Alur
University of Pennsylvania
alur@cis.upenn.edu

P. Madhusudan
University of Illinois, Urbana-Champaign
madhu@cs.uiuc.edu

Abstract

We propose the model of *nested words* for representation of data with both a linear ordering and a hierarchically nested matching of items. Examples of data with such dual linear-hierarchical structure include executions of structured programs, annotated linguistic data, and HTML/XML documents. Nested words generalize both words and ordered trees, and allow both word and tree operations. We define *nested word automata*—finite-state acceptors for nested words, and show that the resulting class of regular languages of nested words has all the appealing theoretical properties that the classical regular word languages enjoys: deterministic nested word automata are as expressive as their nondeterministic counterparts; the class is closed under union, intersection, complementation, concatenation, Kleene-*, prefixes, and language homomorphisms; membership, emptiness, language inclusion, and language equivalence are all decidable; and definability in monadic second order logic corresponds exactly to finite-state recognizability. We also consider regular languages of infinite nested words and show that the closure properties, MSO-characterization, and decidability of decision problems carry over.

The linear encodings of nested words give the class of *visibly pushdown languages* of words, and this class lies between balanced languages and deterministic context-free languages. We argue that for algorithmic verification of structured programs, instead of viewing the program as a context-free language over words, one should view it as a regular language of nested words (or equivalently, a visibly pushdown language), and this would allow model checking of many properties (such as stack inspection, pre-post conditions) that are not expressible in existing specification logics.

We also study the relationship between ordered trees and nested words, and the corresponding automata: while the analysis complexity of nested word automata is the same as that of classical tree automata, they combine both bottom-up and top-down traversals, and enjoy expressiveness and succinctness benefits over tree automata.

1 Introduction

Linearly structured data is usually modeled as words, and queried using word automata and related specification languages such as regular expressions. Hierarchically structured data is naturally modeled as (unordered) trees, and queried using tree automata. In many applications including executions of structured programs, annotated linguistic data, and primary/secondary bonds in genomic sequences, the data has *both* a natural linear sequencing of positions and a hierarchically nested matching of positions. For example, in natural language processing, the sentence is a linear sequence of words, and parsing into syntactic categories imparts the hierarchical structure. Sometimes, even though the only logical structure on data is hierarchical, 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.

In this paper, we propose the model of *nested words* for representing and querying data with dual linear-hierarchical structure. A nested word consists of a sequence of linearly ordered positions, augmented with nesting edges connecting calls to returns (or open-tags to close-tags). The edges do not cross creating a

*This paper unifies and extends results that have appeared in conference papers [AM04], [AM06], and [Alu07].

properly nested hierarchical structure, and we allow 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 the 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. Binary trees, ranked trees, unranked trees, hedges, and documents that do not parse correctly, all can be represented with equal ease. Word operations such as prefixes, suffixes, concatenation, reversal, as well as tree operations referring to the hierarchical structure, can be defined naturally on nested words.

We define and study finite-state automata as acceptors of nested words. A *nested word automaton* (NWA) is similar to a classical finite-state word automaton, and reads the input from left to right according to the linear sequence. At a call, it can propagate states along both linear and nesting outgoing edges, and at a return, the new state is determined based on states labeling both the linear and nesting incoming edges. The resulting class of *regular* languages of nested words has all the appealing theoretical properties that the regular languages of words and trees enjoy. In particular, we show that deterministic nested word automata are as expressive as their nondeterministic counterparts. Given a nondeterministic automaton A with s states, the determinization involves subsets of pairs of states (as opposed to subsets of states for word automata), leading to a deterministic automaton with 2^{s^2} states, and we show this bound to be tight. The class is closed under all Boolean operations (union, intersection, and complement), and a variety of word operations such as concatenation, Kleene-$*$, and prefix-closure. The class is also closed under nesting-respecting language homomorphisms, which can model tree operations. Decision problems such as membership, emptiness, language inclusion, and language equivalence are all decidable. We also establish that the notion of regularity coincides with the definability in the *monadic second order logic* (MSO) of nested words (MSO of nested words has unary predicates over positions, first and second order quantifiers, linear successor relation, and the nesting relation).

The motivating application area for our results has been software verification. Pushdown automata naturally model the control flow of sequential computation in typical programming languages with nested, and potentially recursive, invocations of program modules such as procedures and method calls. Consequently, a variety of program analysis, compiler optimization, and model checking questions can be formulated as decision problems for pushdown automata. For instance, in contemporary software model checking tools, to verify whether a program P (written in C, for instance) satisfies a regular correctness requirement φ (written in linear temporal logic LTL, for instance), the verifier first abstracts the program into a pushdown model P^a with finite-state control, compiles the negation of the specification into a finite-state automaton $A_{\neg\varphi}$ that accepts all computations that violate φ and algorithmically checks that the intersection of the languages of P^a and $A_{\neg\varphi}$ is empty. The problem of checking regular requirements of pushdown models has been extensively studied in recent years leading to efficient implementations and applications to program analysis [RHS95, BEM97, BR00, ABE$^+$05, HJM$^+$02, EKS03, CW02]. While many analysis problems such as identifying dead code and accesses to uninitialized variables can be captured as regular requirements, many others require inspection of the stack or matching of calls and returns, and are context-free. Even though the general problem of checking context-free properties of pushdown automata is undecidable, algorithmic solutions have been proposed for checking many different kinds of non-regular properties. For example, access control requirements such as "a module A should be invoked only if the module B belongs to the call-stack," and bounds on stack size such as "if the number of interrupt-handlers in the call-stack currently is less than 5, then a property p holds" require inspection of the stack, and decision procedures for certain classes of stack properties already exist [JMT99, CW02, EKS03, CMM$^+$04]. A separate class of non-regular, but decidable, properties includes the temporal logic CARET that allows matching of calls and returns and can express the classical correctness requirements of program modules with pre and post conditions, such as "if p holds when a module is invoked, the module must return, and q holds upon return" [AEM04]. This suggests that the answer to the question "which class of properties are algorithmically checkable against pushdown models?" should be more general than "regular word languages." Our results suggest that the answer lies in viewing the program as a generator of nested words. The key feature of checkable requirements,

such as stack inspection and matching calls and returns, is that the stacks in the model and the property are correlated: while the stacks are not identical, the two synchronize on when to push and when to pop, and are always of the same depth. This can be best captured by modeling the execution of a program P as a nested word with nesting edges from calls to returns. Specification of the program is given as a nested word automaton A (or written as a formula φ in one of the new temporal logics for nested words), and verification corresponds to checking whether every nested word generated by P is accepted by A. If P is abstracted into a model P^a with only boolean variables, then it can be interpreted as an NWA, and verification can be solved using decision procedures for NWAs. Nested-word automata can express a variety of requirements such as stack-inspection properties, pre-post conditions, and interprocedural data-flow properties. More broadly, modeling structured programs and program specifications as languages of nested words generalizes the linear-time semantics that allows integration of Pnueli-style temporal reasoning [Pnu77] and Hoare-style structured reasoning [Hoa69]. We believe that the nested-word view will provide a unifying basis for the next generation of specification logics for program analysis, software verification, and runtime monitoring.

Given a language L of nested words over Σ, the linear encoding of nested words gives a language $\hat{L}$ over the tagged alphabet consisting of symbols tagged with the type of the position. If L is regular language of nested words, then $\hat{L}$ is context-free. In fact, the pushdown automata accepting $\hat{L}$ have a special structure: while reading a call, the automaton must push one symbol, while reading a return symbol, it must pop one symbol (if the stack is non-empty), and while reading an internal symbol, it can only update its control state. We call such automata *visibly pushdown automata* and the class of word languages they accept *visibly pushdown languages* (VPL). Since our automata can be determinized, VPLs correspond to a subclass of deterministic context-free languages (DCFL). We give a grammar-based characterization of VPLs, which helps in understanding of VPLs as a generalization of parenthesis languages, bracketed languages, and balanced languages [McN67, GH67, BB02]. Note that VPLs have better closure properties than CFLs, DCFLs, or parenthesis languages: CFLs are not closed under intersection and complement, DCFLs are not closed under union, intersection, and concatenation, and balanced languages are not closed under complement and prefix-closure.

Data with dual linear-hierarchical structure is traditionally modeled using binary, and more generally, using ordered unranked, trees, and queried using tree automata (see [Nev02, Lib05, Sch07] 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. It turns out that, hedges, where a hedge is a sequence of ordered trees, are a special class of nested words, namely, the ones corresponding to Dyck words, and regular hedge languages correspond to balanced languages. For document processing, nested words do have many advantages over ordered trees as trees lack an explicit ordering of all nodes. Tree-based representation 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 $p_1, \ldots p_k$ appear in the document in that order (that is, the regular expression $\Sigma^* p_1 \Sigma^* \ldots p_k \Sigma^*$ over the linear order) compiles into a deterministic word automaton (and hence deterministic NWA) of linear size, but standard deterministic bottom-up tree automaton for this query must be of size exponential in k. In fact, NWAs can be viewed as a kind of tree automata such that both bottom-up tree automata and top-down tree automata are special cases.

Analysis of liveness requirements such as "every write operation must be followed by a read operation" is formulated using automata over infinite words, and the theory of ω-regular languages is well developed with many of the counterparts of the results for regular languages (c.f. [Tho90, VW94]). Consequently, we also define nested ω-words and consider nested word automata augmented with acceptance conditions such as *Büchi* and *Muller*, that accept languages of nested ω-words. We establish that the resulting class of regular languages of nested ω-words is closed under operations such as union, intersection, complementation, and homomorphisms. Decision problems for these automata have the same complexity as the corresponding

problems for NWAs. As in the finite case, the class can be characterized by the monadic second order logic. The significant difference is that deterministic automata with Muller acceptance condition on states that appear infinitely often along the linear run do not capture all regular properties: the language "there are only finitely many pending calls" can be easily characterized using a nondeterministic Büchi NWA, and we prove that no deterministic Muller automaton accepts this language. However, we show that nondeterministic Büchi NWAs can be complemented and hence problems such as checking for inclusion are still decidable.

Outline

Section 2 defines nested words and their word encodings, and gives different application domains where nested words can be useful. Section 3 defines nested word automata and the notion of regularity. We consider some variations of the definition of the automata, including the nondeterministic automata, show how NWAs can be useful in program analysis, and establish closure properties. Section 4 gives logic based characterization of regularity. In Section 5, we define visibly pushdown languages as the class of word languages equivalent to regular languages of nested words. We also give grammar based characterization, and study relationship to parenthesis languages and balanced grammars. Section 6 studies decision problems for NWAs. Section 7 presents encoding of ordered trees and hedges as nested words, and studies the relationship between regular tree languages, regular nested-word languages, and balanced languages. To understand the relationship between tree automata and NWAs, we also introduce bottom-up and top-down restrictions of NWAs. Section 8 considers the extension of nested words and automata over nested words to the case of infinite words. Finally, we discuss related work and conclusions.

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 $\{-\infty, 1, 2, \ldots \ell\} \times \{1, 2, \ldots \ell, +\infty\}$ such that

1. Nesting edges go only forward: if $i \rightsquigarrow j$ then $i < j$;
2. No two nesting edges share a position: for $1 \leq i \leq \ell$, $|\{j \mid i \rightsquigarrow j\}| \leq 1$ and $|\{j \mid j \rightsquigarrow i\}| \leq 1$;
3. Nesting edges do not cross: 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 $1 \leq i \leq \ell$, the position i is called a *call position*. For a call position i, if $i \rightsquigarrow +\infty$, then i is called a *pending call*, otherwise i is called a *matched call*, and the unique position j such that $i \rightsquigarrow j$ is called its *return-successor*. Similarly, when $i \rightsquigarrow j$ holds, for $1 \leq j \leq \ell$, the position j is called a *return position*. For a return position j, if $-\infty \rightsquigarrow j$, then j is called a *pending return*, otherwise j is called a *matched return*, and the unique position i such that $i \rightsquigarrow j$ is called its *call-predecessor*. Our definition requires that a position cannot be both a call and a return. A position $1 \leq i \leq \ell$ that is neither a call nor a return is called *internal*.

A matching relation $\rightsquigarrow$ of length ℓ can be viewed as a a directed acyclic graph over ℓ vertices corresponding to positions. For $1 \leq i < \ell$, there is a linear edge from i to $i+1$. The initial position has an incoming linear edge with no source, and the last position has an outgoing linear edge with no destination. For matched call positions i, there is a *nesting edge* (sometimes also called a *summary edge*) from i to its return-successor. For pending calls i, there is a nesting edge from i with no destination, and for pending returns j, there is a nesting edge to j with no source. We call such graphs corresponding to matching relations as *nested sequences*. Note that a call has indegree 1 and outdegree 2, a return has indegree 2 and outdegree 1, and an internal has indegree 1 and outdegree 1.

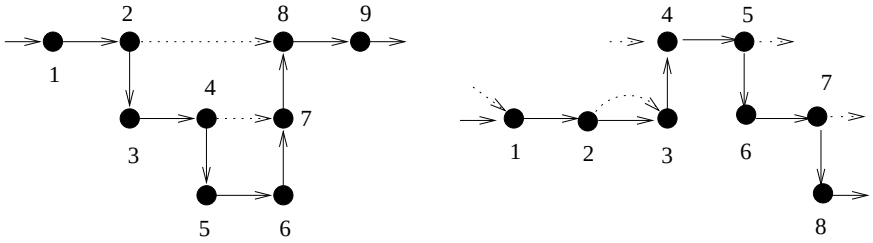

Figure 1: Sample nested sequences

Figure 1 shows two nested sequences. Nesting edges are drawn using dotted lines. For the left sequence, the matching relation is $\{(2,8),(4,7)\}$, and for the right sequence, it is $\{(-\infty,1),(-\infty,4),(2,3),(5,+\infty),(7,+\infty)\}$. Note that our definition allows a nesting edge from a position i to its linear successor, and in that case there will be two edges from i to $i+1$; this is the case for positions 2 and 3 of the second sequence. The second sequence has two pending calls and two pending returns. Also note that all pending return positions in a nested sequence appear before any of the pending call positions.

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. In other words, a nested word is a nested sequence whose positions are labeled with symbols in Σ. Let us denote the set of all 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. In Figure 1, only the left sequence is well-matched, and neither of the sequences is rooted.

While the length of a nested word captures its linear complexity, its (nesting) depth captures its hierarchical complexity. For $i \rightsquigarrow j$, we say that the call position i is pending at every position k such that $i < k < j$. The *depth* of a position i is the number of calls that are pending at i. Note that the depth of the first position 0, it increases by 1 following a call, and decreases by 1 following a matched return. The *depth* of a nested word is the maximum depth of any of its positions. In Figure 1, both sequences have depth 2.

2.2 Word Encoding

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. For each symbol a in Σ, we will use a new symbol $\langle a$ to denote a call position labeled with a, and a new symbol $a\rangle$ to denote a return position labeled with a. We use $\langle\Sigma$ to denote the set of symbols $\{\langle a \mid a \in \Sigma\}$, and $\Sigma\rangle$ to denote the set of symbols $\{a\rangle \mid a \in \Sigma\}$. Then, given an alphabet Σ, define the *tagged alphabet* $\hat{\Sigma}$ to be the set $\Sigma \cup \langle\Sigma \cup \Sigma\rangle$. 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, $\hat{n} = nw\_w(n)$ 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.

For Figure 1, assuming all positions are labeled with the same symbol a, the tagged words corresponding to the two nested sequences are $a\langle aa\langle aaaa\rangle a\rangle a$, and $a\rangle\langle aa\rangle a\rangle\langle aa\langle aa$.

Since we allow calls and returns to be pending, every word over the tagged alphabet $\hat{\Sigma}$ corresponds to a nested word. This correspondence is captured by the following lemma:

Lemma 1 *The transformation $nw\_w : NW(\Sigma) \mapsto \hat{\Sigma}^*$ is a bijection.*

The inverse of nw_w is a transformation function that maps words over $\hat{\Sigma}$ to nested words over Σ, and will be denoted $w\_nw : \hat{\Sigma}^* \mapsto NW(\Sigma)$. This one-to-one correspondence shows that:

```
global int x;
main() {
    x = 3 ;
    if P x=1;
}
bool P() {
  local int y=0;
  x = y;
  if (x==0) return 1
  else return 0;
}
```

Figure 2: Example program

Proposition 1 (Counting nested sequences) *There are exactly 3^ℓ distinct matching relations of length ℓ, and the number of nested words of length ℓ over an alphabet Σ 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.

Using 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. We list a few operations below.

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 pending calls of the first with the pending returns of the latter. Concatenation extends to languages of nested words, and leads to the operation of Kleene-$*$ over languages.

Given a nested word $n = w\_nw(b_1 \dots b_\ell)$, its *subword* from position i to j, denoted $n[i, j]$, is the nested word $w\_nw(b_i \dots b_j)$, 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 nesting edge will change to a pending call edge; and in the subword that starts after i and ends after j, this nesting edge will change to a pending return edge. 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 $1 \leq i \leq \ell$, concatenating the prefix $n[1, i]$ and the suffix $n[i + 1, \ell]$ gives back n.

For example, for the first sequence in Figure 1, the prefix of first five positions is the nested word corresponding to $a\langle aa\langle aa$, and has two pending calls; the suffix of last four positions is the nested word corresponding to $aa\rangle a\rangle a$, and has two pending returns.

2.3 Examples

In this section, we give potential applications where data has the dual linear-hierarchical structure, and can naturally be modeled using nested words.

2.3.1 Executions of sequential structured programs

In the linear-time semantics of programs, execution of a program is typically modeled as a word. We propose to augment this linear structure with nesting edges from entries to exits of program blocks.

As a simple example, consider the program of Figure 2. For program analysis, the choice of Σ depends on the desired level of detail. As an example, suppose we are interested in tracking read/write accesses to the global program variable x, and also whether these accesses belong to the same context. Then, we can choose the following set of symbols: *rd* to denote a read access to x, *wr* to denote a write access to x, *en* to denote beginning of a new scope (such as a call to a function or a procedure), *ex* to denote the ending of the current scope, and *sk* to denote all other actions of the program. Note that in any structured programming

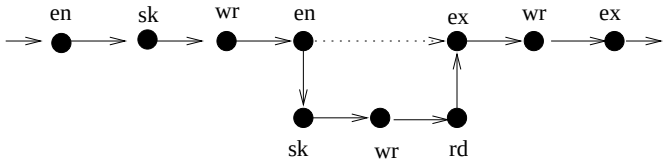

Figure 3: Sample program execution

language, in a given execution, there is a natural nested matching of the symbols *en* and *ex*. Figure 3 shows a sample execution of the program modeled as a nested word.

The main benefit is that using nesting edges one can skip call to a procedure entirely, and continue to trace a local path through the calling procedure. Consider the property that "if a procedure writes to x then it later reads x." This requires keeping track of the context. If we were to model executions as words, the set of executions satisfying this property would be a context-free language of words, and hence, is not specifiable in classical temporal logics. Soon we will see that when we model executions as nested words, the set of executions satisfying this property is a regular language of nested words, and is amenable to algorithmic verification.

2.3.2 Annotated linguistic data

Linguistic research and NLP technologies use large repositories (corpora) of annotated text and speech data. The data has a natural linear order (the order of words in a sentence) while the annotation adds a hierarchical structure. Traditionally, the result is represented as an ordered tree, but can equally be represented as a nested word. For illustration, we use an example from [BCD$^+$06]. The sentence is

I saw the old man with a dog today

The linguistic categorization parses the sentence into following categories: S (sentence), VP (verb phrase), NP (noun phrase), PP (prepositional phrase), Det (determiner), Adj (adjective), N (noun), Prep (proposition), and V (verb). The parsed sentence is given by the tagged word of Figure 4. The call and return positions are tagged with the syntactic categories, while internal positions spell out the original sentence. In the figure, we label each internal position with a word, but this can be a sequence of internal positions, each labeled with a character. Since matching calls and returns have the same symbol labeling them, the symbol is shown on the connecting nesting edge.

To verify hypotheses, linguists need to ask fairly complex queries over such corpora. An example, again from [BCD$^+$06] is "find all sentences with verb phrases in which a noun *follows* a verb which is a *child* of the verb phrase". Here, *follows* means in the linear order of the original sentence, and *child* refers to the hierarchical structure imparted by parsing. The sentence in Figure 4 has this property because "man" (and "dog") follows "saw". For such queries that refer to both hierarchical and linear structure, representation using nested words, as opposed to classical trees, has succinctness benefits as discussed in Section 7.

2.3.3 XML documents

XML documents can be interpreted as nested words: the linear structure corresponds to the sequence of text characters, and the hierarchical structure is given by the matching of open- and close-tag constructs. Traditionally, trees and automata on unranked trees are used in the study of XML (see [Nev02, Lib05] for recent surveys). However, if one is interested in the linear ordering of all the leaves (or all the nodes), then representation using nested words is beneficial. Indeed, the SAX representation of XML documents coincides with the tagged word encoding of nested words. The linear structure is also useful while processing XML documents in streaming applications.

To explain the correspondence between nested words and XML documents, let us revisit the parsed sentence of Figure 4. The same structure can be represented as an XML document as shown in Figure 5.

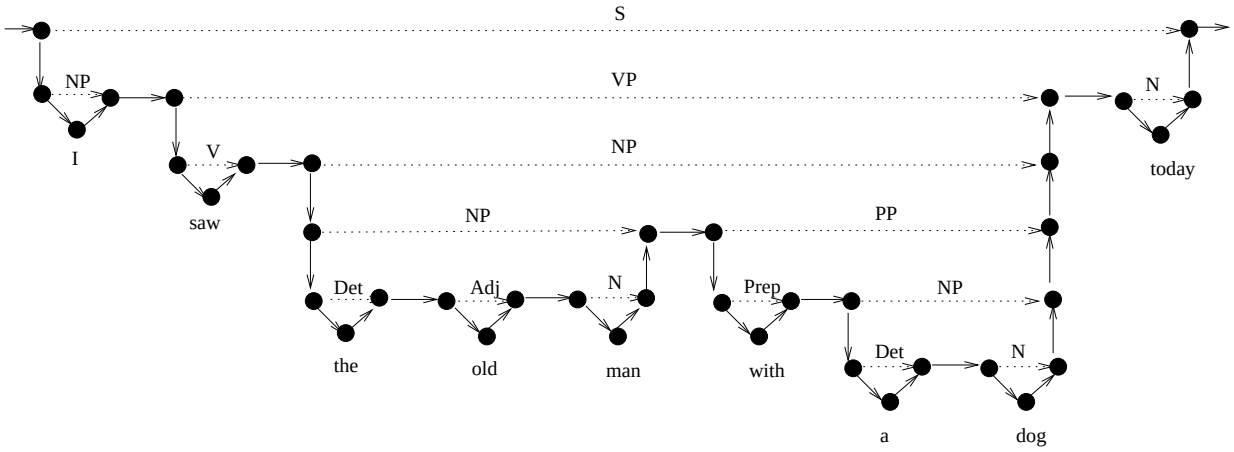

Figure 4: Parsed sentence as a nested word

Instead of developing the connection between XML and nested words in a formal way, we rely on the already well-understood connection between XML and unranked ordered forests, and give precise translations between such forests and nested words in Section 7.

3 Regular Languages of Nested Words

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 $(Q, q_0, Q_f, P, p_0, P_f, \delta_c, \delta_i, \delta_r)$ consisting of

- a finite set of (linear) states Q,
- an initial (linear) state $q_0 \in Q$,
- a set of (linear) final states $Q_f \subseteq Q$,
- a finite set of hierarchical states P,
- an initial hierarchical state $p_0 \in P$,
- a set of hierarchical final states $P_f \subseteq P$,
- a call-transition function $\delta_c : Q \times \Sigma \mapsto Q \times P$,
- an internal-transition function $\delta_i : Q \times \Sigma \mapsto Q$, and
- a return-transition function $\delta_r : Q \times P \times \Sigma \mapsto Q$.

The automaton A starts in the initial state, and reads the nested word from left to right according to the linear order. 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 hierarchical state along the outgoing nesting edge also. At a return, the new state is determined based on the states propagated along the linear edge as well as along the incoming nesting edge. The pending nesting edges incident upon pending returns are labeled with the initial hierarchical state. The run is accepting if the final linear state is accepting, and if the hierarchical states propagated along pending nesting edges from pending calls are also accepting.

```
<S>
  <NP> I
  </NP>
  <VP>
    <V> saw
    </V>
    <NP>
      <NP>
         <Det> the
         </Det>
         <Adj> old
         </Adj>
         <N> man
         </N>
      </NP>
      <PP>
         <Prep> with
         </Prep>
         <NP>
            <Det>  a
            </Det>
            <N>  dog
            </N>
         </NP>
      </PP>
    </NP>
  </VP>
  <N>  today
  </N>
</S>
```

Figure 5: XML representation of parsed sentence

Formally, a *run* r of the automaton A over a nested word $n = (a_1 \dots a_\ell, \rightsquigarrow)$ is a sequence $q_i \in Q$, for $0 \leq i \leq \ell$, of states corresponding to linear edges starting with the initial state q_0, and a sequence $p_i \in P$, for calls i, of states corresponding to nesting edges, such that for each position $1 \leq i \leq \ell$,

- if i is a call, then $\delta_c(q_{i-1}, a_i) = (q_i, p_i)$;
- if i is an internal, then $\delta_i(q_{i-1}, a_i) = q_i$;
- if i is a return with call-predecessor j, then $\delta_r(q_{i-1}, p_j, a_i) = q_i$, and if i is a pending return, then $\delta_r(q_{i-1}, p_0, a_i) = q_i$.

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, $q_\ell \in Q_f$ and for pending calls i, $p_i \in P_f$.

The language $L(A)$ of a nested-word automaton A is the set of nested words it accepts. We define the notion of regularity using acceptance by finite-state automata:

> A language L of nested words over Σ is *regular* if there exists a nested word automaton A over Σ such that $L = L(A)$.

To illustrate the definition, let us consider an example. Suppose $\Sigma = \{0, 1\}$. Consider the language L of nested words n such that every subword starting at a call and ending at a matching return contains an even

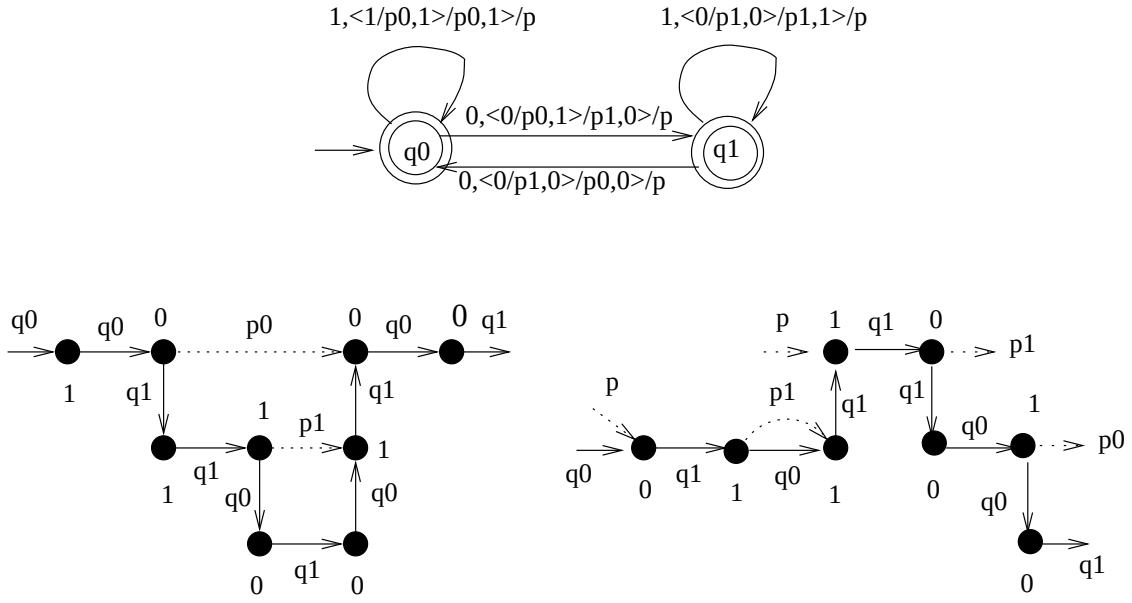

Figure 6: Example of an NWA and its runs

number of 0-labeled positions. That is, whenever $1 \leq i \leq j \leq \ell$ and $i \rightsquigarrow j$, $|\{k \mid i \leq k \leq j \text{ and } a_k = 0\}|$ is even. We will give an NWA whose language is L.

We use the standard convention for drawing automata as graphs over (linear) states. A call transition $\delta_c(q, a) = (q', p)$ is denoted by an edge from q to q' labeled with $\langle a/p$, and a return transition $\delta_r(q, p, a) = q'$ is denoted by an edge from q to q' labeled with $a\rangle/p$. To avoid cluttering, we allow the transition functions to be partial. In such a case, assume that the missing transitions go to the implicit "error" state q_e such that q_e is not a final state, and all transitions from q_e go to q_e.

The desired NWA is shown in Figure 6. It has 3 states q_0, q_1, and q_e (not shown). The state q_0 is initial, and q_0, q_1 are final. It has 3 hierarchical states p, p_0, p_1, of which p is initial, and p_0, p_1 are final. The state q_0 means that the number of 0-labeled positions since the last unmatched call is even, and state q_1 means that this number is odd. Upon a call, this information is propagated along the nesting edge, while the new linear state reflects the parity count starting at this new call. For example, in state q_1, while processing a call, the hierarchical state on the nesting edge is p_1, and the new linear state is q_0/q_1 depending on whether the call is labeled 1/0. Upon a return, if it is a matched return, then the current count must be even, and the state is retrieved along the nesting edge. For example, in state q_1, if the current return is matched, then the return must be labeled 0 (if return is labeled 1, then the corresponding transition is missing in the figure, so the automaton will enter the error state and reject), and the new state is set to q_0/q_1 depending on whether the hierarchical state on the nesting edge is p_0/p_1. Unmatched returns, indicated by the hierarchical state on the incoming nesting edge being p, are treated like internal positions.

The runs of this automaton on two nested word are also shown in Figure 6. Both words are accepted.

One can view nested word automata as graph automata over the nested sequence of linear and hierarchical edges: 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. Labels on edges with unspecified sources (the initial linear edge and nesting edges into pending calls) need to satisfy initialization constraints, and labels on edges with unspecified destination (the linear edge out of last position and nesting edges from pending calls) need to satisfy acceptance constraints.

3.2 Equivalent Definitions

In this section, we first describe some alternate ways of describing the acceptance of nested words by NWAs, and then, some restrictions on the definition of NWAs without sacrificing expressiveness.

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, $\delta_c(q,a) = (\delta_c^l(q,a), \delta_c^h(q,a))$.

For a nested word n, let $1 \leq i_1 < i_2 \cdots < i_k \leq \ell$ be all the pending call positions in n. Then, the sequence $p_{i_1} \dots p_{i_n} q_\ell$ in P^*Q is the *frontier* of the run of the automaton A on n, where each p_{i_j} is the hierarchical state labeling the pending nesting edge from call position i_j, and q_ℓ is the last linear state of the run. The frontier of the run at a position i is the frontier of the run over the prefix $n[1,i]$. The frontier of a run carries all the information of the prefix read so far, namely, the last linear state and the hierarchical states labeling all the nesting edges from calls that are pending at this position. In fact, we can define the behavior of the automaton using only frontiers. The initial frontier is q_0. Suppose the current frontier is $p_1 \dots p_k q$, and the automaton reads a symbol a. If the current position is an internal, the new frontier is $p_1 \dots p_k \delta_i(q,a)$. If the current position is a call, then the new frontier is $p_1 \dots p_k \delta_c^h(q,a) \delta_c^l(q,a)$. If the current position is a return, then if $k > 0$ then the new frontier is $p_1 \dots p_{k-1} \delta_r(q, p_k, a)$, and if $k = 0$, then the new frontier is $\delta_r(q, p_0, a)$. The automaton accepts a word if the final frontier is in $P_f^* Q_f$.

The definition of nested-word automata can be restricted in several ways without sacrificing the expressiveness. Our notion of acceptance requires the last linear state to be final and all pending hierarchical states to be final. However, acceptance using only final linear states is adequate. A nested word automaton $A = (Q, q_0, Q_f, P, p_0, P_f, \delta_c, \delta_i, \delta_r)$ is said to be *linearly-accepting* if $P_f = P$.

Theorem 1 (Linear acceptance) *Given a nested word automaton A, one can effectively construct a linearly-accepting NWA B such that $L(B) = L(A)$ and B has twice as many states as A.*

Proof. Consider an NWA $A = (Q, q_0, Q_f, P, p_0, P_f, \delta_c, \delta_i, \delta_r)$. The automaton B remembers, in addition to the state of A, a bit that indicates whether the acceptance requires a matching return. This bit is set to 1 whenever a non-final hierarchical state is propagated along the nesting edge. The desired automaton B is $(Q \times \{0,1\}, (q_0, 0), Q_f \times \{0\}, P \times \{0,1\}, P_0 \times \{0\}, P \times \{0,1\}, \delta_c', \delta_i', \delta_r')$. The internal transition function is given by $\delta_i'((q,x),a) = (\delta_i(q,a), x)$. The call transition function is given by $\delta_c'((q,x),a) = ((\delta_c^l(q,a), y), (\delta_c^h(q,a), x))$, where $y = 0$ iff $x = 0$ and $\delta_c^h(q,a) \in P_f$. The return transition function is given by $\delta_r'((q,x),(p,y),a) = (\delta_r(q,p,a), y)$.

For a nested word n with k pending calls, the frontier of the run of A on n is $p_1 \dots p_k q$ iff the frontier of the run of B on n is $(p_1, 0), (p_2, x_1) \dots (p_k, x_{k-1})(q, x_k)$ with $x_i = 1$ iff $p_j \in P_f$ for all $j \leq i$. This claim can be proved by induction on the length of n, and implies that the languages of the two automata are the same. □

We can further assume that the hierarchical states are implicitly specified: the set P of hierarchical states equals the set Q of linear states; the initial hierarchical state equals the initial state q_0, and the current state is propagated along the nesting edge at calls. A linearly-accepting nested word automaton $A = (Q, q_0, Q_f, P, p_0, P, \delta_c, \delta_i, \delta_r)$ is said to be *weakly-hierarchical* if $P = Q$, $p_0 = q_0$, and for all states q and symbols a, $\delta_c^h(q,a) = q$. A weakly-hierarchical nested word automaton then can be represented as $(Q, q_0, Q_f, \delta_c^l : Q \times \Sigma \mapsto Q, \delta_i : Q \times \Sigma \mapsto Q, \delta_r : Q \times Q \times \Sigma \mapsto Q)$. Weakly-hierarchical NWAs can capture all regular languages:

Theorem 2 (Weakly-hierarchical automata) *Given a nested word automaton A with s linear states over Σ, one can effectively construct a weakly-hierarchical NWA B with $2s|\Sigma|$ states such that $L(B) = L(A)$.*

Proof. We know that an NWA can be transformed into a linearly accepting one by doubling the states. Consider a linearly-accepting NWA $A = (Q, q_0, Q_f, P, p_0, \delta_c, \delta_i, \delta_r)$. The weakly-hierarchical automaton B remembers, in addition to the state of A, the symbol labeling the innermost pending call for 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 $(Q \times \Sigma, (q_0, a_0), Q_f \times \Sigma, \delta'_c, \delta'_i, \delta'_r)$ (here a_0 is some arbitrarily chosen symbol in Σ). The internal transition function is given by $\delta'_i((q, a), b) = (\delta_i(q, b), a)$. At a call labeled b, the automaton in state (q, a) transitions to $(\delta^l_c(q, b), b)$. At a return labeled c, the automaton in state (q, a), if the state propagated along the nesting edge is (q', b), moves to state $(\delta_r(q, \delta^h_c(q', a), c), b)$. □

3.3 Nondeterministic Automata

Nondeterministic NWAs can have multiple initial states, and at every position, can have multiple choices for updating the state.

A nondeterministic nested word automaton A over Σ has

- a finite set of (linear) states Q,
- a set of (linear) initial states $Q_0 \subseteq Q$,
- a set of (linear) final states $Q_f \subseteq Q$,
- a finite set of hierarchical states P,
- a set of initial hierarchical states $P_0 \subseteq P$,
- a set of final hierarchical states $P_f \subseteq P$,
- a call-transition relation $\delta_c \subseteq Q \times \Sigma \times Q \times P$,
- an internal-transition relation $\delta_i \subseteq Q \times \Sigma \times Q$, and
- a return-transition relation $\delta_r \subseteq Q \times P \times \Sigma \times Q$.

A *run* r of the nondeterministic automaton A over a nested word $n = (a_1 \dots a_\ell, \rightsquigarrow)$ is a sequence $q_i \in Q$, for $0 \leq i \leq \ell$, of states corresponding to linear edges, and a sequence $p_i \in P$, for calls i, of hierarchical states corresponding to nesting edges, such that $q_0 \in Q_0$, and for each position $1 \leq i \leq \ell$,

- if i is a call, then $(q_{i-1}, a_i, q_i, p_i) \in \delta_c$;
- if i is an internal, then $(q_{i-1}, a_i, q_i) \in \delta_i$;
- if i is a matched return with call-predecessor j then $(q_{i-1}, p_j, a_i, q_i) \in \delta_r$, and if i is a pending return then $(q_{i-1}, p_0, a_i, q_i) \in \delta_r$ for some $p_0 \in P_0$.

The run is accepting if $q_\ell \in Q_f$ and for all pending calls i, $p_i \in P_f$. The automaton A accepts the nested word n if A has some accepting run over n. The language $L(A)$ is the set of nested words it accepts.

We now show that nondeterministic automata are no more expressive than the deterministic ones. The determinization construction is a generalization of the classical determinization of nondeterministic word automata. We assume linear acceptance: we can transform any nondeterministic NWA into one that is linearly-accepting by doubling the states as in the proof of Theorem 1.

Theorem 3 (Determinization) *Given a nondeterministic linearly-accepting NWA A, one can effectively construct a deterministic linearly-accepting NWA B such that $L(B) = L(A)$. Moreover, if A has s_l linear states and s_h hierarchical states, then B has $2^{s_l s_h}$ linear states and $2^{s_h^2}$ hierarchical states.*

Proof. Let L be accepted by a nondeterministic linearly-accepting NWA $A = (Q, Q_0, Q_f, P, P_0, \delta_c, \delta_i, \delta_r)$. Given a nested word n, A can have multiple runs over n. Thus, at any position, the state of B needs to keep track of all possible states of A, as in case of classical subset construction for determinization of nondeterministic word automata. However, keeping only a set of states of A is not enough: at a return position, while combining linear states along the incoming linear edge with hierarchical states along the incoming nesting edge, B needs to figure which pairs of states belong to the same run. This can be achieved by keeping a set of pairs of states as follows.

- The states of B are $Q' = 2^{P \times Q}$.
- The initial state is the set of pairs of the form (p, q) such that $p \in P_0$ and $q \in Q_0$.
- A state $S \in Q'$ is accepting iff it contains a pair of the form (p, q) with $q \in Q_f$.
- The hierarchical states of B are $P' = 2^{P \times P}$.
- The initial hierarchical state is the set of pairs of the form (p, p') such that $p, p' \in P_0$.
- The call-transition function δ'_c is given by: for $S \in Q'$ and $a \in \Sigma$, $\delta'_c(S, a) = (S_l, S_h)$, where S_l consists of pairs (p', q') such that there exists $(p, q) \in S$ and a call transition $(q, a, q', p') \in \delta_c$; and S_h consists of pairs (p, p') such that there exists $(p, q) \in S$ and a call transition $(q, a, q', p') \in \delta_c$.
- The internal-transition function δ'_i is given by: for $S \in Q'$ and $a \in \Sigma$, $\delta'_i(S, a)$ consists of pairs (p, q') such that there exists $(p, q) \in S$ and an internal transition $(q, a, q') \in \delta_i$.
- The return-transition function δ'_r is given by: for $S_l \in Q'$ and $S_h \in P'$ and $a \in \Sigma$, $\delta'_r(S_l, S_h, a)$ consists of pairs (p, q') such that there exists $(p, p') \in S_h$ and $(p', q) \in S_l$ and a return transition $(q, p', a, q') \in \delta_r$.

Consider a nested word n with k pending calls. Let the frontier of the unique run of B over n be $S_1 \dots S_k S$. Then, the automaton A has a run with frontier $p_1 \dots p_k q$ over n iff for some $p_0 \in P_0$, $(q, p_k) \in S_k$ and $(p_i, p_{i+1}) \in S_i$ for $0 \leq i < k$. This claim can be proved by induction on the length of the nested word n. It follows that both automata accept the same set of nested words. □

Recall that a nondeterministic word automaton with s states can be transformed into a deterministic one with 2^s states. The determinization construction above requires keeping track of set of pairs of states, and as the following lower bound shows, this is really needed.

Theorem 4 (Succinctness of nondeterminism) *There exists a family L_s, $s \geq 1$, of regular languages of nested words such that each L_s is accepted by a nondeterministic NWA with $O(s)$ states, but every deterministic NWA accepting L_s must have 2^{s^2} states.*

Proof. Let $\Sigma = \{a, b, c\}$. Consider $s = 2^k$. Consider the language L that contains words of the form, for some $u, v \in (a + b)^k$,

$$\langle c\ ((a+b)^* c (a+b)^* cc)^* u\, c\, v\, cc ((a+b)^* c (a+b)^* cc)^* v\, c \rangle\ u$$

Intuitively, the constraint says that the word must end with the suffix $v\, c\rangle\ u$, where u and v are two k-bit strings such that the subsequence $u\, c\, v\, cc$ must have appeared before.

Consider a deterministic NWA accepting L. The words in L have only one nesting edge, and all begin with the same call symbol. Hence, the NWA has no information to propagate across the nesting edge, and behaves essentially like a standard word automaton. As the automaton reads the word from left to right every pair of successive k-bit strings are potential candidates for u and v. A deterministic automaton needs to remember, for each such pair, if it has occurred or not. Formally, we say that two nested words n and n' in $L' = \langle c\ ((a+b)^* c (a+b)^* cc)^*$ are equivalent iff for every pair of words $u, v \in (a+b)^k$, the word $u\, c\, v\, cc$ appears as a subword of n iff it appears as a subword of n'. Since there are s^2 pairs of words $u, v \in (a+b)^k$, the number of equivalence classes of L' by this relation is 2^{s^2}. It is easy to check that if A is a deterministic NWA for L, and n and n' are two inequivalent words in L', then the linear states of A after reading n and n' must be distinct. This implies that every deterministic NWA for L must have at least 2^{s^2} states.

There is a nondeterministic automaton with $O(s)$ states to accept L. We give the essence of the construction. The automaton guesses a word $u \in (a+b)^k$, and sends this guess across linear as well as hierarchical edges. That is, the initial state, on reading a call position labeled c, splits into (q_u, p_u), for every $u \in (a+b)^k$. The state q_u skips over a word in $((a+b)^* c (a+b)^* cc)^*$, and nondeterministically decides that what follows is the desired subword $u\, c\, v\, cc$. For this, it first needs to check that it reads a word that matches the guessed

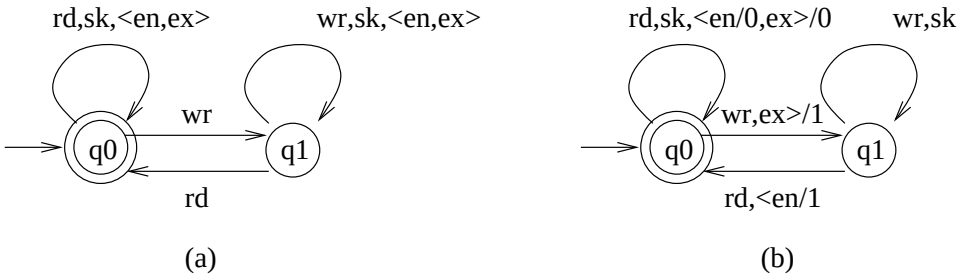

Figure 7: Program requirements as NWAs

u, and then, it remembers $v \in (a+b)^k$ that follows in the state, leading to a state q'_v. The state q'_v skips over a word in $((a+b)^*c(a+b)^*cc)^*$, and nondeterministically decides that what follows must be the word $v\,c\rangle$. If this check succeeds, then at the return position, the state p_u is retrieved along the nesting edge, and checks that what follows matches u. This can be formalized using states that consist of a mode and a word over $\{a, b\}$ of length at most k, where the number modes is a small constant. The resulting automaton has $O(s)$ states. □

3.4 Examples

In this section, we outline the application of nested word automata for program verification and document processing.

3.4.1 NWAs in Program Analysis

In the context of software verification, a popular paradigm relies on *data abstraction*, where the data in a program is abstracted using a finite set of boolean variables that stand for predicates on the data-space [BMMR01, HJM$^+$02]. The resulting models hence have finite-state but stack-based control flow (see Boolean programs [BR00] and recursive state machines [ABE$^+$05] as concrete instances of pushdown models of programs). For example, abstraction of the program in Figure 2 will replace integer variables x and y with boolean variables that keep track of the values of the predicates `x==0` and `y==0`. The abstraction introduces nondeterminism, and the resulting program is an overapproximation of the original program.

Given a program A with boolean variables, we can view A as a nondeterministic generator of nested words in the following manner. We choose an observation alphabet Σ depending upon the goal of verification. A state of the NWA records the line number and the values of all program variables in scope. Internal transitions correspond to execution of a statement within a procedure. Call transitions correspond to calling of a procedure: the updated linear state reflects the control in the called procedure, and the hierarchical state passed along the nesting edge records the return line number along with the values of local variables in the calling procedure. Return transitions combine the returned value, and the updates to global variables, from the called procedure, with the information stored in the incoming hierarchical state to compute the new state in the calling procedure. Thus, we can associate a regular language $L(A)$ of nested words with a program with boolean variables.

The requirements of a program can also be described as regular languages of nested words. Verification corresponds to language inclusion: do all nested words generated by a program satisfy the specification? Thus, verification reduces to decision problems for NWAs.

For sample requirements, consider the example of Figure 3. Suppose we want to specify that writes to x are followed by some read of x. We will consider three variations of this requirement.

First, suppose we want to specify that a symbol *wr* is followed by *rd*, without any reference to the procedural context. This can be captured by standard word automata, and also by NWAs. Figure 7 (a) shows the 2-state NWA for the requirement. In this example, hierarchical states are not really used: assume

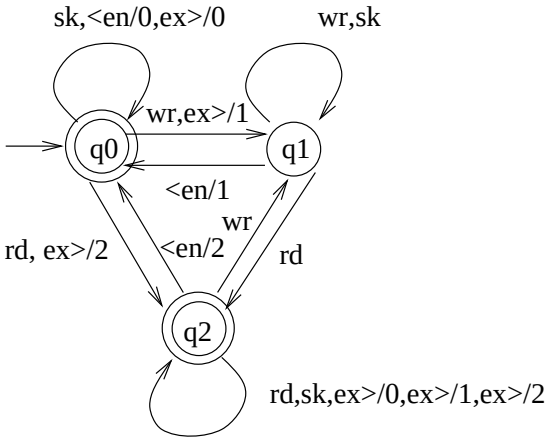

Figure 8: Context-bounded program requirement

that there is a single hierarchical state $\perp$, which is also initial, and is implicitly used in all call and return transitions.

Now suppose, we want to specify that if a procedure writes to x, then the same procedure should read it before it returns. That is, between every pair of matching entry and exit, along the local path obtained deleting every enclosed well-matched subword from an entry to an exit, every *wr* is followed by *rd*. Viewed as a property of words, this is not a regular language, and thus, not expressible in the specification languages supported by existing software model checkers such as SLAM [BR00] and BLAST [HJM$^+$02]. However, over nested words, this can easily be specified using an NWA, see Figure 7 (b). The initial state is q_0, and has no pending obligations, and is the only final state. The hierarchical states are $\{0, 1\}$, where 0 is the initial state. The state q_1 means that along the local path of the current scope, a write-access has been encountered with no following read access. While processing the call, the automaton remembers the current state by propagating 0 or 1 along the nesting edge, and starts checking the requirement for the called procedure by transitioning to the initial state q_0. While processing internal read/write symbols, it updates the state as in the finite-state word automaton of case (a). At a return, if the current state is q_0 (meaning the current context satisfies the desired requirement), it restores the state of the calling context. Note that there are no return transitions from the state q_1, and this means that if a return position is encountered while in state q_1, the automaton implicitly goes to an error state rejecting the input word.

Finally, suppose we want to specify that if a procedure writes to x, then the variable is read before the procedure returns, but either by this procedure or by one of the (transitively) called procedures. That is, along every global path sandwiched between a pair of matching entry and exit, every *wr* is followed by *rd*. This requirement is again not expressible using classical word automata. Figure 8 shows the corresponding NWA. State q_2 means that a read has been encountered, and this is different from the initial state q_0, since a read in the called procedure can be used to satisfy the pending obligation of the calling procedure. There are 3 hierarchical states 0,1,2 corresponding to the three linear states, and the current state is propagated along the nesting edge when processing a call. As before, in state q_0, while processing a return, the state of the calling context is restored; in state q_1, since the current context has unmet obligations, processing a return leads to rejection. While processing a return in the state q_2, the new state is q_2 irrespective of the state retrieved along the nesting edge.

3.4.2 NWAs for document processing

Since finite word automata are NWAs, classical word query languages such as regular expressions can be compiled into NWAs. As we will show in Section 7, different forms of tree automata are also NWAs.

As an illustrative example of a query, let us revisit the query "find all sentences with verb phrases

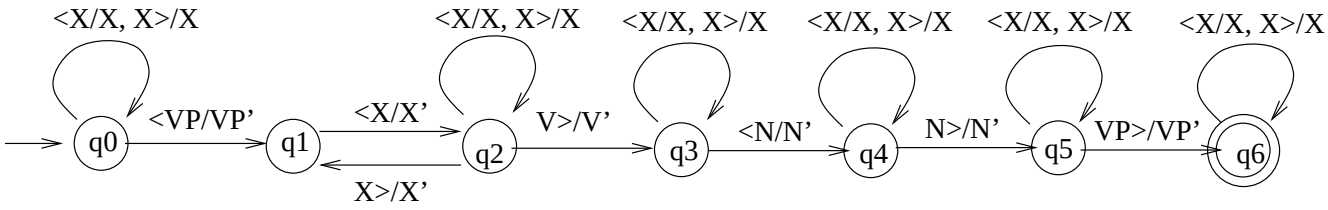

Figure 9: NWA for the linguistic query

in which a noun *follows* a verb which is a *child* of the verb phrase" discussed in Section 2.3.2. For this query, internal positions are not relevant, so we will assume that the alphabet consists of the tags $\{S, VP, NP, PP, Det, Adj, N, Prep, V\}$ corresponding to the various syntactic categories, and the input word has only call and return positions. The nondeterministic automaton is shown in Figure 9. The set of hierarchical states contains the dummy initial state $\perp$, and for each tag X, there are two symbols X and X'. The set of final hierarchical states is empty. Since (1) there are no return transitions if the state on the incoming hierarchical edge is $\perp$, (2) there can be no pending calls as no hierarchical state is final, and (3) every call transition on tag X labels the hierarchical edge with either X or X', and every return transition on tag X requires the label on incoming hierarchical edge to be X or X', the automaton enforces the requirement that all the tags match properly. In Figure 9, X ranges over the set of tags (for example, q_0 has a call transition to itself for every tag X, with the corresponding hierarchical state being X).

The automaton guesses that the desired verb phrase follows by marking the corresponding hierarchical edge with VP' (transition from q_0 to q_1). The immediate children of this verb phrase are also marked using the primed versions of the tags. When a child verb is found the automaton is is in state q_3, and searches for noun phrase (again marked with the primed version). The transition from q_5 to the final state q_6 ensures that the desired pattern lies within the guessed verb phrase.

3.5 Closure Properties

The class of regular languages of nested words enjoys a variety of closure properties. We begin with the boolean operations.

Theorem 5 (Boolean closure) *If L_1 and L_2 are regular languages of nested words over Σ, then $L_1 \cup L_2$, $L_1 \cap L_2$, and $NW(\Sigma) \setminus L_1$ are also regular languages.*

Proof. Let $A_j = (Q^j, q_0^j, Q_f^j, P^j, p_0^j, \delta_c^j, \delta_i^j, \delta_r^j)$, for $j = 1, 2$, be a linearly-accepting NWA accepting L_j. Define the product of these two automata as follows. The set of linear states is $Q^1 \times Q^2$; the initial state is (q_0^1, q_0^2); the set of hierarchical states is $P_1 \times P_2$; and the initial hierarchical state is (p_0^1, p_0^2). The transition functions are defined in the obvious way; for example, the return-transition function δ_r of the product is given by $\delta_r((q_1, q_2), (p_1, p_2), a) = (\delta_r^1(q_1, p_1, a), \delta_r^2(q_2, p_2, a))$. Setting the set of final states to $Q_f^1 \times Q_f^2$ gives the intersection $L_1 \cap L_2$, while choosing $(Q_f^1 \times Q_2) \cup (Q_1 \times Q_f^2)$ as the set of final states gives the union $L_1 \cup L_2$.

For a linearly-accepting deterministic NWA, one can complement the language simply by complementing the set of linear final states: the complement of the linearly-accepting automaton $(Q, q_0, Q_f, P, p_0, \delta_c, \delta_i, \delta_r)$ is the linearly-accepting NWA $(Q, q_0, Q \setminus Q_f, P, p_0, \delta_c, \delta_i, \delta_r)$. □

We have already seen how the word encoding allows us to define word operations over nested words. We proceed to show that the regular languages are closed under such operations.

Theorem 6 (Concatenation closure) *If L_1 and L_2 are regular languages of nested words, then so are $L_1 \cdot L_2$ and L_1^*.*

Proof. Suppose we are given weakly-hierarchical NWAs A_1 and A_2, with disjoint state sets, accepting L_1 and L_2, respectively. We can design a nondeterministic NWA that accepts $L_1 \cdot L_2$ by guessing a split of the input word n into n_1 and n_2. The NWA simulates A_1, and at some point, instead of going to a final state of A_1, switches to the initial state of A_2. While simulating A_2, at a return, if the state labeling the incoming nesting edge is a state of A_1, then it is treated like the initial state of A_2.

A slightly more involved construction can be done to show closure under Kleene-$*$. Let $A = (Q, Q_0, Q_f, \delta_c^l, \delta_i, \delta_r)$ be a weakly-hierarchical nondeterministic NWA that accepts L. We build the automaton A^* as follows. A^* simulates A step by step, but when A changes its state to a final state, A^* can nondeterministically update its state to an initial state, and thus, restart A. Upon this switch, A^* must treat the unmatched nesting edges as if they are pending, and this requires tagging its state so that in a tagged state, at a return, the states labeling the incident nesting edges are ignored. More precisely, the state-space of A^* is $Q \uplus Q'$, and its initial and final states are Q'_0. Its transitions are as follows

(Internal) For each internal transition $(q, a, p) \in \delta_i$, A^* contains the internal transitions (q, a, p) and (q', a, p'), and if $p \in Q_f$, then the internal transitions (q, a, r') and (q', a, r') for each $r \in Q_0$.

(Call) For each (linear) call transition $(q, a, p) \in \delta_c^l$, A^* contains the call transitions (q, a, p) and (q', a, p), and if $p \in Q_f$, then the call transitions (q, a, r') and (q', a, r'), for each $r \in Q_0$.

(Return) For each return transition $(q, r, a, p) \in \delta_r$, A^* contains the return transitions (q, r, a, p) and (q, r', a, p'), and if $p \in Q_f$, then the return transitions (q, r, a, s') and (q, r', a, s'), for each $s \in Q_0$. For each return transition $(q, r, a, p) \in \delta_r$ with $r \in Q_0$, A^* contains the return transitions (q', s, a, p') for each $s \in Q \cup Q'$, and if $p \in Q_f$, also the return transitions (q', s, a, t') for each $s \in Q \cup Q'$ and $t \in Q_0$.

Note that from a tagged state, at a call, A^* propagates the tagged state along the nesting edge and an untagged state along the linear edge. It is easy to check that $L(A^*) = L^*$. □

Besides prefixes and suffixes, we will also consider reversal. *Reverse* of a nested word n is defined to be $w\_nw(b_\ell \dots b_2 b_1)$, where for each $1 \le i \le \ell$, $b_i = a_i$ 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 nesting edges.

Theorem 7 (Closure under word operations) *If L is a regular language of nested words then all the following languages are regular: the set of reversals of all the nested words in L; the set of all prefixes of all the nested words in L; the set of all suffixes of all the nested words in L.*

Proof. Consider a nondeterministic NWA $A = (Q, Q_0, Q_f, P, P_0, P_f, \delta_c \delta_i, \delta_r)$. Define A^R to be $(Q, Q_f, Q_0, P, P_f, P_0, \delta_c^R, \delta$ where $(q, a, q', p) \in \delta_c$ iff $(q', p, a, q) \in \delta_r^R$, $(q, p, a, q') \in \delta_r$ iff $(q', a, q, p) \in \delta_c^R$, and $(q, a, q') \in \delta_i$ iff $(q' a, q) \in \delta_i^R$. Thus, A^R is obtained by switching the roles of initial and final states for both linear and hierarchical components, reversing the internal transitions, and dualizing call and return transitions. It is easy to show that A^R accepts precisely the reversals of the nested words accepted by A.

For closure under prefixes, consider a weakly-hierarchical nondeterministic NWA $A = (Q, Q_0, Q_f, \delta_c^l, \delta_i, \delta_r)$. The automaton B has the following types of states: $(q, q', 1)$ if there exists a nested word n which takes A from state q to state $q' \in Q_f$; $(q, q', 2)$ if there exists a nested word n without any pending returns, which takes A from state q to state $q' \in Q_f$; $(q, q', 3)$ if there exists a well-matched nested word n which takes A from state q to state q'. Initial states of B are of the form $(q, q', 1)$ such that $q \in Q_0$ and $q' \in Q_f$. All states are final. The state of B keeps track the current state of A along with a target state where the run of A can end so that we are sure of existence of a suffix leading to a word in $L(A)$. Initially, the target state is required to be a final state, and this target is propagated along the run. At a call, B can either propagate the current target across the linear edge requiring that the current state can reach the target without using pending returns; or propagate the current target across the nesting edge, and across the linear edge, guess a new target state requiring that the current state can reach this target using a well-matched word. The third component of the state is used to keep track of the constraint on whether pending calls and/or returns are allowed. Note that the reachability information necessary for effectively constructing the automaton B can be computed using analysis techniques discussed in decision problems. Transitions of B are described below.

(Internal) For every internal transition $(q, a, p) \in \delta_i$, for $x = 1, 2, 3$, for every $q' \in Q$, if both (q, q', x) and (p, q', x) are states of B, then there is an internal transition $((q, q', x), a, (p, q', x))$.

(Call) Consider a linear call transition $(q, a, p) \in \delta_c^l$ and $q' \in Q$ and $x = 1, 2, 3$, such that (q, q', x) is a state of B. Then for every state r such that $(p, r, 3)$ is a state of B and there exists $b \in \Sigma$ and state $r' \in Q$ such that (r', q', x) is a state of B and $(r, q, b, r') \in \delta_r$, there is a call transition $((q, q', x), a, (p, r, 3))$. In addition, if $x = 1, 2$ and $(p, q', 2)$ is a state of B, then there is a call transition $((q, q', x), a, (p, q', 2))$.

(Return) For every return transition $(q, p, a, r) \in \delta_r$, for $x = 1, 2, 3$, for $q' \in Q$, if (p, q', x) and (r, q', x) are states of B, then there is a return transition $((q, q, 3), (p, q', x), a, (r, q', x))$. Also, for every return transition $(q, p, a, r) \in \delta_r$ with $p \in Q_0$, for every $q' \in Q_f$, if $(q, q', 1)$ and $(r, q', 1)$ and $(p, q', 1)$ are states of B then there is a return transition $((q, q', 1), (p, q', 1), a, (r, q', 1))$.

The automaton B accepts a nested word n iff there exists a nested word n' such that the concatenation of n and n' is accepted by A.

Closure under suffixes follows from the closure under prefixes and reversals. □

Finally, we consider language homomorphisms. For every symbol $a \in \hat{\Sigma}$, let $h(a)$ be a language nested words. We say that h respects nesting if for each $a \in \Sigma$, $h(a) \subseteq WNW(\Sigma)$, $h(\langle a) \subseteq \langle \Sigma \cdot WNW(\Sigma)$, and $h(a\rangle) \subseteq WNW(\Sigma) \cdot \Sigma\rangle$. That is, internal symbols get mapped to well-matched words, call symbols get mapped to well-matched words with an extra call symbol at the beginning, and return symbols get mapped to well-matched words with an extra return symbol at the end. Given a language L over $\hat{\Sigma}$, $h(L)$ consists of words w obtained from some word $w' \in L$ by replacing each letter a in the tagged word for w' by some word in $h(a)$. Nesting-respecting language homomorphisms can model a variety of operations such as renaming of symbols and tree operations such as replacing letters by well-matched words.

Theorem 8 (Homomorphism closure) *If L is a regular language of nested words over Σ, and h is a language homomorphism such that h respects nesting and for every $a \in \hat{\Sigma}$, $h(a)$ is a regular language of nested words, then $h(L)$ is regular.*

Proof. Let A be the NWA accepting L, and for each a, let B_a be the NWA for $h(a)$. The nondeterministic automaton B for $h(L)$ has states consisting of three components. The first keeps track of the state of A. The second remembers the current symbol $a \in \hat{\Sigma}$ of the word in L being guessed. The third component is a state of B_a. When this automaton B_a is in a final state, then the second component can be updated by nondeterministically guessing the next symbol b, updating the state of A accordingly, and setting the third component to the initial state of B_b. When b is a call symbol, we know that the first symbol of the word in $h(b)$ is a pending call, and we can propagate the state of A along the nesting edge, so that it can be retrieved correctly later to simulate the behavior of A at the matching return. □

4 Monadic Second Order Logic of Nested Words

We show that the monadic second order logic (MSO) of nested words has the same expressiveness as nested word automata. The vocabulary of nested sequences includes the linear successor and the matching relation $\rightsquigarrow$. In order to model pending edges, we will use two unary predicates `call` and `ret` corresponding to call and return positions.

Let us fix a countable set of first-order variables FV and a countable set of monadic second-order (set) variables SV. We denote by x, y, x', etc., elements in FV and by X, Y, X', etc., elements of SV.

The *monadic second-order logic of nested words* is given by the syntax:

$$\varphi := a(x) \mid X(x) \mid \texttt{call}(x) \mid \texttt{ret}(x) \mid x = y + 1 \mid x \rightsquigarrow y \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi,$$

where $a \in \Sigma$, $x, y \in FV$, and $X \in SV$.

The semantics is defined over nested words in a natural way. The first-order variables are interpreted over positions of the nested word, while set variables are interpreted over sets of positions. $a(x)$ holds if the symbol at the position interpreted for x is a, $\texttt{call}(x)$ holds if the position interpreted for x is a call, $x = y + 1$ holds if the position interpreted for y is (linear) next to the position interpreted for x, and $x \rightsquigarrow y$ holds if the positions x and y are related by a nesting edge. For example,

$$\forall x.(\ \texttt{call}(x)\ \rightarrow \exists y.\ x \rightsquigarrow y\ )$$

holds in a nested word iff it has no pending calls;

$$\forall x. \forall y.\ (a(x) \wedge x \rightsquigarrow y) \Rightarrow b(y)$$

holds in a nested word iff for every matched call labeled a, the corresponding return-successor is labeled b.

For a sentence φ (a formula with no free variables), the language it defines is the set of all nested words that satisfy φ. We show that the class of all nested-word languages defined by MSO sentences is exactly the regular nested-word languages.

Theorem 9 (MSO characterization) *A language L of nested words over Σ is regular iff there is an MSO sentence φ over Σ that defines L.*

Proof. The proof is similar to the proof that MSO over words defines the same class as that of regular word languages (see [Tho90]).

First we show that for any sentence φ, the set $L(\varphi)$ of satisfying models is regular. Let us assume that in all formulas, each variable is quantified at most once. Consider any *formula* $\psi(x_1, \ldots, x_m, X_1, \ldots, X_k)$ (i.e. with free variables $Z = \{x_1, \ldots, x_m, X_1, \ldots, X_k\}$). Then consider the alphabet Σ^Z consisting of pairs (a, V) such that $a \in \Sigma$ and $V : Z \mapsto \{0, 1\}$ is a valuation function. Then a nested word n' over Σ^Z encodes a nested word n along with a valuation for the variables (provided singleton variables get assigned to exactly one position). Let $L(\psi)$ denote the set of nested words n' over Σ^Z such that the underlying nested word n satisfies ψ under the valuation defined by n'. Then we show, by structural induction, that $L(\psi)$ is regular.

The property that first-order variables are assigned exactly once can be checked using the finite control of an NWA. The atomic formulas $X(x)$, $a(x)$ and $x = y + 1$ are easy to handle.

To handle the atomic formula $x \rightsquigarrow y$, we build a NWA that propagates, at every call position, the current symbol in Σ^Z onto the outgoing nesting edge. While reading a return labeled with (a, v) where v assigns y to 1, the automaton requires that the hierarchical state along the incoming nesting edge is of the form (a', v') such that v' assigns x to 1.

Disjunction and negation can be dealt with using the fact that NWAs are closed under union and complement. Also, existential quantification corresponds to restricting the valuation functions to exclude a variable and can be done by renaming the alphabet, which is a special kind of nesting-respecting language homomorphism.

For the converse, consider a weakly-hierarchical NWA $A = (Q, q_0, Q_f, \delta_c^l, \delta_i, \delta_r)$ where $Q = \{q_0, \ldots q_k\}$. The corresponding MSO formula will express that there is an accepting run of A on the input word and will be of the form $\exists X_0 \ldots \exists X_k\ \varphi$. Here X_i stands for the positions where the run is in state q_i. We can write conditions in φ that ensure that the variables X_i indeed define an accepting run. The clauses for initialization, acceptance, and consecution according to call and internal transition functions are straightforward. The only interesting detail here is to ensure that the run follows the return-transition function at return positions. The case for matched returns can be expressed by the formula:

$$\forall x\, \forall y\, \forall z\ \wedge_{i=0}^{k} \wedge_{j=0}^{k} \wedge_{a \in \Sigma}\ (\ z = y + 1 \wedge x \rightsquigarrow z \wedge X_j(x) \wedge X_i(y) \wedge a(z) \rightarrow X_{\delta_r(q_i, q_j, a)}(z))$$

□

5 Visibly Pushdown Languages of Words

5.1 Visibly Pushdown Automata

Given a language L of nested words over Σ, let $nw\_w(L)$ be the language of tagged words over $\hat{\Sigma}$ corresponding to the nested words in L. One can interpret a linearly-accepting nested word automaton $A = (Q, q_0, Q_f, P, p_0, \delta_c, \delta_i, \delta_r)$ as a pushdown word automaton $\hat{A}$ over $\hat{\Sigma}$ as follows. Assume without loss of generality that call transitions of A do not propagate p_0 on the nesting edge. The set of states of $\hat{A}$ is Q, with q_0 as the initial state, and acceptance is by final states given by Q_f. The set of stack symbols is P, and p_0 is the bottom stack symbol. The call transitions are push transitions: in state q, while reading $\langle a$, the automaton pushes $\delta_c^h(q, a)$ onto the stack, and updates state to $\delta_c^l(q, a)$. The internal transitions consume an input symbol in Σ without updating the stack. The return transitions are pop transitions: in state q, with p on top the stack, while reading a symbol $a\rangle$, the automaton pops the stack, provided $p \neq p_0$, and updates the state to $\delta_r(q, p, a)$. If the frontier of the run of A after reading a nested word n is $p_1 \dots p_k q$, then, after reading the tagged word $nw\_w(n)$, the pushdown automaton $\hat{A}$ will be in state q, and its stack will be $p_0 p_1 \dots p_k$, with p_k on top.

The readers familiar with pushdown automata may prefer to understand NWAs as a special case. We chose to present the definition of NWAs in Section 3.1 without explicit reference to a stack for two reasons. First, the definition of NWA is really guided by the shape of the input structures they process, and are thus closer to definitions of tree automata. Second, while a stack-based implementation is the most natural way to process the tagged word encoding a nested word, alternatives are possible if the entire nested word is stored in memory as a graph.

This leads to:

Proposition 2 (Regular nested-word languages as context-free word languages) *If L is a regular language of nested words over Σ then $nw\_w(L)$ is a context-free language of words over $\hat{\Sigma}$.*

Not all context-free languages over $\hat{\Sigma}$ correspond to regular languages of nested words. A (word) language L over $\hat{\Sigma}$ is said to be a *visibly pushdown language* (VPL) iff $w\_nw(L)$ is a regular language of nested words. In particular, $\{(\langle a)^k (b\rangle)^k \mid k \geq 0\}$ is a visibly pushdown language, but $\{a^k b^k \mid k \geq 0\}$ is a context-free language which is not a VPL.

The pushdown automaton $\hat{A}$ corresponding to an NWA A is of a special form: it pushes while reading symbols of the form $\langle a$, pops while reading symbols of the form $a\rangle$, and does not update the stack while reading symbols in Σ. We call such automata *visibly pushdown automata*. The height of the stack is determined by the input word, and equals the depth of the prefix read plus one (for the bottom of the stack). Visibly pushdown automata accept precisely the visibly pushdown languages. Since NWAs can be determinized, it follows that the VPLs is a subclass of *deterministic* context-free languages (DCFLs). Closure properties and decision problems for VPLs follow from corresponding properties of NWAs.

While visibly pushdown languages are a strict subclass of context-free languages, for every context-free language, we can associate a visibly pushdown language by projection in the following way.

Theorem 10 (Relation between CFLs and VPLs) *If L is a context-free language over Σ, then there exists a VPL L' over $\hat{\Sigma}$ such that $L = h(L')$, where h is the renaming function that maps symbols $\langle a$, a, and $a\rangle$, to a.*

Proof. Let A be a pushdown automaton over Σ and let us assume, without loss of generality, that on reading a symbol, A pushes or pops at most one stack symbol, and acceptance is defined using final states. Now consider the visibly pushdown automaton A' over $\hat{\Sigma}$ obtained by transforming A such that every transition on a that pushes onto the stack is transformed to a push transition on $\langle a$, transitions on a that pop the stack are changed to pop transitions on $a\rangle$ and the remaining a-transitions are left unchanged. Then a word $w = a_1 a_2 \dots a_\ell$ is accepted by A iff there is some augmentation w' of w, $w' = b_1 b_2 \dots b_\ell$, where each $b_i \in \{a_i, \langle a_i, a_i\rangle\}$, such that w' is accepted by A'. Thus A' accepts the words in $L(A)$ annotated with information on how A handles the stack. It follows that $L(A) = h(L(A'))$, where h is the renaming function that maps, for each $a \in \Sigma$, symbols $\langle a$, a, and $a\rangle$, to a. $\square$

5.2 Grammar-based Characterization

It is well known that context-free languages can be described either by pushdown automata or by context-free grammars. In this section, we identify a class of context-free grammars that corresponds to visibly pushdown languages.

A context-free grammar over an alphabet Σ is a tuple $G = (V, S, \mathit{Prod})$, where V is a finite set of variables, $S \in V$ is a start variable, and Prod is a finite set of productions of the form $X \to \alpha$ such that $X \in V$ and $\alpha \in (V \cup \Sigma)^*$. The semantics of the grammar G is defined by the derivation relation $\Rightarrow$ over $(V \cup \Sigma)^*$: for every production $X \to \alpha$ and for all words $\beta, \beta' \in (V \cup \Sigma)^*$, $\beta X \beta' \Rightarrow \beta\alpha\beta'$ holds. The language $L(G)$ of the grammar G consists of all words $w \in \Sigma^*$ such that $S \Rightarrow^* w$, that is, a word w over Σ is in the language of the grammar G iff it can be derived from the start variable S in one or more steps.

A context-free grammar $G = (V, S, \mathit{Prod})$ over $\hat{\Sigma}$ is a *visibly pushdown grammar* if the set V of variables is partitioned into two disjoint sets V^0 and V^1, such that all the productions are of one the following forms

- $X \to \varepsilon$ for $X \in V$;
- $X \to aY$ for $X, Y \in V$ and $a \in \hat{\Sigma}$ such that if $X \in V^0$ then $a \in \Sigma$ and $Y \in V^0$;
- $X \to \langle aYb\rangle Z$ for $X, Z \in V$ and $Y \in V^0$ and $a, b \in \Sigma$ such that if $X \in V^0$ then $Z \in V^0$.

The variables in V^0 derive only well-matched words where there is a one-to-one correspondence between calls and returns. The variables in V^1 derive words that can contain pending calls as well as pending returns. In the rule $X \to aY$, if a is a call or a return, then either it is unmatched or its matching return or call is not remembered, and the variable X must be in V^1. In the rule $X \to \langle aYb\rangle Z$, the positions corresponding to symbols a and b are the matching calls and returns, with a well-matched word, generated by $Y \in V^0$, sandwiched in between, and if X is required to be well-matched then that requirement propagates to Z.

Observe that the rule $X \to aY$ is right-linear, and is as in regular grammars. The rule $X \to \langle aYb\rangle Z$ requires a and b to be matching call and return symbols, and can be encoded by a visibly pushdown automaton that, while reading a, pushes the obligation that the matching return should be b, with Z to be subsequently expanded. This intuition can be made precise:

Theorem 11 (Visibly pushdown grammars) *A language L over Σ is a regular language of nested words iff the language $nw\_w(L)$ over $\hat{\Sigma}$ has a visibly pushdown grammar.*

Proof. Let $G = (V, S, \mathit{Prod})$ be a visibly pushdown grammar over $\hat{\Sigma}$. We build a nondeterministic NWA A_G that accepts $w\_nw(L(G))$ as follows. The set of states of A_G is V. The unique initial state is S. The set of hierarchical states is $\Sigma \times V$ along with an initial hierarchical state $\perp$. The transitions of A_G from a state X on a symbol a are as follows:

Internal: δ_i contains (X, a, Y) for each variable Y such that $X \to aY$ is a production of G.

Call: δ_c contains $(X, a, Y, \perp)$ for each variable Y such that $X \to \langle aY$ is a production of G; and $(X, a, Y, (b, Z))$ for each production $X \to \langle aYb\rangle Z$ of G.

Return: δ_r contains $(X, \perp, a, Y)$ for each variable Y such that $X \to a\rangle Y$ is a production of G; and if X is a nullable symbol (that is, $X \to \varepsilon$ is a production of G) and is in V^0, then for each variable Y, δ_r contains $(X, (a, Y), a, Y)$.

The first clause says that the automaton can update state from X to Y while processing an a-labeled internal position according to the rule $X \to aY$. The second clause says that while reading a call, to simulate the rule $X \to \langle aY$ (this can happen only when $X \in V^1$), the automaton propagates the initial state $\perp$ along the nesting edge, and updates the state to Y. To simulate the rule $X \to \langle aYb\rangle Z$, the automaton changes the state to Y while remembering the continuation of the rule by propagating the pair (b, Z) onto the nesting edge. The third clause handles returns. The return can be consumed using a rule $X \to a\rangle Y$ when X is in V^1. If the current state is nullable and in V^0, then the state along the nesting edge contains the required

continuation, and the symbol being read should be consistent with it. If neither of these conditions hold, then no transition is enabled, and the automaton will reject. The sole accepting hierarchical state is $\perp$ (which means that there is no requirement concerning matching return), and the linear accepting states are nullable variables X.

In the other direction, consider a linearly-accepting NWA $A = (Q, q_0, Q_f, P, p_0, \delta_c, \delta_i, \delta_r)$. We will construct a visibly pushdown grammar G_A that generates $nw\_w(L(A))$. For each state $q \in Q$, the set V^1 has two variables X_q and Y_q; and for every pair of (linear) states q, p, the set V^0 has a variable $Z_{q,p}$. Intuitively, the variable X_q says that the state is q and there are no pending call edges; the variable Y_q says that the state is q and no pending returns should be encountered; and the variable $Z_{q,p}$ says that the current state is q and the state just before the next pending return is required to be p. The start variable is X_{q_0}.

1. For each state q, there is a production $Z_{q,q} \to \varepsilon$, and if $q \in Q_F$, there are productions $X_q \to \varepsilon$ and $Y_q \to \varepsilon$.
2. For each symbol a and state q, let $p = \delta_i(q, a)$. There are productions $X_q \to aX_p$ and $Y_q \to aY_p$, and for each state q', there is a production $Z_{q,q'} \to aZ_{p,q'}$.
3. For symbols a, b, and states q, p, let $q' = \delta_c^l(q, a)$ and $p' = \delta_r(p, \delta_c^h(q, a), b)$. There are productions $X_q \to \langle aZ_{q',p}b\rangle X_{p'}$ and $Y_q \to \langle aZ_{q',p}b\rangle Y_{p'}$, and for every state r, there is a production $Z_{q,r} \to \langle aZ_{q',p}b\rangle Z_{p',r}$.
4. For each symbol a and state q, let $p = \delta_c^l(q, a)$. There are productions $X_q \to \langle aY_p$ and $Y_q \to \langle aY_p$.
5. For each symbol a and state q, let $p = \delta_r(q, p_0, a)$. There is a production $X_q \to a\rangle X_p$.

In any derivation starting from the start variable, the string contains only one trailing X or Y variable, which can be nullified by the first clause, provided the current state is accepting. The first clause allows nullifying a variable $Z_{q,q'}$ when the current state q is same as the target state q', forcing the next symbol to be a return. Clause 2 corresponds to processing internal positions consistent with the intended interpretation of the variables. Clause 3 captures summarization. In state q, while reading a call a, the automaton propagates $\delta_c^h(q, a)$ while updating its state to $q' = \delta_c^l(q, a)$ We guess the matching return symbol b and the state p just before reading this matching return. The well-matched word sandwiched between is generated by the variable $Z_{q',p}$, and takes the automaton from q' to p. The variable following the matching return b is consistent with the return transition that updates state p, using hierarchical state $\delta_c^h(q, a)$ along the nesting edge while reading b. The clause 4 corresponds to the guess that the call being read has no matching return, and hence, it suffices to remember the state along with the fact that no pending returns can be read by switching to the Y variables. The final clause allows processing of unmatched returns. □

Recall that a *bracketed language* consists of well-bracketed words of different types of parentheses (c.f. [GH67, HU79]). A parenthesis language is a bracketed language with only one kind of parentheses. Bracketed languages are special case of balanced grammars [BB02, BW04]. The original definition of balanced grammars considers productions of the form $X \to \langle aLa\rangle$, where L is a regular language over the nonterminals V. We present a simpler formulation that turns out to be equivalent.

A grammar $G = (V, S, Prod)$ is a *balanced grammar* if all the productions are of the form $X \to \varepsilon$ or $X \to \langle aYa\rangle Z$. Clearly, a balanced grammar is also a visibly pushdown grammar. In particular, the maximal parenthesis language—the Dyck language consisting of all well-bracketed words, denoted $Dyck(\Sigma)$, is generated by the grammar with sole variable S with productions $S \to \varepsilon$ and $S \to \langle aSa\rangle\, S$, for every $a \in \Sigma$. It is known that every context-free language is a homomorphism of the intersection of the Dyck language with a regular language (in contrast, Theorem 10 asserts that every CFL is a homomorphism of a VPL).

The table of Figure 5.2 summarizes and compares closure properties for CFLs, deterministic CFLs (DCFLs), VPLs, balanced languages, and regular languages.

6 Decision Problems

As we have already indicated, a nested word automaton can be interpreted as a pushdown automaton. The emptiness problem (given A, is $L(A) = \emptyset$?) and the membership problem (given A and a nested word n, is

	Closure under				
	Union	Intersection	Complement	Concat/Kleene-$*$	Prefixes/Suffixes
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	Yes
Balanced	Yes	Yes	No	Yes	No
VPL	Yes	Yes	Yes	Yes	Yes

Figure 10: Closure properties of classes of word languages

$n \in L(A)$?) for nested word automata are solvable in polynomial-time since we can reduce it to the emptiness and membership problems for pushdown automata. For these problems, A can be nondeterministic.

If the automaton A is *fixed*, then we can solve the membership problem in simultaneously linear time and linear space, as we can determinize A and simply simulate the word on A. In fact, this would be a *streaming* algorithm that uses at most space $O(d)$ where d is the *depth* of nesting of the input word. A streaming algorithm is one where the input must be read left-to-right, and can be read only once. Note that this result comes useful in type-checking streaming XML documents, as the depth of documents is often not large. When A is fixed, the result in [vBV83] exploits the visibly pushdown structure to solve the membership problem in logarithmic space, and [Dym88] shows that membership can be checked using boolean circuits of logarithmic depth. These results lead to:

Proposition 3 (Emptiness and membership) *The emptiness problem for nondeterministic nested word automata is decidable in time $O(|A|^3)$. The membership problem for nondeterministic nested word automata, given A and a nested word n of length ℓ, can be solved in time $O(|A|^3.\ell)$. When A is fixed, it is solvable (1) in time $O(\ell)$ and space $O(d)$ (where d is the depth of n) in a streaming setting; (2) in space $O(log\ \ell)$ and time $O(\ell^2.log\ \ell)$; and (3) by (uniform) Boolean circuits of depth $O(log\ \ell)$.*

The inclusion problem (and hence the equivalence problem) for nested word automata is decidable. Given A_1 and A_2, we can check $L(A_1) \subseteq A_2$ by checking if $L(A_1) \cap \overline{L(A_2)}$ is empty, since regular nested languages are effectively closed under complement and intersection. Note that if the automata are deterministic, then these checks are polynomial-time, and if the automata are nondeterministic, the checks require the determinization construction.

Theorem 12 (Universality and inclusion) *The universality problem and the inclusion problem for nondeterministic nested word automata are* Exptime*-complete.*

Proof. Decidability and membership in Exptime for inclusion hold because, given nondeterministic NWAs A_1 and A_2, we can take the complement of A_2 after determinizing it, take its intersection with A_1 and check for emptiness. Universality reduces to checking inclusion of the language of the fixed 1-state NWA A_1 accepting all nested words with the given NWA. We now show that universality is Exptime-hard for nondeterministic NWAs (hardness of inclusion follows by the above reduction).

The reduction is from the membership problem for alternating linear-space Turing machines (TM) and is similar to the proof in [BEM97] where it is shown that checking pushdown systems against linear temporal logic specifications is Exptime-hard.

Given an input word for such a fixed TM, a run of the TM on the word can be seen as a binary tree of configurations, where the branching is induced by the universal transitions. Each configuration can be encoded using $O(s)$ bits, where s is the length of the input word. Consider an infix traversal of this tree, where every configuration of the tree occurs twice: when it is reached from above for the first time, we write out the configuration and when we reach it again from its left child we write out the configuration in reverse. This encoding has the property that for any parent-child pair, there is a place along the encoding where the configuration at the parent and child appear consecutively. We then design, given an input word to the TM,

	Decision problems for automata		
	Emptiness	Universality/Equivalence	Inclusion
DFA	NLOGSPACE	NLOGSPACE	NLOGSPACE
NFA	NLOGSPACE	PSPACE	PSPACE
PDA	PTIME	Undecidable	Undecidable
DPDA	PTIME	Decidable	Undecidable
NWA	PTIME	PTIME	PTIME
Nondet NWA	PTIME	EXPTIME	EXPTIME

Figure 11: Summary of decision problems

a nondeterministic NWA that accepts a word n iff n is either a wrong encoding (i.e. does not correspond to a run of the TM on the input word) or n encodes a run that is not accepting. The NWA checks if the word satisfies the property that a configuration at a node is reversed when it is visited again using the nesting edges. The NWA can also guess nondeterministically a parent-child pair and check whether they correspond to a wrong evolution of the TM, using the finite-state control. Thus the NWA accepts all nested words iff the Turing machine does not accept the input. □

The table of Figure 6 summarizes and compares decision problems for various kinds of word and nested-word automata.

7 Relation to Tree Automata

In this section, we show that ordered trees, and more generally, hedges—sequences of ordered trees, can be naturally viewed as nested words, and existing versions of tree automata can be interpreted as nested word automata.

7.1 Hedges as Nested Words

Ordered trees and hedges 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 and the set $H(\Sigma)$ of hedges over an alphabet Σ is defined inductively:

1. ε is in $OT(\Sigma)$ and $H(\Sigma)$: this is the empty tree;
2. $t_1, \ldots t_k \in H(\Sigma)$, where $k \geq 1$ and each t_i is a nonempty tree in $OT(\Sigma)$: this corresponds to the hedge with k trees.
3. for $a \in \Sigma$ and $t \in H(\Sigma)$, $a(t)$ is in $OT(\Sigma)$ and $H(\Sigma)$: this represents the tree whose root is labeled a, and has children corresponding to the trees in the hedge t.

Consider the transformation $t\_w : H(\Sigma) \mapsto \hat{\Sigma}^*$ that encodes an ordered tree/hedge over Σ as a word over $\hat{\Sigma}$: $t\_w(\varepsilon) = \varepsilon$; $t\_w(t_1, \ldots t_k) = t\_w(t_1) \cdots t\_w(t_k)$; and $t\_w(a(t)) = \langle a\, t\_w(t)\, a \rangle$. This transformation can be viewed as a traversal of the hedge, 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. This is the standard representation of trees for streaming applications [SV02]. An a-labeled leaf corresponds to the word $\langle aa \rangle$, we will use $\langle a \rangle$ as its abbreviation.

The transformation $t\_nw : H(\Sigma) \mapsto NW(\Sigma)$ is the functional composition of t_w and w_nw. However, not all nested words correspond to hedges: a nested word $n = (a_1 \ldots a_\ell, \rightsquigarrow)$ is said to be a *hedge word* iff it has no internals, and for all $i \rightsquigarrow j$, $a_i = a_j$. A hedge word is a tree word if it is rooted (that is, $1 \rightsquigarrow \ell$ holds). We will denote the set of hedge words by $HW(\Sigma) \subseteq WNW(\Sigma)$, and the set of tree words by $TW(\Sigma) \subseteq HW(\Sigma)$. It is easy to see that hedge words correspond exactly to the Dyck words over $\hat{\Sigma}$ [BW04].

Proposition 4 (Encoding hedges) *The transformation $t\_nw : H(\Sigma) \mapsto NW(\Sigma)$ is a bijection between $H(\Sigma)$ and $HW(\Sigma)$ and a bijection between $OT(\Sigma)$ and $TW(\Sigma)$; and the composed mapping $t\_nw \cdot nw\_w$ is a bijection between $H(\Sigma)$ and $Dyck(\Sigma)$.*

The inverse of t_nw then is a transformation function that maps hedge/tree words to hedges/trees, and will be denoted nw_t. It is worth noting that a nested word automaton can easily check the conditions necessary for a nested word to correspond to a hedge word or a tree word.

Proposition 5 (Hedge and tree words) *The sets $HW(\Sigma)$ and $TW(\Sigma)$ are regular languages of nested words.*

7.2 Bottom-up Automata

A weakly-hierarchical nested word automaton $A = (Q, q_0, Q_f, \delta_c^l, \delta_i, \delta_r)$ is said to be *bottom-up* iff 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 is a pending call, 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 an 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 13 (Expressiveness of bottom-up automata) *Given a weakly-hierarchical NWA A with s states, one can effectively construct a weakly-hierarchical bottom-up NWA B with s^s states such that $L(A) \cap WNW(\Sigma) = L(B) \cap WNW(\Sigma)$.*

Proof. Let $A = (Q, q_0, Q_f, \delta_c^l, \delta_i, \delta_r)$ be a weakly-hierarchical NWA. A state of B is a function $f : Q \mapsto Q$. When a call is encountered, since B cannot use the current state, it simulates A for every possible state. Consider a nested word n and a position i for which the inner-most pending call is 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 $f(q_0) \in Q_f$. 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 nesting edge is g, then B updates its state to f' such that $f'(q) = \delta_r(f(g(q)), g(q), a)$. To complete the proof, one establishes that for a well-matched word n, A accepts n iff B accepts n. □

A variety of definitions of bottom-up tree automata have been considered in the literature. In the generic definition of a bottom-up automaton over unranked trees, the automaton has a finite set of states Q, an initial state q_0, a set of final states $Q_f \subseteq Q$, and a transition function $\delta : Q^* \times \Sigma \mapsto Q$. The run r of the automaton maps each ordered tree t to a state $r(t)$. For the empty tree ε, $r(\varepsilon)$ is the initial state q_0, and for a tree t with an a-labeled root and children $t_1 \ldots t_k$, $r(t)$ is $\delta(r(t_1) \cdots r(t_k), a)$. The automaton accepts a tree t if $r(t) \in Q_f$. The transition function must be specifiable by a finite-state automaton itself. The definition simplifies for binary trees, where the transition function maps $Q \times Q \times \Sigma$ to Q.

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 [BKMW01, MN05]. The only difference is that stepwise automata were defined to read only tree or hedge words, and process the symbol at a call when first encountered. That is, a stepwise bottom-up tree automaton is a bottom-up NWA on hedge words with the restriction that $\delta_r : Q \times Q \times \Sigma \mapsto Q$ does not depend on its third argument.

Proposition 6 (Bottom-up tree automata) *If $L \subseteq H(\Sigma)$ 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 [MN05], so succinctness gap of NWAs with respect to bottom-up NWAs carries over to these classes. We show the following exponential gap, using techniques developed for defining congruences for nested words [AKMV05]:

Theorem 14 (Succinctness gap for bottom-up automata) *There exists a family L_s, $s \geq 1$, of regular languages of tree words such that each L_s is accepted by an 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 $\langle a\langle b\rangle^m \langle aL^{i-1}\langle a\rangle L^{s-i}a\rangle a\rangle$, where $i = m \bmod s$.

First, we want to establish that there is a deterministic word automaton (and hence, also an NWA) with $O(s^2)$ states accepting L_s. The automaton can compute the value of $i = m \bmod s$ 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. □

Finally, let us revisit the grammar based characterization using visibly pushdown grammars. If we are restricting to hedge words, then we do not need the right-linear rules, and Dyck rules suffice. This leads to the following equivalence theorem. The equivalence of regular hedge languages and the original balanced languages (defined using rules of the form $X \to \langle aLa\rangle$, where L is a regular word language over nonterminals) implies that our definition of balanced grammars is equivalent to the original one [BW04].

Theorem 15 (Regular Hedge Languages and Balanced Languages) *Let $L \subseteq HW(\Sigma)$ be a set of hedge words over Σ. Then the following are equivalent*

1. *L is a regular nested-word language.*
2. *$nw\_t(L)$ is a regular hedge language.*
3. *$nw\_w(L)$ is a balanced language.*

Proof. We already know that over hedge words, NWAs have the same expressiveness as tree automata. From Theorem 11, we know that a balanced language, a special case of a visibly pushdown grammar, can be translated into an NWA. What remains to be shown is that while the translation from an NWA A to the grammar G_A can be done using only Dyck rules when $L(A)$ contains only hedge words. The translation is similar as in the proof of Theorem 11. Let $A = (Q, q_0, Q_f, P, \delta_c, \delta_r)$ be an NWA accepting only hedge words (initial hierarchical state, final hierarchical states, and internal transition relation are not used while processing hedge words). We need only variables $Z_{q,p}$ generating hedge words with the interpretation that the current state is q and the state before the next pending return is p. The start variables are of the $Z_{q_0,p}$ with $p \in Q_f$. For each state q, we have a production $Z_{q,q} \to \varepsilon$. For each symbol a and states p, q, q', we have a production $Z_{q,p} \to \langle aZ_{\delta_c^l(q,a),q'}a\rangle Z_{\delta_r(q',\delta_c^h(q,a),a),p}$. The grammar generates only those hedge words accepted by A, and has only Dyck rules. □

The relationship among various classes of languages is depicted in Figure 12.

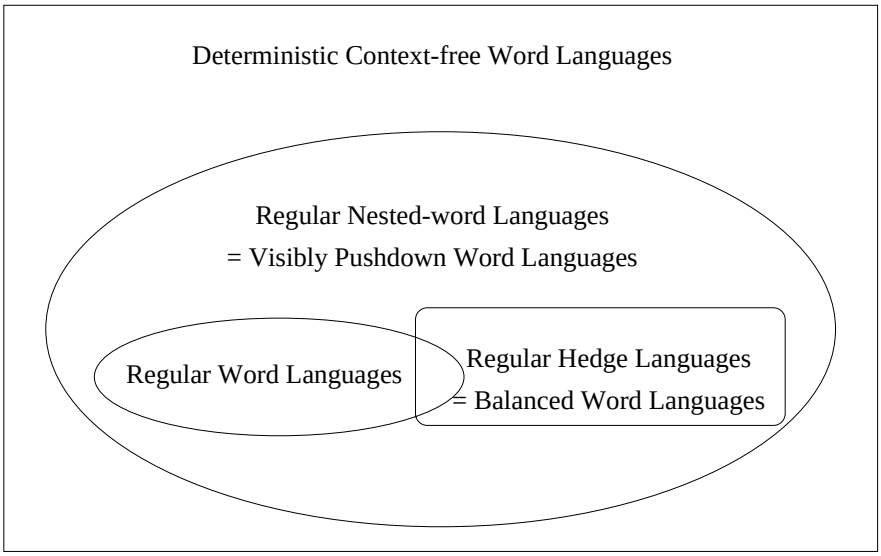

Figure 12: Relationship among languages classes

7.3 Top-down Automata

A nested word automaton at a return position joins the information flowing along the linear edge and the nesting edge. In this section, we study the impact of disallowing such a join. A top-down automaton, at a call, processes the subword upto the matching return and the suffix after the return independently. This can be formulated as a restriction of the return transition relation: the next state is based upon the state propagated along the nesting edge and the only information along the linear edge is whether the inside subword is accepted or not.

A *nondeterministic top-down nested word automaton* is a nondeterministic NWA $(Q, Q_0, Q_f, P, \delta_c, \delta_i, \delta_r)$ such that the return relation is specified by the hierarchical return relation $\delta_r^h \subseteq P \times \Sigma \times Q$: $(q, p, a, q') \in \delta_r$ iff $(p, a, q') \in \delta_r^h$ and $q \in Q_f$. The automaton is deterministic if there is only one initial state and choice of at most one transition given the current state and symbol.

Over tree words, the standard definition of top-down tree automata is essentially the same as our notion of top-down automata. A top-down tree automaton processing hedges consists of states Q, initial state q_0, final states $F \subseteq Q$, and a transition function $\delta : Q \times \Sigma \mapsto 2^{Q \times Q}$. The automaton starts in the initial state q_0 processing the root node of the left-most tree in the hedge t. While processing an a-labeled node in state q, it chooses a transition $(q_1, q_2) \in \delta(q, a)$, and sends a copy in state q_1 to the left-most child of the current node, and a copy in state q_2 to the right-sibling of the current node. If there is no child, then q_1 is processing the empty tree, and if there is no right sibling, q_2 is processing the empty tree. An empty tree is accepted in a state q iff $q \in F$. To accept the hedge t, all copies must accept. Such a top-down tree automaton is deterministic if $|\delta(q, a)| \leq 1$ for all states q and labels a.

Proposition 7 (Top-down tree automata) *If $L \subseteq H(\Sigma)$, 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 cannot be checked by a top-down automaton.

Corollary 1 (Deficiency of deterministic top-down automata) *Deterministic top-down nested word automata are strictly less expressive than nested word automata.*

Nondeterminism can be used to address this deficiency, provided we restrict attention to well-matched words.

Theorem 16 (Expressive completeness of nondeterministic top-down automata) *Given a nondeterministic NWA A with s states, one can effectively construct a nondeterministic NWA B with $O(s^2|\Sigma|)$ states such that $L(A) \cap WNW(\Sigma) = L(B) \cap WNW(\Sigma)$.*

Proof. Let $A = (Q, Q_0, Q_f, P, \delta_c, \delta_i, \delta_r)$ be an NWA. We can ignore initial and final hierarchical states, since we are interested only in well-matched words. For every pair (q, q') of states of A, B has a (linear) 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 hierarchical states of the form (q, q', a) to label nesting edges to mean that the symbol at the return is guessed to be a. The initial states are of the form (q, q') with $q \in Q_0$ and $q' \in Q_f$. States of the form (q, q) are accepting. For every internal transition (q, a, q') of A, for every p, B has an internal transition $((q, p), a, (q', p))$. For every call transition (q, a, q_l, q_h) of A, for every return transition (p, q_h, b, r), for every state q', B has a call transition $((q, q'), a, (q_l, p), (r, q', b))$. Note that here B is demanding a run from q_l to p on the inside subword, and the accepting condition ensures that this obligation is met. The hierarchical return transitions of B are of the form $((q, q', a), a, (q, q'))$, and ensure the consistency of the return symbol guessed at the call-predecessor with the symbol being read. □

7.4 Path Languages

The mix of top-down and bottom-up traversal in nested word automata can be better explained on unary trees. We will 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 $path(a_1 \dots a_\ell)$ is $w\_nw(\langle a_1 \dots \langle a_\ell a_\ell \rangle \dots a_1 \rangle)$. Note that for a word w, $path(w)$ is rooted and has depth $|w|$.

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 proposition follows from definitions:

Proposition 8 (Path languages) *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 17 (Bottom-up and top-down traversal of NWAs) *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 Proposition 8, the theorem follows. □

8 Nested ω-words

Automata over finite words are useful for specification and verification of safety properties. To be able to specify and verify liveness properties (for example, "every write is eventually followed by a read"), we need to consider infinite words. Consequently, we now consider extensions of the results in the previous sections to infinite words with a matching relation.

The definition of a matching relation over $\mathbb{N}$ as a subset of $(\mathbb{N} \cup \{-\infty\}) \times (\mathbb{N} \cup \{+\infty\})$ is a straightforward generalization of the definition of matching relation of length ℓ; the axioms stay the same. A *nested ω-word* over Σ is a pair $(a_1 a_2 \ldots, \rightsquigarrow)$, where $a_1 a_2 \ldots$ is an ω-word over Σ and $\rightsquigarrow$ is a matching relation over $\mathbb{N}$. The notions such as word encoding and operations generalize to nested ω-words in the obvious way.

In order to generalize NWAs to ω-automata over nested words, we need to define the notion of acceptance of an infinite run over nested ω-words, where the run consists of infinite sequences of linear and hierarchical states. Since there are multiple possibilities here, we begin with the simplest possible notion of acceptance: Büchi acceptance using linear states. Since Büchi word automata need to be nondeterministic to capture all ω-regular languages, we will consider nondeterministic Büchi automata over nested ω-words.

A *(linearly-accepting) nondeterministic Büchi nested word automaton* (BNWA) over an alphabet Σ consists of states Q, initial states $Q_0 \subseteq Q$, Büchi states $Q_f \subseteq Q$, hierarchical states P, initial hierarchical states $P_0 \subseteq P$, a call transition relation $\delta_c \subseteq Q \times \Sigma \times Q \times P$, an internal transition relation $\delta_i \subseteq Q \times \Sigma \times Q$, and a return transition relation $\delta_r \subseteq Q \times P \times \Sigma \times Q$. A run r of the BNWA A over a nested ω-word $n = (a_1 a_2 \ldots, \rightsquigarrow)$ is an infinite sequence $q_i \in Q$, for $i \geq 0$, of states corresponding to linear edges, and a sequence $p_i \in P$, for calls i, of hierarchical states corresponding to nesting edges, such that $q_0 \in Q_0$, and for each position $i \geq 1$, if i is a call then $(q_{i-1}, a_i, q_i, p_i) \in \delta_c$; if i is an internal then $(q_{i-1}, a_i, q_i) \in \delta_i$; if i is a matched return with call-predecessor j then $(q_{i-1}, p_j, a_i, q_i) \in \delta_r$, and if i is a pending return then $(q_{i-1}, p_0, a_i, q_i) \in \delta_r$ for some $p_0 \in P_0$. The run is accepting if $q_i \in Q_f$ for infinitely many indices $i \geq 0$. The automaton A accepts the nested ω-word n if A has some accepting run over n. The language $L(A)$ is the set of nested ω-words it accepts. A set L of nested ω-words is regular iff there is a BNWA A such that $L(A) = L$.

The class of regular languages of nested ω-words is closed under various operations. In particular:

Theorem 18 (Closure for ω-languages) *Let L_1 and L_2 be regular languages of nested ω-words over Σ. Then, $L_1 \cup L_2$ and $L_1 \cap L_2$ are also regular. Further, if L_3 is a regular language of nested words over Σ, then $L_3.L_1$ and $(L_3)^\omega$ are regular languages of nested ω-words. If h is a language homomorphism such that h respects nesting and for every $a \in \hat{\Sigma}$, $h(a)$ is a regular language of nested words, then $h(L_1)$ is a regular language of nested ω-words.*

Proof. The closure constructions for nondeterministic Büchi NWAs are similar to the ones in Theorem 5. For the case of intersection, we need to make sure that the Büchi states of both automata are visited infinitely often, and this can be done by adding a bit to the state as is done for intersection of Büchi word automata. The construction for L^ω is similar to the construction for L^* in Theorem 6. In this construction, for every transition leading to a state $q \in Q_F$, the state is nondeterministically reset to some initial state. To ensure membership in L^ω, we need to require that this reset happens infinitely often, and this can be captured by a Büchi condition after adding a bit to the state. The proof of closure under nesting-respecting homomorphisms is similar to the case of finite words. □

For automata over ω-words, it is known that Büchi acceptance condition cannot capture all regular languages if we restrict attention to deterministic automata. For deterministic automata over ω-words, adequate acceptance conditions include *parity condition* and *Muller condition*. For the nested case, it turns out that, for deterministic automata, no condition on the set of linear states repeating infinitely often can capture all regular languages.

A *(linearly-accepting) deterministic Muller nested word automaton* (MNWA) over an alphabet Σ consists of states Q, initial state $q_0 \in Q$, a Muller set $\mathcal{F} = \{F_1, \ldots F_k\}$ of accepting sets $F_i \subseteq Q$, hierarchical states P, initial hierarchical state $p_0 \subseteq P$, a call transition function $\delta_c : Q \times \Sigma \mapsto Q \times P$, an internal transition function $\delta_i : Q \times \Sigma \mapsto Q$, and a return transition function $\delta_r : Q \times P \times \Sigma \mapsto Q$. Given a nested ω-word n,

the unique run r of a MNWA A over n is defined as in case of deterministic NWAs. This run is accepting if the set $\{q \in Q \mid \text{for infinitely many indices } i \geq 0,\ q_i = q\} \in \mathcal{F}$.

It is easy to see that Muller acceptance condition does not increase expressiveness if nondeterminism is allowed. However, the classical determinization of Büchi automata fails in this case. Consider the language L_{repbdd} consisting of all nested ω-words on over the unary alphabet such that n has only finitely many pending calls (that is, there exists $k \in \mathbb{N}$ such that the depth of infinitely many positions is less than or equal to k). This property has been studied in the context of verification of pushdown programs, and is called "repeatedly bounded stack depth" in [CDT02].

Theorem 19 (Inadequacy of deterministic linear Muller acceptance) *The language L_{repbdd} of nested ω-words with only finitely many pending calls is accepted by a nondeterministic Büchi NWA but cannot be accepted by any deterministic Muller NWA.*

Proof. We can easily design a BNWA that accepts L_{repbdd}. The automaton nondeterministically chooses a position in the word, and checks that there are no subsequent pending calls. The check can be performed by tagging the state. The tag is propagated at internal positions. At calls, the tag is propagated only across nesting edges, and at returns the tag is retrieved from the incoming nesting edges. The check succeeds if a tagged state is encountered infinitely often.

Now to show that no MNWA can accept L_{repbdd}, assume the contrary and let $A = (Q, q_0, \mathcal{F}, P, p_0, \delta_c, \delta_i, \delta_r)$ be a deterministic Muller automaton that accepts L_{repbdd}. Let $G_1 = (Q, \rightarrow)$ be the summary-graph of A where $q \rightarrow q'$ iff there exists a well-matched nested word n that takes the automaton from q to q'. Note that if q is a state reachable by A on some nested word n', then there must be an outgoing edge from q in G_1 (if n is any well-matched nested word, then $n'n^\omega$ is in L_{repbdd}, and is accepted by the deterministic automaton A, and hence, there will be a summary edge from q corresponding n). Also, since concatenation of well-matched words is well-matched, G_1 is transitively closed.

Consider the strongly connected components (SCC) of G_1. A sink SCC of G_1 is a strongly connected component S' such that every edge $(q, q') \in G_1$, if $q \in S'$ then q' is also in S'.

Now let $G_2 = (Q, \Rightarrow)$ which is a super-graph of G_1 with additional *call-edges*: (q, q') is a call-edge if there is a transition from q to q' in A on a call. We now show:

> **Lemma:** There is a sink SCC S of G_1 and a state $q \in S$ reachable from the initial state q_0 in G_2 such that there is a cycle involving q in G_2 that includes a call-edge.

If the lemma is true, then we can show a contradiction. Consider a nested word that from q_0 reaches q using the summary edges and call-edges in G_2 and then loops forever in S. This word has only finitely many pending calls, and hence must be accepting and hence Q_S, the union of all states reachable using summary edges in S must be in the Muller set $\mathcal{F}$. Now consider another nested ω-word that takes A from q_0 to q but now follows the cycle that involves q. Along the cycle, some are well-matched words corresponding to summary edges and some are calls; note that there will be no returns that match these calls. If Q' is the set of states visited from going to q along the cycle, then we can show that $Q' \subseteq Q_S$ (after the cycle, if we feed enough returns to match the calls then we get a summary edge from q; however summary edges from q go only to S and hence the states seen must be a subset of Q_S). Now, consider the infinite word that first goes from q_0 to q, and then alternately takes the cycle in G_2 to reach q and takes the set of all possible summary edges in S to reach q again. This word has infinitely many pending calls, but the set of states seen infinitely often is Q_S and is hence accepted, a contradiction.

Now let us show the lemma. Note that from any state, one can take summary edges in G_1 to reach a sink SCC of G_1. Let us take summary edges from q_0 to reach a state q_1 in a sink SCC S_1 of G_1 and take the call edge from q_1 to reach q_1'. If $q_1' \in S_1$, we are done as we have a cycle from q_1 to q_1 using a call-edge. Otherwise take summary edges in G_1 from q_1' to reach a state q_2 in a sink SCC S_2. If $S_2 = S_1$, we are again done, else take a call-edge from q_2 and repeat till some sink SCC repeats. □

This raises the question: what is the appropriate acceptance condition for automata over infinite nested words? This was answered in [LMS04]. Given a nested ω-word n, a position i is a top-level position if it is not

within a nesting edge (that is, there are no positions $j < i < k$ with $j \rightsquigarrow k$). If the acceptance condition can examine the part of the run restricted to the top-level positions, then deterministic Muller (or deterministic Parity) condition suffices. More precisely, a *deterministic stair Muller NWA* is like a MNWA, but its run over a nested ω-word is accepting if the set $\{q \in Q \mid \text{for infinitely many } \textit{top-level} \text{ positions } i \geq 0,\ q_i = q\} \in \mathcal{F}$. Deterministic stair Muller NWAs can accept all regular languages of nested ω-words.

Closure of BNWAs under complementation, can be shown without resorting to determinization using the stair Muller acceptance condition.

Theorem 20 (Complementation for ω-languages) *The class of regular languages of nested ω-words is closed under complement: given a BNWA A with s states, one can construct a BNWA with $2^{O(s^2)}$ states accepting the complement of $L(A)$.*

Proof. Let $A = (Q, Q_0, Q_f, P, P_0, \delta_c, \delta_i, \delta_r)$ be a BNWA with s states over Σ. We can assume that there are transitions from every state on every letter, and that there is a run of A on every nested ω-word. Consider an ω-word $w \in \hat{\Sigma}^\omega$. Then w can be factored into finite words where we treat a segment of symbols starting at a call and ending at the matching return as a block. This factorization can be, say, of the form $w = a_1 a_2 w_1 a_3 w_2 w_3 a_4 \ldots$ where each w_i is a finite word over $\hat{\Sigma}$ that starts with a call and ends with the matching return. The symbols a_i can be calls, returns or internal symbols but if some a_i is a call, then a_j $(j > i)$ cannot be a return. Consider the following ω-word which can be seen as a pseudo-run of A on w: $w' = a_1 a_2 S_1 a_3 S_2 S_3 a_4 \ldots$ where each S_i is the set of all triples (q, q', f) where $q, q' \in Q$, $f \in \{0, 1\}$ such that there is some run of A on w_i starting at the state q and ending at the state q' and containing a state in Q_f iff $f = 1$.

Let $\mathcal{S}$ denote the set of all sets S where S contains triples of the form (q, q', f) where $q, q' \in Q$ and $f \in \{0, 1\}$; the summary edges used above hence are in $\mathcal{S}$. Then $PR = (\Sigma\rangle \cup \Sigma \cup \mathcal{S})^\omega \cup (\Sigma\rangle \cup \Sigma \cup \mathcal{S})^*.(\langle\Sigma \cup \Sigma \cup \mathcal{S})^\omega$ denotes the set of all possible pseudo-runs. We can now construct a nondeterministic Büchi word automaton $\mathcal{A}_R$ that accepts the set L_R of all *accepting pseudo-runs*; where a pseudo-run is accepting if there is a run of A that runs over the $\hat{\Sigma}$-segments of the word in the usual way, and on letters $S \in \mathcal{S}$, updates the state using a summary edge in S and either meets Q_f infinitely often or uses summary edges of the form $(q, q', 1)$ infinitely often. Note that an ω-word w is accepted by A iff the pseudo-run corresponding to w is accepting. Now we construct a deterministic Streett automaton $\mathcal{B}_R$ that accepts the *complement* of L_R. The automaton $\mathcal{A}_R$ has $O(s)$ states. Due to complexity of complementation, $\mathcal{B}_R$ has $2^{O(s \log\ s)}$ states and $O(s)$ accepting constraints [Tho90].

We now construct a nondeterministic Büchi NWA B that, on reading w, generates the pseudo-run of w online and checks whether it is in L_R. The factorization of w into segments is done nondeterministically and the summary edges are computed online using the nesting edges (as in the proof of determinization of NWAs on finite words). The states of this automaton B correspond to the summaries in $\mathcal{S}$, and thus, it has $2^{O(s^2)}$ states.

The desired automaton is the product of B and $\mathcal{B}_R$, and has $2^{O(s^2)}$ states and $O(s)$ Streett accepting constraints. The resulting automaton can be converted to a Büchi automaton and accepts the complement of $L(A)$, and has $2^{O(s^2)}$ states. □

We can also characterize the class of regular languages of nested ω-words using monadic second order logic which is now interpreted over infinite words, using the closure properties.

Theorem 21 (MSO characterization of ω-languages) *A language L of nested ω-words over Σ is regular iff there is an MSO sentence φ over Σ that defines L.*

Proof. The proof of Theorem 9 can be easily adopted to the infinite case.

To show that for a given MSO sentence, the set of its satisfying nested ω-words can be accepted by a Büchi NWA, we use the same encoding. Consider any formula $\psi(x_1, \ldots, x_m, X_1, \ldots, X_k)$, and consider the alphabet Σ^Z consisting of pairs (a, V) such that $a \in \Sigma$ and $V : Z \mapsto \{0, 1\}$ is a valuation function. Then a nested ω-word n' over Σ^Z encodes a nested ω-word n along with a valuation for the variables. Let $L(\psi)$ denote the set of nested words n' over Σ^Z such that the underlying nested word n satisfies ψ under the

valuation defined by n'. Then we show, by structural induction, that $L(\psi)$ is regular. The property that first-order variables are assigned exactly once can be checked using the finite control of a Büchi NWA and the acceptance condition. The atomic formulas are handled as before. Disjunction,negation, and existential quantification are handled using corresponding constructions for Büchi NWAs.

The translation from Büchi NWAs to MSO uses the same encoding for capturing the runs of the automaton by a formula. Only the conjunct corresponding to the acceptance requirement for the run needs to be modified. □

The emptiness problem ω-NWAs is decidable in polynomial time since they can be interpreted as pushdown automata over infinite words over $\hat{\Sigma}$ [BS92]. From our results it also follows that the universality and inclusion problems are EXPTIME-complete for nondeterministic Büchi NWAs: the upper bounds follow from the complexity of complementation, and lower bounds follow from the corresponding bounds for NWAs.

9 Related Work

Software Model Checking

Our work was motivated by the use of pushdown automata in software verification. It is worth noting that most of the algorithms for inter-procedural program analysis and context-free reachability compute summary edges between control locations to capture the computation of the called procedure (see, for example [SP81, RHS95]). The problem of checking regular requirements of pushdown models has been extensively studied in recent years leading to efficient implementations and applications to program analysis [RHS95, BEM97, BR00, ABE+05, HJM+02, EKS03, CW02]. Decision procedures for certain classes of non-regular properties already exist [JMT99, CW02, EKS03, CMM+04]. The idea of making calls and returns in a recursive program visible to the specification language for writing properties appears implicitly in [JMT99] which proposes a logic over stack contents to specify security constraints, and in [EKS03] which augments linear temporal logic with regular valuations over stack contents, and in our recent work on the temporal logic CARET that allows modalities for matching calls and returns [AEM04]. Also, properties expressing boundedness of stack, and repeatedly boundedness, have received a lot of attention recently [CDT02, BSW03].

Context-free Languages

There is an extensive literature on pushdown automata, context-free languages, deterministic pushdown automata, and context-free ω-languages (c.f. [ABB97]). The most related work is McNaughton's parenthesis languages with a decidable equivalence problem [McN67]. Knuth showed that parentheses languages are closed under union, intersection, and difference (but not under complementation, primarily because parenthesis languages can consist of only well parenthesized words), and it is decidable to check whether a context-free language is a parenthesis language [Knu67]. These proofs are grammar-based and complex, and connection to pushdown automata was not studied. Furthermore, parenthesis languages are a strict subclass of visibly pushdown languages, even when restricted to languages of well-bracketed words. Recently, balanced grammars are defined as a generalization of parenthesis languages by allowing several kinds of parentheses and regular languages in the right hand sides of productions [BB02]. It turns out that this class of languages is also a strict subclass of VPLs. The class of visibly pushdown languages, was also considered implicitly in papers related to parsing *input-driven languages* [vBV83, Dym88]. Input-driven languages are precisely visibly pushdown languages (the stack operations are *driven* by the input). However, the papers considered only the membership problem for these languages (namely showing that membership is easier for these languages than for general context-free languages) and did not systematically study the *class* of languages defined by such automata.

It has been observed that propositional dynamic logic can be augmented with some restricted class of context-free languages, and *simple-minded* pushdown automata, which may be viewed as a restricted class of VPAs, have been proposed to explain the phenomenon [HKT00].

There is a logical characterization of context free languages using quantifications over matchings [LST94], and Theorem 10 follows from that result.

Tree Automata

There is a rich literature on tree automata, and we used [Sch07, CDG+02] for our research. Besides classical top-down and bottom-up automata over binary trees, stepwise bottom-up tree automata for processing unranked ordered trees [MN05, BKMW01] are the most relevant to this paper. The connection between balanced grammars and regular hedge languages has been explored [BB02, BW04]. Many of the constructions for nested word automata can be traced to the corresponding constructions for tree automata. Deterministic word automata have been also used for stream processing of XML documents [GMOS03], 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. Pushdown automata have been used in various ways for streaming algorithms for querying XML data. For instance, [GS03] defines XPush machines, a particular form deterministic pushdown automata developed for processing multiple queries.

10 Conclusions

We have proposed a new model of nested words that allows capturing linear and hierarchical structure simultaneously. Both words and ordered trees are special cases of nested words, and nested words support both word and tree operations. Automata over nested words lead to a robust class of languages with appealing theoretical properties. Linear encodings of nested words gives the class of visibly pushdown languages, and this class lies between the parenthesis languages and deterministic context-free languages. Over trees, nested word automata combine top-down and bottom-up traversals, and are exponentially more succinct than existing definitions of tree automata.

This theory offers a way of extending the expressiveness of specification languages supported in model checking and program analysis tools: instead of modeling a boolean program as a context-free language of words and checking regular properties, one can model both the program and the specification as regular languages of nested words. More generally, pushdown automata can be replaced by NWAs, provided the pushdown automaton is only a consumer, rather than a producer, of the matching relation.

Given that NWAs can be more succinct than standard tree automata, and have same complexity of decision problems as that of standard tree automata, NWAs can potentially replace tree automata in some applications such as streaming algorithms for query processing. We need to explore if compiling existing XML query languages into nested word automata reduces query processing time in practice.

Acknowledgments

We thank Marcelo Arenas, Pablo Barcelo, Swarat Chaudhuri, Kousha Etessami, Neil Immerman, Viraj Kumar, Leonid Libkin, Christof Löding, Val Tannen, Mahesh Viswanathan, and Mihalis Yannakakis for fruitful discussions related to this paper. This research was partially supported by ARO URI award DAAD19-01-1-0473 and NSF awards CCR-0306382 and CPA-0541149.

References

[ABB97] J. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In *Handbook of Formal Languages*, volume 1, pages 111–174. Springer, 1997.

[ABE+05] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languages and Systems*, 27(4):786–818, 2005.

[AEM04] R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04: Tenth International Conference on Tools and Algorithms for the Construction and Analysis of Software*, LNCS 2988, pages 467–481. Springer, 2004.

[AKMV05] 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.

[Alu07] R. Alur. Marrying words and trees. In *Proceedings of the 26th ACM Symposium on Principles of Database Systems*, pages 233–242, 2007.

[AM04] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 202–211, 2004.

[AM06] R. Alur and P. Madhusudan. Adding nesting structure to words. In *Developments in Language Theory*, LNCS 4036, pages 1–13. Springer, 2006.

[BB02] J. Berstel and L. Boasson. Balanced grammars and their languages. In *Formal and Natural Computing: Essays Dedicated to Grzegorz Rozenberg*, LNCS 2300, pages 3–25. Springer, 2002.

[BCD+06] S. Bird, Y. Chen, S.B. Davidson, H. Lee, and Y. Zheng. Designing and evaluating an XPath dialect for linguistic queries. In *Proceedings of the 22nd International Conference on Data Engineering*, page 52, 2006.

[BEM97] A. Boujjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Applications to model checking. In *CONCUR'97: Concurrency Theory, Eighth International Conference*, LNCS 1243, pages 135–150. Springer, 1997.

[BKMW01] 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.

[BMMR01] T. Ball, R. Majumdar, T.D. Millstein, and S.K. Rajamani. Automatic predicate abstraction of C programs. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 203–213, 2001.

[BR00] T. Ball and S. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN 2000 Workshop on Model Checking of Software*, LNCS 1885, pages 113–130. Springer, 2000.

[BS92] O. Burkart and B. Steffen. Model checking for context-free processes. In *CONCUR'92: Concurrency Theory, Third International Conference*, LNCS 630, pages 123–137. Springer, 1992.

[BSW03] A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd International Conference*, LNCS 2914, pages 88–99. Springer, 2003.

[BW04] A. Brüggermann-Klein and D. Wood. Balanced context-free grammars, hedge grammars and pushdown caterpillar automata. In *Proceedings of the Extreme Markup Languages*, 2004.

[CDG+02] 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.

[CDT02] T. Cachat, J. Duparc, and W. Thomas. Solving pushdown games with a Σ_3 winning condition. In *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic, CSL 2002*, LNCS 2471, pages 322–336. Springer, 2002.

[CMM+04] K. Chatterjee, D. Ma, R. Majumdar, T. Zhao, T.A. Henzinger, and J. Palsberg. Stack size analysis for interrupt driven programs. *Information and Computation*, 194(2):144–174, 2004.

[CW02] H. Chen and D. Wagner. MOPS: An infrastructure for examining security properties of software. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 235–244, 2002.

[Dym88] P. Dymond. Input-driven languages are in *log n* depth. *Information Processing Letters*, 26(5):247–250, 1988.

[EKS03] J. Esparza, A. Kucera, and S. S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003.

[GH67] S. Ginsburg and M.A. Harrison. Bracketed context-free languages. *Journal of Computer and System Sciences*, 1:1–23, 1967.

[GMOS03] T.J. 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*, LNCS 2572, pages 173–189. Springer, 2003.

[GS03] A. Gupta and D. Suciu. Stream processing of XPath queries with predicates. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 419–430, 2003.

[HJM+02] T.A. Henzinger, R. Jhala, R. Majumdar, G.C. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. In *CAV 02: Proc. of 14th Conf. on Computer Aided Verification*, LNCS 2404, pages 526–538. Springer, 2002.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[JMT99] T. Jensen, D. Le Metayer, and T. Thorn. Verification of control flow based security properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 89–103, 1999.

[Knu67] D.E. Knuth. A characterization of parenthesis languages. *Information and Control*, 11(3):269–289, 1967.

[Lib05] L. Libkin. Logics for unranked trees: An overview. In *Automata, Languages and Programming, 32nd International Colloquium, Proceedings*, LNCS 3580, pages 35–50. Springer, 2005.

[LMS04] 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.

[LST94] A. Lautemann, T. Schwentick, and D. Thérien. Logics for context-free languages. In *Proceedings of Computer Science Logic, 8th International Workshop, CSL 94*, LNCS 933, pages 205–216. Springer, 1994.

[McN67] R. McNaughton. Parenthesis grammars. *Journal of the ACM*, 14(3):490–500, 1967.

[MN05] 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.

[Nev02] F. Neven. Automata, logic, and XML. In *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic, CSL 2002*, LNCS 2471, pages 2–26. Springer, 2002.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.

[RHS95] T. Reps, S. Horwitz, and S. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 49–61, 1995.

[Sch07] T. Schwentick. Automata for XML – A survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

[SP81] M. Sharir and A. Pnueli. Two approaches to inter-procedural data-flow analysis. In *Program flow analysis: Theory and applications*. Prentice Hall, 1981.

[SV02] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pages 53–64, 2002.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, 1990.

[vBV83] B. von Braunmühl and R. Verbeek. Input-driven languages are recognized in *log n* space. In *Fundamentals of Computation Theory, Proceedings*, LNCS 158, pages 40–51. Springer, 1983.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.