Colored Nested Words*

Rajeev Alur and Dana Fisman

Department of Computer and Information Sciences,
University of Pennsylvania, Philadelphia PA, USA

This is the full version of the LATA'16 paper.

Abstract. Nested words allow modeling of linear and hierarchical structure in data, and nested word automata are special kinds of pushdown automata whose push/pop actions are directed by the hierarchical structure in the input nested word. The resulting class of regular languages of nested words has many appealing theoretical properties, and has found many applications, including model checking of procedural programs. In the nested word model, the hierarchical matching of open- and close-tags must be properly nested, and this is not the case, for instance, in program executions in presence of exceptions. This limitation of nested words narrows its model checking applications to programs with no exceptions.
We introduce the model of *colored nested words* which allows such hierarchical structures with mismatches. We say that a language of colored nested words is *regular* if the language obtained by inserting the missing closing tags is a well-colored regular language of nested words. We define an automata model that accepts regular languages of colored nested words. These automata can execute restricted forms of ε-pop transitions. We provide an equivalent grammar characterization and show that the class of regular languages of colored nested words has the same appealing closure and decidability properties as nested words, thus removing the restriction of programs to be exception-free in order to be amenable for model checking, via the nested words paradigm.

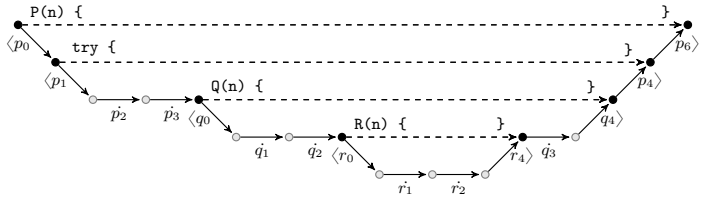
1 Introduction

Nested words, introduced in [4], are a data model capturing both a linear ordering and a hierarchically nested matching of items. Examples for data with both of these characteristics include executions of structured programs, annotated linguistic data, and documents in marked-up languages such as XML. While *regular languages of nested words* allow capturing of more expressive structure than traditional words, they retain all the good properties of regular languages. In particular, deterministic nested word automata are as expressive as their non-deterministic counterparts; the class is closed under the following operations: union, intersection, complementation, concatenation, Kleene-*, prefixes

* This research was supported by US NSF grant CCF-1138996.

```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:    try {                q1:    y = n / 2       r1:    z = n-1
p2:       x = 2*n           q2:    y = R(y)        r2:    if (z<0)
p3:       x = Q(x)          q3:    y = y+1         r3:       throw 0
p4:    }                    q4:    return y }      r4:    return z }
p5:    catch (int) {}
p6:    return x }
```

(a) A procedural program.



(b) An illustration of an execution of the code of the program where function calls are captured hierarchically.

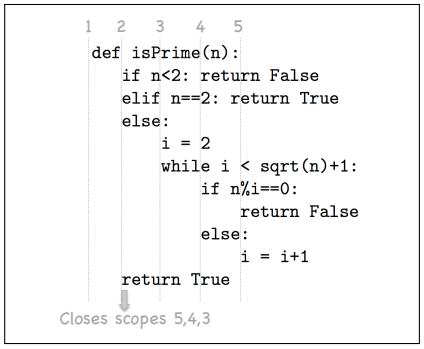
Fig. 1.

and language homomorphism; and the following problems are decidable: emptiness, membership, language inclusion and language equivalence.

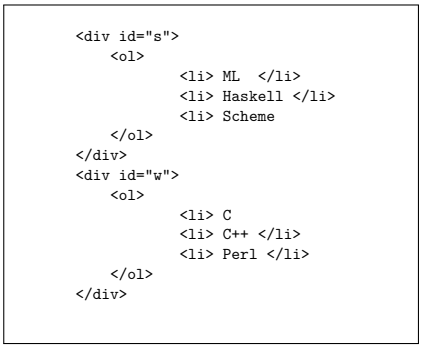
Many algorithms for problems concerning such data can be formalized and solved using constructions for basic operations and algorithms for decision problems. This fact led the way to many interesting applications and tools. Two prominent areas are XML processing (see e.g. [16,9,15]) and model checking of procedural programs (see e.g. [1,2,7,21,3,10]). By modeling executions of structured programs as nested words, one can algorithmically verify/refute various aspects of program correctness. Consider for instance, the program in Fig. 1a. An example execution is illustrated in Fig. 1b. Each step of the execution is mapped to the program counter line, and in addition, function calls create hierarchical connections to their respective returns. In the illustrations calls are depicted with down arrows, returns with up arrows, and internal code with horizontal arrows.

Nested words can be represented by graphs as in Fig. 1b or via an implicit representation using words over an alphabet $\Sigma \times \{\langle, \cdot, \rangle\}$. We use $\langle a$, $\dot{a}$ and $a\rangle$ as abbreviations for $(a, \langle)$, $(a, \cdot)$ and $(a, \rangle)$, respectively.[1] For the program in Fig. 1a, we can define the first component of the alphabet to be the set of possible program counter lines $\{p_0, p_1, \ldots, r_4\}$. Then the call to `Q()`, for instance, will be modeled by the letter $\langle q_0$. The implicit representation for the nested

[1] Our notation for internal letters, marking a letter with a dot as in $\dot{a}$, differs slightly from nested words literature which uses simply a. When there is no risk of confusion we may use un-dotted versions too.



(a) A Python program



(b) An excerpt from an HTML document

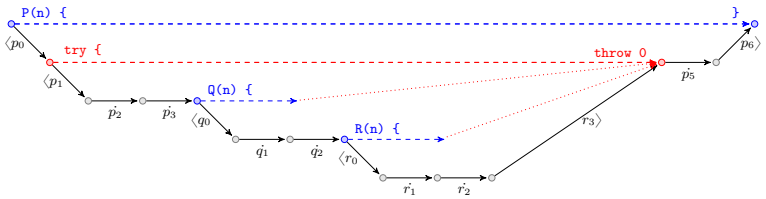
Fig. 2.

word in Fig. 1b is the word obtained by concatenating the letters on the path consisting of all solid edges. The fact that the hierarchical matching between calls and returns is explicitly captured (in comparison to treating them as a linear sequence of instructions) can be exploited for writing more expressive specifications of procedural programs such as pre/post conditions that can be algorithmically verified — see [18] for details.

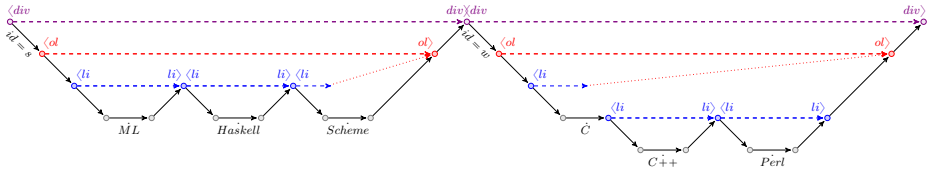
But what happens if an exception is thrown? Then a call (or several calls) will not have a matching return. Viewing the run as a nested word might match the thrown exception with the most recent call, but this is not what we want.

A similar situation happens in parsing programs written in programming languages like *Python* or *Haskell*, that use whitespace to delimit program blocks and deduce variables' scope. In such programming languages, a new block begins by a line starting at a column greater than that of the previous line. If the current line starts at column n (i.e. after n spaces from a new line) and the following line starts at the same column n it is considered on the same block. If the next line starts at a column $n' > n$ it is considered a new block. Last, if the next line starts in a column $n' < n$ then it is considered in the block that started at n' and this implicitly closes all blocks that were opened in between. (If no block started at column n' this would be a syntax error.) If we were to model this with nested words, we can only close the last block, but here we need to close as many blocks as needed.

For strongly matched languages, such as XML, one might want to use this principle to help recover un-closed tags, in cases where this will not result in a confusion, but rather help processing the rest of the document. Consider for instance, the example of Fig. 2b. In this example we have a list with a couple of well-matched list items, and one list item that has no closing tag. We would like to be able to process it and recover from the unmatched list item. If we consider `</ol>` in a way similar to a thrown exception, we can achieve this task.



(a) a colored nested word corresponding to an execution of the code in Fig. 1a where an exception is thrown.



(b) a colored nested word corresponding to the HTML excerpt in Fig. 2b.

Fig. 3.

If we can't model exceptions correctly, we cannot use model checking to formally prove/refute properties about them, and a fundamental property such as "if a certain condition occurs in a program, an exception is thrown and properly caught" is left beyond the scope of verification.

In this work we suggest to augment the nested words model with *colors*. Each call and return, or opening and closing tags, are associated with some color. The hierarchical structure matches only nodes of the same color. This allows relaxing the requirements on the hierarchical edges. A hierarchical edge of a certain color may be unmatched if it is encapsulated by a matched hierarchical edge of a different color. This models catching thrown exceptions, closing as many blocks as needed, or recovering from unmatched tags. Fig. 3a depicts *colored nested words* for the two elaborated examples.

Following a formalization of *colored nested words*, we ask ourselves whether we can use existing machinery of nested word automata and/or nested words tranducers to process colored nested words. Realizing that this is unfeasible, we present *colored nested word automata* (CNA). These automata augment automata for nested words with restricted forms of ε-pop transitions. These ε-transitions enable the automaton to read all the information on the stack that was pushed on un-matched calls. We study also a *blind* version (BCNA), that can see just the information recorded on the matched call. We show that although there could be unboundedly many stack symbols that it cannot observe in comparison to the first automata model, their expressive power is the same.

We show that CNA recognize exactly the class of *regular languages of colored nested words*. We then show that this class of languages is as robust as regular languages: Deterministic CNA are as expressive as their non-deterministic coun-

terparts. It is closed under the following operations: union, intersection, complementation, concatenation, Kleene-*, prefixes, suffixes, reversal, homomorphism and inverse homomorphism. The following problems are decidable: emptiness, membership, language inclusion and language equivalence. We conclude with a grammar characterization. Due to lack of space proofs are omitted; they can be found in the full version on the authors' homepages.

Related Work The key idea in the model of nested words as well as colored nested words is to expose the hierarchical matching between the open and close tags, and the corresponding automata models are really processing the input DAG. To understand the relationship of these automata with classical formalisms such as context-free languages, we can view the input as a sequence of symbols, with the hierarchical structure only implicit, and measure expressiveness by the class of languages of words they define. With this interpretation, the class of regular languages of nested words is a strict superset of regular word languages and a strict subset of DCFLs. This relationship has led to renewed interest in finding classes between regular languages and DCFLs such as realtime height deterministic PDAs [17], synchronized grammars [6], and Floyd grammars/automata [8] (and some interested in languages accepted by higher order pushdown automata, e.g. [14], which are not CFL.) The class of regular languages of colored nested words is a strict superset of regular nested-word-languages and a strict subset of Floyd grammars. While a CNA can be encoded as a Floyd automaton by defining a suitable dependency matrix between input symbols to dictate the stack operations, the view that CNAs are finite-state machines operating over the DAG structure of the input colored nested word leads to a clean theory of regular languages of colored nested words.

2 Colored Nested Words

As is the case in nested words, colored nested words can be represented explicitly using graphs as in Fig. 3a or implicitly using words over an augmented alphabet. We start with the implicit representation. Formally, we define *colored nested words* to be words over alphabets of the form $A \cup A \times C \times \{+,-\}$. Given a triple $\langle a, c, h\rangle$, the first component $a \in A$ provides some content, the second component $c \in C$ provides a *color* and the third component h indicates whether a hierarchical connection starts $(+)$ or ends $(-)$. Letters in A do not influence the hierarchical structure. Letters of the form $\langle a, red, +\rangle$ and $\langle a, red, -\rangle$, can be abbreviated using $(\!|a$ and $a|\!)$, respectively, and similarly for other colors. One can use instead, different parenthesis types for the different colors and form abbreviations such as $\{a$, $[a$, $(\!|a$ and $a\}$, $a]$, $a|\!)$. More generally, we can use $(\!|_c a$, $a_c|\!)$ to abbreviate $\langle a, c, +\rangle$ and $\langle a, c, -\rangle$, respectively. When A and C are clear from the context we use $(\!|\Sigma$, $\dot{\Sigma}$ and $\Sigma|\!)$ for $A \times C \times \{+\}$, A and $A \times C \times \{-\}$, respectively. For a given color $c \in C$ we use $(\!|_c \Sigma$ and $\Sigma_c|\!)$ for the sets $A \times \{c\} \times \{+\}$ and $A \times \{c\} \times \{-\}$, respectively. Finally, we use $\hat{\Sigma}$ for $(\!|\Sigma \cup \dot{\Sigma} \cup \Sigma|\!)$, and $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c}$, and $\mathtt{w}$, $\mathtt{u}$, $\mathtt{v}$ for letters and words in $\hat{\Sigma}$, respectively.

Explicit Representation A colored nested word of length n can be represented by explicitly a tuple $(w, \kappa, \leftrightharpoons, \rightrightarrows, \leftarrow, \rightarrow)$ where w is a word of length n over a finite set of symbols A; κ maps nodes in $[0..n]$ to colors in C; $\leftrightharpoons, \rightrightarrows$ are binary relations in $[0..n-1] \times [1..n]$ and $\rightarrow$ and $\leftarrow$ are unary relations over $[0..n-1]$ and $[1..n]$, resp. The relations $\leftrightharpoons, \rightrightarrows, \leftarrow, \rightarrow$ describe the *hierarchical edges*. The *linear edges* are implicit; there is a linear edge from every $i \in [0..n-1]$ to $i+1$, and w maps the linear edges to the A-symbols, as shown in Fig. 3a.

We refer to $\leftrightharpoons$ and $\rightrightarrows$ as *matched* edges, and *recovered* edges, and to $\leftarrow$ and $\rightarrow$ as *pending calls* and *pending returns*. The following conditions must be satisfied, where for uniformity we view the relations $\leftarrow$ and $\rightarrow$ as binary by interpreting $i \leftarrow$ and $\rightarrow j$ as $i \leftarrow \infty$ and $-\infty \rightarrow j$.

1. Edges point forward: if $i \rightsquigarrow j$ for some $\rightsquigarrow \in \{\leftrightharpoons, \rightrightarrows, \leftarrow, \rightarrow\}$ then $i < j$.
2. Edges do not cross: if $i \rightsquigarrow j$ and $i' \rightsquigarrow' j'$ for some $\rightsquigarrow, \rightsquigarrow' \in \{\leftrightharpoons, \rightrightarrows, \leftarrow, \rightarrow\}$ then it is not the case that $i < i' < j < j'$.
3. Source positions may not be shared: if $i \rightsquigarrow j$ and $i' \rightsquigarrow' j'$ for some $\rightsquigarrow, \rightsquigarrow' \in \{\leftrightharpoons, \rightrightarrows, \leftarrow\}$ then $i \neq i'$.[2]
4. Target positions join at a match: for every $j \in [1..n]$ if the set $\{i \rightsquigarrow j \mid \rightsquigarrow \in \{\leftrightharpoons, \rightrightarrows\}\}$ is non-empty then it contains exactly one $\leftrightharpoons$ edge.

A colored nested word is said to be *well-colored* if (a) matched edges are monochromatic, i.e. if $i \leftrightharpoons j$ then $\kappa(i) = \kappa(j)$, and (b) recovered edges are bi-chromatic, i.e. if $i \rightrightarrows j$ then $\kappa(i) \neq \kappa(j)$ and there exists $i' < i$ such that $i' \leftrightharpoons j$. A well-colored colored nested word is said to be *well-matched* if it has no pending calls, no pending returns, and no recovered edges. It is said to be *weakly-matched* if it has no pending returns and no pending calls (but it may have recovered edges). A weakly-matched colored nested word is said to be *rooted* if the first letter is in $(\!|\Sigma$ and the last letter is in $\Sigma|\!)$. It is said to be c-rooted if it is rooted and the first and last letters are colored c. The *outer level* of a colored nested word is the word obtained by omitting all weakly-matched proper infixes. For instance, if $\mathbb{w} = (\!|a[\![bb]\!]c(\!|d[\![ef]\!)g|\!)$ its outer level is $(\!|acg|\!)$.

For executions of programs with exceptions (Fig. 1b and 3a), a word is well-matched if no exceptions are thrown. For the HTML example, being well-matched means that all open tags are closed in the correct order. If not, as is the case Fig. 2b, the explicit representation of the nested word will contain bi-chromatic edges. In the case of Python programs, being well matched means that after a block ends, there are always some lines of code before the outermost block ends, which is very unlikely.

3 Regularity

Next, we define a notion of *regularity* for colored nested words. We would like to say that a language of colored nested words is *regular* if the language obtained by inserting the missing closing tags is a well-colored regular language of nested

[2] Pending returns ($\rightarrow$) by definition share a source.

words. First we need to define this mapping from a colored nested word to the uncolored nested word obtained by adding the missing closing tags.

Assume our colored alphabet is $\hat{\Sigma} = A \cup A \times C \times \{+,-\}$. We can define the uncolored alphabet $\tilde{\Sigma} = A \cup (A \times C \times \{+,-\}) \cup (\_ \times C \times \{-\})$. A letter $\mathtt{a} \in \hat{\Sigma}$ can be mapped to a letter $\tilde{\mathtt{a}}$ in $\tilde{\Sigma}$ as follows. An opening letter $(a,c,+)$ can be mapped to $\langle(a,c)$, an internal letter a to itself, and a closing letter $(a,c,-)$ can be mapped to $(a,c)\rangle$. We will use letters of the form $(\_,c)\rangle$ to fill the gap of "missing" closing letters. We say that a language over $\tilde{\Sigma}$ is *well-colored* if it can be accepted by a product of two nested word automata (NWA) $\mathcal{A}$ and $\mathcal{A}_c$ where $\mathcal{A}_c$ is a fixed two-state NWA that upon reading $(\!|_c a$ letters pushes the color c to the stack and upon reading $b_d|\!)$ checks that the color on the stack is d, and if it is not goes to its rejecting state.

Adding the missing closing letters is the tricky part. The opening letter corresponding to a missing closing letter is some index i such that in the explicit representation $i \Rrightarrow j$ for some j. For each $j \leqslant |w|$ let I_j be the set of indices i such that $i \Rrightarrow j$. That is, I_j is the set of indices i that are recovered by j. Assume $I_j = \{i_1, i_2, \ldots, i_{\ell_j}\}$ where $i_1 < i_2 < \ldots i_{\ell_j}$. Define $\mathtt{u}_j = \mathtt{c}_{\ell_j} \cdots \mathtt{c}_2 \mathtt{c}_1$ where $\mathtt{c}_k = (\_, \kappa(i_k))\rangle$ for $k \in [1..\ell_j]$. Note that if I_j is empty then $\mathtt{u}_j = \epsilon$. Then adding $\mathtt{u}_j$ just before the j-th letter will close the missing parenthesis recovered by j (if such exist). Formally, for a word $\mathtt{w} \in \hat{\Sigma}$ we define the mapping

$$f(\mathtt{a}_1\mathtt{a}_2\cdots\mathtt{a}_n) = \mathtt{u}_1\tilde{\mathtt{a}}_1\mathtt{u}_2\tilde{\mathtt{a}}_2\ldots\mathtt{u}_n\tilde{\mathtt{a}}_n$$

Definition 1. *A language L of colored nested words is* regular *if the language $f(L) = \{f(\mathtt{w}) \mid \mathtt{w} \in L\}$ is a well-colored regular language of nested words.*

Now that we have a definition of regularity in place, we can ask what machinery can we use to process regular languages of colored nested words. If we can define a transducer machine $\mathcal{M}$ that implements f then we can feed its output $f(\mathtt{w})$ to a nested word automaton and process it instead of $\mathtt{w}$. But such a transducer machine $\mathcal{M}$ won't be a finite state transducer, nor it will be a nested words transducer (NWT) [19,20,12,11,13]. Intuitively, since it needs to map a return letter to several return letters, in fact to an unbounded number of return letters, dependent on the number of unmatched call letters, and while the stack can be used to store this information, an NWT can only inspect the top symbol of the stack.

Therefore we need new machinery to process colored nested words. We can either define a new transducer model that will allow implementing the desired transformation or we can simply define a new automata model that directly process colored nested words. We pursue the second option, which generalizes nested words, and can serve as a base line for a respective transducer model.

4 Colored Nested Word Automata

A *colored nested word automaton* (CNA), is a pushdown automaton that operates in a certain manner, capturing the colored nested structure of the read word.

A CNA over $\hat{\Sigma}$ uses some set of stack symbols P to record information on the hierarchical structure. As in the case of nested word automata, opening letters always cause a push, closing letters always cause a pop, and internal letters do not affect the stack. For CNAs, when a symbol is pushed to the stack, it is automatically colored by the color of the opening letter. Formally on reading $(\!|_c a$ a letter in $P \times \{c\}$ is pushed. When reading a closing letter $b_c|\!)$ the CNA will pop from the stack symbol after symbol until reaching the most recent stack symbol which is c-colored, and make its final move on this letter. We can see this move as composed of several ε-transitions. Note, though, that these are the only possible ε-transitions; the CNA can and must apply an ε-transition only when reading a closing letter of color c, and until a symbol colored c is visible on the stack, but it may not apply an ε-transition at any other time.

We assume a default color $\bot \in C$ for coloring the bottom of the stack. We use Γ to denote stack pairs, i.e. symbols in $P \times C$. For $c \in C$ we use Γ_c and Γ_{-c} for $P \times \{c\}$ an $P \times (C \backslash \{c\})$, respectively. A *configuration* of the automaton is a string γq where q is a state and $\gamma \in \Gamma_\bot \Gamma^*$. The *frontier* of a configuration $s = \gamma q$, denoted $\mathit{frnt}(\gamma q)$ is the pair (q, p) where (p, c) is the top pair of γ for some $c \in C$. We use the term frontiers also for arbitrary pairs in $Q \times P$.

Definition 2 (Colored Nested Word Automaton (CNA)). *A* CNA *over alphabet $A \cup A \times C \times \{+, -\}$ is a tuple $\mathcal{A} = (Q, P, I, F, \delta^(\!|, \dot{\delta}, \delta^)\!|, \delta^\varepsilon)$ where Q is a finite set of states, P is a finite set of stack symbols, $I \subseteq Q \times P$ is a set of initial frontiers, $F \subseteq Q \times P$ is a set of final frontiers. The transition relation is split into four components $\delta^(\!|, \dot{\delta}, \delta^)\!|, \delta^\varepsilon$. Letters in $(\!|\Sigma$ and $\dot{\Sigma}$ are processed by $\delta^(\!|$ and $\dot{\delta}$, respectively. Letters in $\Sigma|\!)$ are processed by both $\delta^)\!|$ and δ^ε. The types of the different δ's are as follows: $\delta^(\!| : Q \times (\!|\Sigma \to 2^{Q \times P}$, $\dot{\delta} : Q \times \dot{\Sigma} \to 2^Q$, $\delta^)\!| : Q \times \Sigma|\!) \times P \to 2^Q$ and $\delta^\varepsilon : Q \times P \to 2^Q$.*

From δ we can infer the evolution of the configuration of the automaton, η as follows.

- Case $\mathbb{a} \in \dot{\Sigma}$: $\quad \eta(\gamma q, \dot{a}) = \{\gamma q' \mid q' \in \dot{\delta}(q, \dot{a})\}$
- Case $\mathbb{a} \in (\!|\Sigma$: $\quad \eta(\gamma q, (\!|_c a) = \{\gamma(p', c)q' \mid (q', p') \in \delta^(\!|(q, (\!|_c a)\}$
- Case $\mathbb{a} \in \Sigma|\!)$:

$$\eta(\gamma q, a_c|\!)) = \left\{ \gamma' q' \;\middle|\; \begin{array}{l} \exists k \geqslant 0, \quad q_0, \ldots, q_k, \quad p_0, \ldots, p_k, \quad c_0, \ldots, c_k \\ \text{s.t. } q_k = q, \quad c_0 = c, \quad c_k, \ldots, c_1 \neq c, \\ \gamma = \gamma'(p_0, c_0)(p_1, c_1) \ldots (p_k, c_k), \\ \forall 0 \leqslant i < k : \; q_i \in \delta^\varepsilon(q_{i+1}, p_{i+1}) \text{ and } q' \in \delta^)\!|(q_0, p_0, a_c|\!)) \end{array} \right\}$$

$$\bigcup \left\{ (p_0, \bot)q_0 \;\middle|\; \begin{array}{l} \exists k \geqslant 0, \quad q_0, \ldots, q_k, \quad p_0, \ldots, p_k, \quad c_1, \ldots, c_k \\ \text{s.t. } q_k = q, \quad c_k, \ldots, c_1 \neq c, \\ \gamma = (p_0, \bot)(p_1, c_1) \ldots (p_k, c_k) \in \Gamma_{-c}^*, \\ \forall 0 \leqslant i < k : \; q_i \in \delta^\varepsilon(q_{i+1}, p_{i+1}) \end{array} \right\}$$

A *run* of the automaton on a $\hat{\Sigma}$-word $\mathbb{w} = \mathbb{a}_1 \ldots \mathbb{a}_n$ is a sequence of configurations $s_0 s_1 \ldots s_n$ such that s_0 is an initial frontier and $s_{i+1} \in \eta(s_i, \mathbb{a}_{i+1})$ for every $0 \leqslant i < n$. A run is *accepting* if $\mathit{frnt}(s_n) \in F$. The automaton accepts a

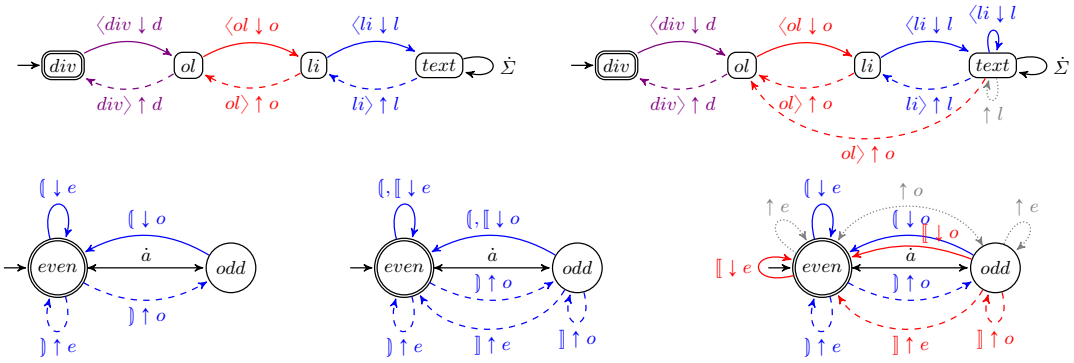


Fig. 4. Some examples of CNAs.

word $\mathbb{w}$ if there exists an accepting run on $\mathbb{w}$. We also use $(q,p) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q',p')$ if $\mathcal{A}$ starting from configuration (q,p) and reading $\mathbb{w}$ may reach a configuration whose frontier is (q',p'). Thus $\mathbb{w}$ is accepted by $\mathcal{A}$ if $(q,p) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q',p')$ for some $(q,p) \in I$ and $(q',p') \in F$. We use $\mathcal{L}(\mathcal{A})$ to denote the set of words accepted by $\mathcal{A}$. An automaton is *deterministic* if I is a singleton and the right hand side of all the δ's are singletons. We use DCNA and NCNA for deterministic and non-deterministic CNAs, respectively.

Fig. 4 provides some examples of CNAs. Push and pop transitions are colored by the respective color, whereas internal transitions are colored black. Push edges have labels of the form $(\!|_c b \downarrow p$ signifying that p is pushed to the stack, pop edges have labels of the form $b_c|\!) \uparrow p$ signifying that the top symbol of the respective color is p. To ease distinction between push and pop transitions, pop edges are dashed. Finally, ε-transitions use grey dotted edges. The initial frontiers in first and second line are all $(div, \bot)$ and $(even, \bot)$, respectively, and the final frontier of each is the same as its initial frontier.

In the first line we have a CNA recognizing a subset of HTML with `div`, `ol` and `li` tags requiring the document to be well matched (left), and a CNA allowing `li` to be unmatched if recovered by `ol` (right).

In the second line the left and middle automata are actually nested word automata — no use of the color is made. The left recognizes all words over $\{a, (\!|, |\!)\}$ where the number of a's within any $(\!|$ and within the outer level, is even. The middle recognized words over $\{[\![,]\!], a, (\!|, |\!)\}$ where in addition the number of a's between any $[\![]\!]$ is odd. When we say here "the number of a's in the word" we mean in the outer level of the word as defined in Sec. 2. The language recognized by the right automaton allows also unmatched $(\!|$ if it is recovered by an encapsulating $[\![]\!]$ in which case the number of a letters in between should be odd. For instance $[\![a(\!|a(\!|aa(\!|a]\!]$ should be accepted whereas $[\![a(\!|a(\!|aa(\!|aa]\!]$ should not.

Theorem 1. *A language of colored nested words is regular iff it is accepted by a* DCNA.

5 Equivalent Models

We show that as is the case in finite automata and nested word automata, nondeterminism does not add expressive power. The proof goes via a generalization of the subset construction. The states of the DCNA are sets of pairs of states. A run of the DCNA on word w will reach state $\{(q_1, q_1'), (q_2, q_2'), \ldots, (q_k, q_k')\}$ iff any run of the NCNA on w reaches one of the states q_i' for $i \in [1..k]$ and for every $i \in [1..k]$ the respective run entered the current hierarchical level at state q_i.

Theorem 2. DCNA*s have the same expressive power as* NCNA*s.*

Blind CNA Next we consider a model where upon reading a return letter $a_{c)}$ the automaton does not have the privilege to read all the stack until the most recent c-symbol. Instead it immediately jumps to the most recent c-colored stack symbol p, popping and ignoring everything above it, and makes a move solely on the base of that p. We call this model *blind* CNA and show that blind CNAs are as expressive as (sighted) CNAs.[3] Dependent on the application the blind or original (sighted) CNA may be more natural. For instance, in the context of software executions, one might prefer the sighted automata to allow modeling of operations such as releasing allocated memory that are taken when an exception is thrown. In the context of parsing Python programs, or recovering from unmatched HTML tags, the blind model may be more natural.

Definition 3 (Blind Colored Nested Word Automaton (BCNA)). *A* BCNA *is a tuple* $\mathcal{B} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)})$ *where all the components are as in the definition of a* CNA*. The evolution of the configuration of the automaton for* $\dot{\delta}$ *and* $\delta^{(}$ *is the same as in* CNA*s. For* $\delta^{)}$ *we have that* $\eta(\gamma q, a_{c)}) = \{\gamma' q' \mid \gamma = \gamma'(p, c)\gamma'' \text{ where } \gamma'' \in \Gamma_{-c}^* \text{ and } q' \in \delta^{)}(q, p, a_{c)})\} \cup \{(p_0, \bot)q \mid \gamma = (p_0, \bot)\gamma' \in \Gamma_{-c}^*\}$*. As in* CNA*s a run of a* BCNA *is a sequence of configurations which adheres to* η *and whose first element is an initial frontier.*

Clearly every BCNA can be simulated by a CNA whose epsilon transitions do not change the state of the automaton. Some CNAs are naturally blind. For instance, the CNA at the top right of Fig. 4 can be made blind by omitting the ε-transition. Simulating the CNA at the bottom right of Fig. 4 by a blind CNA requires adding more states to account for the computations done by the ε-transitions. The proof of the following theorem provides a constructive way to perform such a simulation. The idea is that the states and stack symbols carry an additional component recording a function $\varphi : Q \times C \to Q$ such that $\varphi(q, c)$ tells to which state the CNA will get after popping all non c-stack symbols if the current state is q.

Theorem 3. *Given a deterministic* CNA $\mathcal{A}$ *with* n *states,* k *colors and* m *stack symbols, one can effectively construct a deterministic* BCNA $\mathcal{B}$ *such that* $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$ *with* kn^{n+1} *states and* kmn^n *stack symbols.*

[3] Note that a blind CNA is still different than a traditional nested word automaton, as it has the means to skip all the unmatched calls and arrive to the matching call.

6 Closure Properties and Decision Problems

Theorem 4. *Regular languages of colored nested words are closed under complementation, intersection and union.*

Complementation is done by complementing the set of final frontiers, and intersection and union are done via a product construction.

Theorem 5. *Regular languages of colored nested words are closed under concatenation and Kleene-*.*

Concatenation is proved by guessing a split point and simulating the automata for the operands on each part of the split word. The second automaton treats stack symbols of the first automaton as the bottom of the stack. For Kleene-* the idea is similar, but requires two copies of the simulated automaton to distinguish different sub-words of the split.

Theorem 6. *Regular languages of colored nested words are closed under the operations of prefix, suffix and reversal.*

Closure under reversal is done by dualizing the transitions and switching initial and final frontiers. Closure under prefixes requires guessing an accepting frontier for an extension of the word and tracking it via the transitions. It relies on decidability of emptiness that is provided in Thm. 8. Closure for suffix follows from these two: $\mathit{suff}(L) = (\mathit{pref}(L^R))^R$.

Let $\hat{\Sigma}$ and $\hat{\Sigma}'$ be two colored alphabets. For every $\mathtt{a} \in \hat{\Sigma}$ let $H(\mathtt{a})$ be a language of colored nested words over $\hat{\Sigma}'$. We call H a *substitution*. We say that substitution H is *color-respecting* if the following three conditions hold: (1) for every $\dot{a} \in \dot{\Sigma}$ any word $\mathtt{w}' \in H(\dot{a})$ is weakly-matched, (2) for every $(\!|_c a \in (\!|\Sigma$ any word $\mathtt{w}' \in H((\!|_c a)$ is of the form $(\!|_{c'} b\, \mathtt{v}$ where $\mathtt{v}$ is weakly-matched (3) for every $a_c|\!) \in \Sigma|\!)$ any word $\mathtt{w}' \in H(a_c|\!))$ is of the form $\mathtt{v}\, b_{c'}|\!)$ where $\mathtt{v}$ is weakly-matched. When H maps every $\mathtt{a}$ to a singleton set, we refer to H as an *homomorphism*, and usually denote it with small h. A homomorphism thus maps letters to strings. If h is a homomorphism from $\hat{\Sigma}$ to $\hat{\Sigma}'$, given a language L' over $\hat{\Sigma}'$ we can define its inverse-homomorphic image as $h^{-1}(L') = \{\mathtt{w} \mid \exists \mathtt{w}' \in L'.\ h(\mathtt{w}) = \mathtt{w}'\}$.

Theorem 7. *Regular languages of colored nested words are closed under color-respecting substitution, homomorphism and inverse homomorphism.*

The idea in these proofs is to use an NCNA that guesses a substituted letter $\mathtt{a}$ and then runs in parallel the DCNA for L on the guessed letter $\mathtt{a}$ and the DCNA $\mathcal{M}_{\mathtt{a}}$ for the substitution $H(\mathtt{a})$.

We note that in general, it may be that $h(h^{-1}(L)) \neq L$ and $h^{-1}(h(L)) \neq L$, or moreover that, $h^{-1}(h(L)) \not\subseteq L$. Examples are given in the full version.

The following theorem follows from the result on emptiness of pushdown automata and from the closure under complementation and intersection.

Theorem 8. *Emptiness of* NCNA*s can be solved in polynomial time. Inclusion, universality and equivalence of* NCNA*s are* EXPTIME*-complete.*

The *membership* problem for non-deterministic pushdown automata too is solvable in polynomial time. It thus follows that we can decide on NCNA's membership in polynomial time.

In some contexts, it makes sense to ask about the complexity of membership when the given CNA is fixed. This is the case, for instance, in parsing programs in a given programming language. A CNA for this will stay valid as long as the programming language syntax has not changed. If $\mathcal{A}$ is fixed, we can construct the equivalent deterministic automaton $\mathcal{D}$ using the method in Theorem 2 and then simulate it on the given membership query $\mathtt{w}$. This yields a *streaming* algorithm — an algorithm which reads the input in one pass from left to right (and cannot traverse it again). Thus, this would take $O(|\mathtt{w}|)$ time and $O(d(\mathtt{w}))$ space where $d(\mathtt{w})$ is the hierarchical nesting depth of the colored nested word $\mathtt{w}$.

Theorem 9. *Membership of* NCNA*s can be solved in polynomial time. For a fixed* CNA $\mathcal{A}$ *and colored nested word* $\mathtt{w}$ *of length* ℓ *and depth* d*, the membership problem can be solved in time* $O(\ell)$ *and space* $O(d)$*.*

7 Grammar Characterization

In the following we provide a grammar characterization for regular languages of colored nested words. We first recall some basic definitions. We assume familiarity of basic definition of context-free grammar (and provide it explicitly in the full version.)

Definition 4. *A grammar* $(\mathcal{V}, S, Prod)$ *is said to be a* CNW grammar *w.r.t a set* $C = \{c_1, c_2, \ldots, c_k\}$ *of colors if its variables can be partitioned into sets* $\mathcal{V}^{(}, \mathcal{V}^{)}, \mathcal{V}^{c_1}, \mathcal{V}^{c_2}, \ldots, \mathcal{V}^{c_k}$ *such that the production rules of the grammar are in one of the following forms, where* $X^{(}, Y^{(}, Z^{(} \in \mathcal{V}^{(}$, $X^{)} \in \mathcal{V}^{)}$, $Y, Z \in \mathcal{V}^{)} \cup \mathcal{V}^{(}$, $X^c, Y^c, Z^c \in \mathcal{V}^c$ *and* $X \in \mathcal{V}$*:*

- $X^{(} \longrightarrow \mathtt{a}\, Y^{(}$ *for* $\mathtt{a} \in \dot{\Sigma} \cup (\![\Sigma$
- $X^{(} \longrightarrow (\!|_c a\ Y^c\ b_c|\!)\, Z^{(}$
- $X^{)} \longrightarrow \mathtt{a}\, Y$ *for* $\mathtt{a} \in \Sigma]\!) \cup \dot{\Sigma}$
- $X^{)} \longrightarrow (\!|_c a\ Y^c\ b_c|\!)\, Z$
- $X \longrightarrow \varepsilon$
- $X^c \longrightarrow \dot{a}\, Y^c$
- $X^c \longrightarrow (\!|_{c'} a\ Y^c\ b_{c'}|\!)\, Z^c$
- $X^c \longrightarrow (\!|_{c'} a\ Y^c$ *for* $c' \neq c$

Intuitively, variables in $\mathcal{V}^{(}$ and $\mathcal{V}^{)}$ derive words with no pending returns and no pending calls, respectively, and variables in $\mathcal{V}^{c_i}$ derive weakly matched c_i-rooted words. Note that while the grammar characterization of nested words partitions the grammar variables into two categories, one that disallows pending calls and one that disallows pending returns. For colored nested words, we have additional categories, one per each color. The variables in the category of color c derive weakly matched c-rooted words, thus allowing pending calls of any color other than c.

Theorem 10. *A language* L *is derived by a* CNW*-grammar iff* L *is recognized by a* CNA*.*

References

1. R. Alur and S. Chaudhuri. Temporal reasoning for procedural programs. In *VMCAI*, pages 45–60, 2010.
2. R. Alur, S. Chaudhuri, and P. Madhusudan. A fixpoint calculus for local and global program flows. In *POPL*, pages 153–165, 2006.
3. R. Alur, S. Chaudhuri, and P. Madhusudan. Software model checking using languages of nested trees. *ACM Trans. Program. Lang. Syst.*, 33(5):15, 2011.
4. R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211, 2004.
5. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009, http://robotics.upenn.edu/∼alur/Jacm09.pdf.
6. D. Caucal and S. Hassen. Synchronization of grammars. In *CSR*, pages 110–121, 2008.
7. S. Chaudhuri and R. Alur. Instrumenting C programs with nested word monitors. In *SPIN*, pages 279–283, 2007.
8. S. Crespi-Reghizzi and D. Mandrioli. Operator precedence and the visibly pushdown property. *J. Comput. Syst. Sci.*, 78(6):1837–1867, 2012.
9. D. Debarbieux, O. Gauwin, J. Niehren, T. Sebastian, and M. Zergaoui. Early nested word automata for xpath query answering on XML streams. In *CIAA'13*.
10. E. Driscoll, A. Burton, and T. W. Reps. Checking conformance of a producer and a consumer. In *SIGSOFT/FSE*, pages 113–123, 2011.
11. E. Filiot, O. Gauwin, P-A. Reynier, and F. Servais. Streamability of nested word transductions. In *Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 312–324, 2011.
12. E. Filiot, J-F. Raskin, P-A. Reynier, F. Servais, and J-M. Talbot. Properties of visibly pushdown transducers. In *In Proc. 35th MFCS*, pages 355–367, 2010.
13. E. Filiot and F. Servais. Visibly pushdown transducers with look-ahead. In *SOFSEM 2012: Conf. on Current Trends The. and Prac. of CS*, pages 251–263, 2012.
14. M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.
15. P. Madhusudan and M. Viswanathan. Query automata for nested words. In *MFCS*, pages 561–573, 2009.
16. B. Mozafari, K. Zeng, and C. Zaniolo. High-performance complex event processing over xml streams. In *SIGMOD Conference*, pages 253–264, 2012.
17. D. Nowotka and J. Srba. Height-deterministic pushdown automata. In *MFCS*, pages 125–134, 2007.
18. J. Esparza R. Alur, A. Bouajjani. Model checking of procedural programs. In *Handbook of Model Checking*. Springer, 2015. To Appear.
19. J-F. Raskin and F. Servais. Visibly pushdown transducers. In *Automata, Languages and Programming, ICALP 2008*, pages 386–397, 2008.
20. S. Staworko, G. Laurence, A. Lemay, and J. Niehren. Equivalence of deterministic nested word to word transducers. In *In Proc. 17th FCT*, pages 310–322, 2009.
21. A. Thomo and S. Venkatesh. Rewriting of visibly pushdown languages for XML data integration. *Theor. Comput. Sci.*, 412(39):5285–5297, 2011.

A Proofs of Section 4

A.1 Proof of Theorem 1

Theorem 1 states that a language of colored nested words is regular iff it is accepted by a DCNA. Here is a proof.

Proof. According to Def. 1 a language of colored nested words L is regular, iff $f(L)$ is a well-colored regular language of nested words. Let $\mathcal{D}$ be a DCNA, over $\hat{\Sigma}$. We can define a transducer $\mathcal{T}_{\mathcal{D}}$ that when $\mathcal{D}$ reads a letter in $[\![\Sigma \cup \dot{\Sigma} \cup \Sigma]\!]$ (i.e. when it makes a transition in $\delta^{(} \cup \dot{\delta} \cup \delta^{)}$) $\mathcal{T}_{\mathcal{D}}$ outputs the read letter, and when $\mathcal{D}$ makes a transition in δ^{ε} from $(q,(p,c))$, $\mathcal{T}_{\mathcal{D}}$ outputs $(\_,c)\rangle$. By the definition of DCNAs we get that when a letter in $a_c]\!] \in \Sigma]\!]$ is read $\mathcal{T}_{\mathcal{D}}$ will output all the content of the stack until the next c-colored symbol. If $a_c]\!]$ is the j-th read letter, that would be the word $\mathbb{u}_j$ as defined in Section 3. It follows that on reading $\mathbb{w}$, $\mathcal{T}_{\mathcal{D}}$ will output $f(\mathbb{w})$ and, $f(L)$ is a well-colored regular language of nested words.

Assume $f(L)$ is a well-colored regular language of nested words. Then there exists a nested word automaton (NWA) $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A} \times \mathcal{A}_c) = f(L)$ where $\mathcal{A}_c$ is the fixed NWA that checks that the languages is well-colored. Let $\mathcal{A}' = \mathcal{A} \times \mathcal{A}_c$. Assume $\mathcal{A}' = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)})$.[4] Recall that the alphabet of $\mathcal{A}'$ is $\tilde{\Sigma} = A \cup (A \times C \times \{+,-\}) \cup (\_ \times C \times \{-\})$. We define from $\mathcal{A}'$ a DCNA $\mathcal{C} = (Q, P, I_{\mathcal{C}}, F_{\mathcal{C}}, \delta^{(}_{\mathcal{C}}, \dot{\delta}_{\mathcal{C}}, \delta^{)}_{\mathcal{C}}, \delta^{\varepsilon}_{\mathcal{C}})$ that recognizes L as follows. While the states of $\mathcal{A}'$ are pairs (s_1, s_2) where s_1 is a state of $\mathcal{A}$ and s_2 is a state of $\mathcal{A}_c$, $\mathcal{C}$ does not look at this inner partition. The stack of $\mathcal{A}'$ is also composed of pairs (p, c) where p is the symbol pushed by $\mathcal{A}$ and c is the symbol pushed by $\mathcal{A}_c$, and by the definition of $\mathcal{A}_c$ it is the color of the read open letter. Here $\mathcal{C}$ will makes use of both components of the stack. The initial frontiers $I_{\mathcal{C}}$ are $I \times \{\bot\}$ and the accepting frontiers $F_{\mathcal{C}}$ are $F \times P$. If $q' \in \dot{\delta}(q, \dot{a})$ is a transition of $\mathcal{A}'$ then $q' \in \dot{\delta}_{\mathcal{C}}(q, \dot{a})$ is a transition of $\mathcal{C}$. Likewise, if $(q', p') \in \delta^{(}(q, \langle(a, c))$ is a transition of $\mathcal{A}'$ then $(q', p) \in \delta^{(}_{\mathcal{C}}(q, [\![_c a)$ is a transition of $\mathcal{C}$. Finally, if $q' \in \delta^{)}(q, (a, c)\rangle, (p, c))$ is a transition of $\mathcal{A}'$ then $q' \in \delta^{)}_{\mathcal{C}}(q, a_c]\!], p)$ is a transition of $\mathcal{C}$, and otherwise if $q' \in \delta^{)}(q, (\_, c)\rangle, (p, c'))$ then $q' \in \delta^{\varepsilon}_{\mathcal{C}}(q, p)$ is a transition of $\mathcal{C}$. ▫

B Proofs of Section 5

B.1 CNA with a default initial stack symbol

For the determinization construction it is easier to work with a CNA whose initial condition is a set of states rather than a set of frontiers (that is, a default initial bottom of stack is assumed to be the same for all initial frontiers). We first show that this restriction does not compromise the expressive power.

A CNA $(Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)})$ is said to be an SCNA if there exists $p_0 \in P$ such that $I \subseteq Q \times \{p_0\}$.

Lemma 1. *Every* CNA *can be converted into an equivalent* SCNA.

Proof. Let $\mathcal{A} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon})$ be a CNA. We define a SCNA $\mathcal{B} = (Q_{\mathcal{B}}, P_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}}, \delta^{(}_{\mathcal{B}}, \dot{\delta}_{\mathcal{B}}, \delta^{)}_{\mathcal{B}}, \delta^{\varepsilon}_{\mathcal{B}})$ as follows. The stack symbols $P_{\mathcal{B}}$ are $P \cup \{p_\bot\}$ where $p_\bot$ is a new symbol. The set of states $Q_{\mathcal{B}}$ is $Q \times P_{\mathcal{B}}$. The idea is that $\mathcal{B}$ records

[4] For nested word automata we follow the definition in [4]. Thus I and F are sets of states rather than set of frontiers.

in the state the initial stack symbol. If $\mathcal{B}$ encounters $p_\perp$ on a pop operation, it proceeds as $\mathcal{A}$ would if the current stack symbol was what is recorded in its state. Formally, the initial frontiers $I_\mathcal{B}$ are $\{((q,p),p_\perp) \mid (q,p) \in I\}$. The final frontiers are $F_\mathcal{B} = \{((q,p'),p) \mid (q,p) \in F\} \cup \{((q,p),p_\perp) \mid (q,p) \in F \cap I\}$. For the transition relation we have:

- $(q',p) \in \dot{\delta}_\mathcal{B}((q,p),\dot{a})$ iff $q' \in \dot{\delta}(q,\dot{a})$
- $((q',p),p') \in \delta^{(}_\mathcal{B}((q,p),(\!|_c a)$ iff $(q',p') \in \delta^{(}(q,(\!|_c a)$
- $(q',p) \in \delta^{)}_\mathcal{B}((q,p),(\!|_c a,p')$ iff $p' \neq p_\perp$ and $q' \in \delta^{)}(q,a_{c}|\!)),p')$
- $(q',p) \in \delta^{)}_\mathcal{B}((q,p),(\!|_c a,p_\perp)$ iff $q' \in \delta^{)}(q,a_{c}|\!)),p)$
- $(q',p) \in \delta^{\varepsilon}_\mathcal{B}((q,p),p')$ iff $p' \neq p_\perp$ and $q' \in \delta^{\varepsilon}(q,p')$
- $(q',p) \in \delta^{\varepsilon}_\mathcal{B}((q,p),p_\perp)$ iff $q' \in \delta^{\varepsilon}(q,p)$

□

Lemma 2. *Every* CNA *can be converted into an equivalent* CNA *with a single initial frontier.*

Proof. By Lemma 1 we can convert the given CNA to a CNA $\mathcal{A} = (Q,P,I,F,\delta^{(},\dot{\delta},\delta^{)},\delta^{\varepsilon})$ where $I \subseteq Q \times \{p_0\}$ for some $p_0 \in P$. Let $\mathcal{B} = (Q,P,\{q_I\},F_\mathcal{B},\delta^{(}_\mathcal{B},\dot{\delta}_\mathcal{B},\delta^{)}_\mathcal{B},\delta^{\varepsilon}_\mathcal{B})$ where q_I is a fresh state, $F_\mathcal{B}$ is same as F if $F \cap I = \varnothing$ and otherwise $F_\mathcal{B} = F \cup \{(q_I,p_0)\}$. For the transition relations we connect q_I to the states that are reachable from one of the initial states, thus $\dot{\delta}_\mathcal{B}(q_I,\dot{a}) = \cup_{q\in I}\dot{\delta}(q,\dot{a})$, $\delta^{(}_\mathcal{B}(q_I,(\!|_c a) = \cup_{q\in I}\delta^{(}(q,(\!|_p a)$, $\delta^{\varepsilon}_\mathcal{B}(q_I,p) = \cup_{q\in I}\delta^{\varepsilon}(q,p)$ and $\delta^{)}_\mathcal{B}(q_I,a_{c}|\!)),p) = \cup_{q\in I}\delta^{)}(q,a_{p}|\!)),p)$. □

B.2 Proof of Theorem 2

Theorem 2 states that DCNAs have the same expressive power as NCNAs. Here is a proof.

Proof. The determinization construction is a generalization of the subset construction following that of [5].[5] Let $\mathcal{N} = (Q,P,I,F,\delta^{(},\dot{\delta},\delta^{)},\delta^{\varepsilon})$ be an NCNA. We build an expressively equivalent DCNA $\mathcal{D}$ as follows. The states of $\mathcal{D}$ collect pairs of states of $\mathcal{N}$, so that a pair (q,q') in the collected set captures that on the current level of the hierarchy the NCNA started at state q and reached state q'. Thus, upon reading a letter $\dot{a} \in \dot{\Sigma}$, if the current state is (q,q') and the NCNA can transit from q' to q'' upon reading $\dot{a}$ then (q,q'') will be a member of the collection in the next state of the DCNA. When reading a letter $(\!|_c a$ the DCNA will push the triple (S,a,c) where S is the current set of pairs. As for the state at a call, it will consist of pairs (q'',q'') such that the automaton can get from q' to q'' upon reading $(\!|_c a$. This corresponds to that in the current hierarchical level, the automaton starts at q'' and since so far it processed just ε it is still in q''.

Upon reading a return letter $a_{c}|\!))$ the DCNA can connect the current state with the state the automaton was at the corresponding call as follows. Suppose the

[5] Please see the version in http://robotics.upenn.edu/∼alur/Jacm09.pdf which fixes a bug in the journal version.

current state is S and the uppermost stack triple is (S', a', c'), there exists a state $(q_1, q_2) \in S$ and a state $(q, q') \in S'$ and there is a stack symbol p on which the NCNA can transit from q' to (q_1, p) upon reading $(\!|_c a'$. Then if $c' \neq c$ and there exists an ε-transition on (q_2, p) to q'' then (q, q'') should be a member of the next state of the DCNA. Similarly, if $c' = c$ and when reading the current letter $a_c|\!)$ the NCNA can move from q_2 to q'' then (q, q'') will be a member of the next state of the DCNA.

Formally, the states of $\mathcal{D}$ are $Q' = 2^{Q \times Q}$. The initial state of $\mathcal{D}$ is $\{(q_0, q_0) \mid q_0 \in I\}$. A pair $(S, p) \in Q' \times P$ is accepting if there exists a pair $(q, q') \in S$ such that $(q', p) \in F$. The transition relation is defined as follows:

- For $\mathbb{a} \in \dot{\Sigma}$: $\quad \dot{\delta}_{\mathcal{D}}(S, \dot{a}) = \{(q, q'') \mid (q, q') \in S \text{ and } q'' \in \dot{\delta}(q', \dot{a})\}$.
- For $\mathbb{a} \in (\!|\Sigma$: $\quad \delta^{(}_{\mathcal{D}}(S, (\!|_c a) = (S', (S, a, c))$ where
$$S' = \{(q'', q'') \mid (q, q') \in S \text{ and } q'' \in \delta^{(}(q', (\!|_c a)\}.$$
- For $\mathbb{a} \in \Sigma|\!)$:
$$\delta^{)}_{\mathcal{D}}(S, (S', a', c), a_c|\!)) = \left\{ (q, q'') \;\middle|\; \begin{array}{l} \exists q_1, q_2, q' \in Q,\ p \in P,\ S' \in Q'. \\ (q_1, q_2) \in S,\ (q, q') \in S', \\ (q_1, p) \in \delta^{(}(q', (\!|_c a'),\ q'' \in \delta^{)}(q_2, p, a_c|\!)) \end{array} \right\}$$
$$\delta^{)}_{\mathcal{D}}(S, \bot, a_c|\!)) = \{(q, q'') \mid \exists q' \in Q.\ (q, q') \in S,\ q'' \in \delta^{)}(q', \bot, a_c|\!))\}$$
$$\delta^{\varepsilon}_{\mathcal{D}}(S, (S', a', c')) = \left\{ (q, q'') \;\middle|\; \begin{array}{l} \exists q_1, q_2, q' \in Q,\ p \in P,\ S' \in Q'. \\ (q_1, q_2) \in S,\ (q, q') \in S', \\ (q_1, p) \in \delta^{(}(q', (\!|_{c'} a'),\ q'' \in \delta^{\varepsilon}(q_2, p) \end{array} \right\}$$
$$\delta^{\varepsilon}_{\mathcal{D}}(S, \bot) = \{(q, q'') \mid \exists q' \in Q.\ (q, q') \in S,\ q'' \in \delta^{\varepsilon}(q', \bot)\}$$

□

B.3 Proof of Theorem 3

Given a deterministic CNA $\mathcal{A}$ with n states, k colors and m stack symbols, Theorem 3 states that one can effectively construct a deterministic BCNA $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$ with kn^{n+1} states and kmn^n stack symbols. Here is a proof.

Proof. Let $\mathcal{A} = (Q, P, q_0, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon})$. We simulate $\mathcal{A}$ by a blind CNA that records in the stack, for every possible current state, to which state $\mathcal{A}$ would get to after making all the ε-transitions when reading a closing letter in $\Sigma_c|\!)$. Let Φ be the set of all possible functions from $Q \times C$ to Q. The set of stack symbols $P_{\mathcal{B}}$ is $P \times \Phi$ and the set of states $Q_{\mathcal{B}}$ is $Q \times \Phi$. The accepting states of $\mathcal{B}$ are $F_{\mathcal{B}} = \{((q, \varphi), (p, \varphi')) \mid (q, p) \in F,\ \varphi, \varphi' \in \Phi\}$. The initial state is (q_0, φ_0) where $\varphi_0(q_i, c) = q_i$ for every $q_i \in Q$ and every $c \in C$.

Suppose the current state of $\mathcal{B}$ is (q, φ) and $\mathcal{B}$ reads $a_c|\!)$ and the top symbol at the stack after all non-c symbols have been popped is (p, φ''). Then, since the current state of $\mathcal{B}$ is (q, φ) then after popping all the non c-symbols $\mathcal{A}$ should get to state $\varphi(q, c)$. Let $q' = \varphi(q, c)$ thus $\mathcal{B}$ will move to state (q'', φ'') where $q'' = \delta^{)}(q', a_c|\!))$. Upon reading $(\!|_c a$, if $\delta^{(}(q, (\!|_c a) = (q', p')$ then $\mathcal{B}$ will push $((p', \varphi'), c)$

to the stack where $\varphi'(q_i, c_j) = \varphi(\delta^\varepsilon(q_i, p'), c_j)$ for every $q_i \in Q$ and $c_j \in C$; and move to state (q', φ'). Upon reading a $\dot{a}$ letter, $\mathcal{B}$ will move to state (q', φ) where $q' = \dot{\delta}(q, \dot{a})$.

It is not hard to see that indeed if (q, φ) is the current state, and $\mathcal{A}$ reads a letter in Σ_c], it will reach $\varphi(q, c)$ after processing all the non-c pairs, and thus $\mathcal{B}$ accepts the same language as $\mathcal{A}$. □

C Proofs of Section 6

C.1 Boolean closure

Theorem 4 states that regular languages of colored nested words are closed under complementation, intersection and union. Here is a proof.

Proof. Let $\mathcal{M} = (Q, P, I, F, \delta^(, \dot{\delta}, \delta^), \delta^\varepsilon)$ be a deterministic CNA. Then by complementing the set of final frontiers we can get an automaton for the complement of $\mathcal{L}(\mathcal{M})$. That is, $\mathcal{M}' = (Q, P, I, (Q \times P) \backslash F, \delta^(, \dot{\delta}, \delta^), \delta^\varepsilon)$ accepts $\hat{\Sigma}^* \backslash \mathcal{L}(\mathcal{M})$.

Let $\mathcal{M}_i = (Q_i, P_i, I_i, F_i \delta_i^(, \dot{\delta}_i, \delta_i^), \delta_i^\varepsilon)$ for $i \in \{\text{I}, \text{II}\}$ be two deterministic CNAs. For their intersection we can build the *product* automaton, and choose as accepting states the states whose both components are accepting. Similarly, for union we can build the product automaton, and choose as accepting states the states with at least one accepting component. Formally, $\mathcal{M}_\cap = (Q_\text{I} \times Q_\text{II}, P_\text{I} \times P_\text{II}, I_\times, F_\cap, \delta_\times^(, \dot{\delta}_\times, \delta_\times^), \delta_\times^\varepsilon)$ and $\mathcal{M}_\cup = (Q_\text{I} \times Q_\text{II}, P_\text{I} \times P_\text{II}, I_\times, F_\cup, \delta_\times^(, \dot{\delta}_\times, \delta_\times^), \delta_\times^\varepsilon)$ where

- $I_\times = \{(q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II}) \mid (q_\text{I}, p_\text{I}) \in I_\text{I},\ (q_\text{II}, p_\text{II}), \in I_\text{II}, \}$
- $\delta_\times^(((q_\text{I}, q_\text{II}), \mathtt{a}) = ((q'_\text{I}, q'_\text{II}), (p_\text{I}, p_\text{II}))$ if $\delta_i^(q_i, \mathtt{a}) = (q'_i, p_i)$ for $i \in \{\text{I}, \text{II}\}$
- $\dot{\delta}_\times((q_\text{I}, q_\text{II}), \mathtt{a}) = (q'_\text{I}, q'_\text{II})$ if $\dot{\delta}_i(q_i, \mathtt{a}) = q'_i$ for $i \in \{\text{I}, \text{II}\}$
- $\delta_\times^)((q_\text{I}, q_\text{II}), \mathtt{a}, (p_\text{I}, p_\text{II})) = (q'_\text{I}, q'_\text{II})$ if $\delta_i^)(q_i, \mathtt{a}, p_i) = q'_i$ for $i \in \{\text{I}, \text{II}\}$
- $\delta_\times^\varepsilon((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) = (q'_\text{I}, q'_\text{II})$ if $\delta_i^\varepsilon(q_i, p_i) = q'_i$ for $i \in \{\text{I}, \text{II}\}$
- $F_\cap = \{((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) \mid (q_i, p_i) \in F_i \text{ for } i \in \{\text{I}, \text{II}\}\ \}$
- $F_\cup = \begin{array}{l} \{((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) \mid (q_\text{I}, p_\text{I}) \in F_\text{I},\ (q_\text{II}, p_\text{II}) \in Q_\text{II} \times P_\text{II}, \} \cup \\ \{((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) \mid (q_\text{I}, p_\text{I}) \in Q_\text{I} \times P_\text{I},\ (q_\text{II}, p_\text{II}) \in F_\text{II}, \} \end{array}$

□

C.2 Concatenation closure

Theorem 5 states that regular languages of colored nested words are closed under concatenation and Kleene-*. We provide here its proof.

Proof. Here again we assume there is a unique bottom of stack symbol $\perp \in P$.

Let L_1 and L_2 be two languages of colored nested words, and let $\mathcal{A}_1$ and $\mathcal{A}_2$ be non-deterministic CNA accepting them, respectively. An NCNA $\mathcal{B}$ for their concatenation can work by guessing a split of the given word $\mathtt{w}$ into two words $\mathtt{w}_1$ and $\mathtt{w}_2$ that should be in L_1 and L_2, respectively. The automaton $\mathcal{B}$ starts by simulating $\mathcal{A}_1$ and at any accepting frontier can decide to start simulating

$\mathcal{A}_2$. When simulating $\mathcal{A}_2$, if a read stack symbol is a symbol of $\mathcal{A}_1$ (we assume w.o.l.g. the stack states of $\mathcal{A}_1$ and $\mathcal{A}_2$ are distinct) then it treats it as $\perp$.

For Kleene-* the idea is similar. Given an NCNA $\mathcal{A}$ for a language L the constructed NCNA, $\mathcal{K}$ should guess a split of the given word $\mathtt{w}$ to $\mathtt{w}_1, \ldots, \mathtt{w}_k$ for some k such that for each i, $\mathtt{w}_i$ is in L. As in the case of concatenation, if the automaton while simulating $\mathtt{w}_i$ reads a stack symbol that was pushed when simulating $\mathtt{w}_j$ for any $j < i$, it should treat it as $\perp$. For this purpose $\mathcal{K}$ works with two copies of $\mathcal{A}$: $\mathcal{A}$ and $\mathcal{A}'$, when simulating $\mathcal{A}$ if it reads a stack symbol of $\mathcal{A}'$ (i.e. a tagged stack symbol) it treats it as $\perp$ and vice versa. Again, the choice to alter between $\mathcal{A}$ and $\mathcal{A}'$ can non-deterministically be made when an accepting frontier is reached. □

C.3 Closure under word operations

For a tuple $d = (d_1, d_2, \ldots, d_k)$ we use $d|_i$ for its projection on the i^{th} element — that is, $d|_i = d_i$. For $i \leqslant j \in \mathbb{N}$ we use $[i..j]$ as an abbreviation for $\{i + 1, i + 2 \ldots, j\}$. Given a colored word $\mathtt{w}$ of length n, for every $i \in [1..n]$ we say that $\mathtt{w}[1..i]$ is a *prefix* of $\mathtt{w}$ and $\mathtt{w}[i..n]$ is a *suffix* of $\mathtt{w}$. We use $\mathtt{u} \preceq \mathtt{v}$ and $\mathtt{w} \succeq \mathtt{v}$ if $\mathtt{u}$ is prefix of $\mathtt{v}$ and $\mathtt{w}$ is a suffix of $\mathtt{v}$. For a letter $\mathtt{a} \in \hat{\Sigma}$ we define its *dual* $\overline{\mathtt{a}}$ as follows: $\overline{\dot{a}} = \dot{a}$, $\overline{(\!|_c a} = a_{c}|\!)$, and $\overline{a_{c}|\!)} = (\!|_c a$. If $\mathtt{w} = \mathtt{a}_1\mathtt{a}_2 \ldots \mathtt{a}_n$ then $\overline{\mathtt{a}}_n \ldots \overline{\mathtt{a}}_2\overline{\mathtt{a}}_1$ is the *reverse* of $\mathtt{w}$, and is denoted $\mathtt{w}^R$. Note that $(\mathtt{w}^R)^R = \mathtt{w}$. For a set of colored nested words L we define *pref*(L), *suff*(L) and L^R to be the sets $\{\mathtt{u} \mid \exists \mathtt{v} \in L \text{ s.t. } \mathtt{u} \preceq \mathtt{v}\}$, $\{\mathtt{w} \mid \exists \mathtt{v} \in L \text{ s.t. } \mathtt{w} \succeq \mathtt{v}\}$, $\{\mathtt{v} \mid \exists \mathtt{w} \in L \text{ s.t. } \mathtt{v} = \mathtt{w}^R\}$, respectively.

Theorem 6 states that the class of colored nested words is closed under the operations of prefix, suffix and reversal. Here is a proof.

Proof. We show closure under reversal and prefix. Closure for suffix follows since *suff*$(L) = (\textit{pref}(L^R))^R$.

Let $\mathcal{A} = (Q, P, I, F, \delta^{(\!|}, \dot{\delta}, \delta^{|\!)})$ be a non-deterministic blind CNA. We can build a NBCNA $\mathcal{R}$ for $\mathcal{L}(\mathcal{A})^R$ by switching the roles of initial and accepting frontiers, and dualizing the transition relation. Formally, $\mathcal{R} = (Q, P, F, I, \delta^{(\!|}_{\mathcal{R}}, \dot{\delta}_{\mathcal{R}}, \delta^{|\!)}_{\mathcal{R}})$ where $q \in \dot{\delta}_{\mathcal{R}}(q', \dot{a})$ iff $q' \in \dot{\delta}(q, \dot{a})$; $q \in \delta^{|\!)}_{\mathcal{R}}(q', (\!|_c a, p')$ iff $(q', p') \in \delta^{(\!|}(q, a_{c}|\!))$; and finally $(q, p) \in \delta^{(\!|}_{\mathcal{R}}(q', (\!|_c a)$ iff $q' \in \delta^{|\!)}(q, a_{c}|\!), p)$. It is easy to see that $\mathcal{R}$ accepts a word $\mathtt{w}$ iff its reverse $\mathtt{w}^R$ is accepted by $\mathcal{A}$.

A word $\mathtt{v}$ is in *pref*(L) if it has an extension $\mathtt{w} \succeq \mathtt{v}$ such that $\mathtt{w} \in L$. We first note that given a pair of frontiers (q, p) and (q', p') we can compute whether $\exists \mathtt{w} : \ (q, p) \overset{\mathtt{w}}{\Longrightarrow}_{\mathcal{A}} (q', p')$ by answering the emptiness question for $\mathcal{A}'$ which is obtained from $\mathcal{A}$ by making the initial frontiers $\{(q, p)\}$ and the finals $\{(q', p')\}$. Hence we can compute the set G of frontiers (q, p) such that for some final frontier (q_f, p_f) there exists $\mathtt{w}$ such that $(q, p) \overset{\mathtt{w}}{\Longrightarrow}_{\mathcal{A}} (q_f, p_f)$. A word $\mathtt{v}$ is a prefix of a word in L if when $\mathcal{A}$ reads $\mathtt{v}$ it reaches a frontier $(q', p') \in G$. Thus by taking the CNA $\mathcal{B}$ obtained from $\mathcal{A}$ by setting the set of final frontiers to G we get a CNA for which $\mathcal{L}(\mathcal{B}) = \textit{pref}(\mathcal{L}(A))$.

□

C.4 Closure under substitution and homomorphism

Theorem 3 states that regular languages of colored nested words are closed under color-respecting substitution, homomorphism and inverse homomorphism. Its proof follows from Lemmas 3 and 4 and the fact that homomorphism is a special case of substitution.

Lemma 3. *Regular languages of colored nested words are closed under color-respecting substitution.*

Proof. Let L be a regular language of colored nested words over $\hat{\Sigma}$ and H a color-respecting substitution from $\hat{\Sigma}$ to $\hat{\Sigma}'$ that is describable by a CNA. Given a CNA $\mathcal{M}$ for L, and CNAs $\mathcal{M}_{\mathtt{a}}$ for $H(\mathtt{a})$ for every $\mathtt{a} \in \hat{\Sigma}$ we can build an NCNA for $H(L)$ as follows. The constructed NCNA $\mathcal{M}'$ has three components in its states. The first component records the state of $\mathcal{M}$, the second component records the state of the currently simulated $\mathcal{M}_{\mathtt{a}}$, and the third component records a letter $\mathtt{a}$ that is the current guess of processed letter. The stack symbols of $\mathcal{M}'$ is the set $(\Gamma_{\mathcal{M}} \cup \{\_\}) \times (\cup_{\mathtt{a} \in \hat{\Sigma}} \Gamma_{\mathcal{M}_{\mathtt{a}}})$. The NCNA works as follows. Roughly speaking, $\mathcal{M}'$ starts by guessing a letter $\mathtt{a} \in \hat{\Sigma}$, recording it on the third component, and continues by simulating $\mathcal{M}$ on this letter in the first component of the state and stack, and $\mathcal{M}_{\mathtt{a}}$ in the second component of the state and stack. When $\mathcal{M}_{\mathtt{a}}$ reaches an accepting frontier $\mathcal{M}'$ may switch to guess a new letter $\mathtt{b}$ and continue with simulating $\mathcal{M}$ on $\mathtt{b}$ and $\mathcal{M}_{\mathtt{b}}$ on the next read letters.

When the guessed letter $\mathtt{a}$ is in $\dot{\Sigma} \cup (\!|\Sigma$ the simulation of $\mathcal{M}$ is done with the first processed letter of $H(\mathtt{a})$, when $\mathtt{a} \in \Sigma|\!)$ it is done with the last letter processed for $H(\mathtt{a})$ (i.e. when $\mathcal{M}_{\mathtt{a}}$ reached a frontier and before the next guess is made).

Note that when $\mathtt{a} \in \dot{\Sigma}$, since $H(\mathtt{a})$ is weakly-matched, it will never encounter a foreign stack symbol, and the stack after processing $\mathcal{M}_{\mathtt{a}}$ will be the same as before processing it. When $\mathtt{a} \in (\!|\Sigma$, again no foreign stack symbol will be encountered and after processing $\mathcal{M}_{\mathtt{a}}$ the stack consists of one symbol more than before (whose first component corresponds to $\mathcal{M}$ and second to $\mathcal{M}_{\mathtt{a}}$). When $\mathtt{a} \in \Sigma|\!)$ after processing all letters but the last, $\mathcal{M}_{\mathtt{a}}$ stack's will be same as before processing it, and the last processed letter should incur a pop. As the resulting frontier is accepting $\mathcal{M}'$ will proceed with simulation of $\mathcal{M}$ on the guessed $\mathtt{a}$. If $\mathtt{a}$ recovers some earlier pending calls in the guessed Σ-word then more than one symbol will be popped before the next guess begins. At any case, the last transition of both $\mathcal{M}$ and $\mathcal{M}_{\mathtt{a}}$ took into account the correct stack symbol(s) (where for $\mathcal{M}_{\mathtt{a}}$ it is the bottom of the stack, since that is what it would have encountered on the last letter of $H(\mathtt{a})$). ▫

Since homomorphism is a special case of substitution closure under homomorphism follows.

C.5 Closure under inverse homomorphism

Lemma 4. *Regular languages of colored nested words are under color-respecting inverse homomorphism.*

Proof. Given a CNA $\mathcal{M}$ over $\hat{\Sigma}$ recognizing a language L and a color-respecting homomorphism $h : \hat{\Sigma}' \to \hat{\Sigma}$, we can build a CNA $\mathcal{M}'$ recognizing $h^{-1}(L)$ as follows. Basically, $\mathcal{M}'$ on reading $\mathbb{a}' \in \hat{\Sigma}'$ simulates $\mathcal{M}$ on $h(\mathbb{a}')$. Note that if $\mathbb{a}' \in [\![\Sigma'$ then $\mathcal{M}'$ can (and must) push only one symbol to the stack, but $\mathcal{M}$ can push and pop many symbols to the stack on reading $h(\mathbb{a}')$. However, since h is color-respecting we know that $h(\mathbb{a}')$ is of the form $(\!|_c b\, \mathbb{w}$ where $\mathbb{w}$ is weakly-matched. Thus, at the end of processing $h(\mathbb{a}')$ the stack of $\mathcal{M}$ will consist of just one more symbol. Similarly after reading a letter in $\dot{\Sigma}'$ the stack of $\mathcal{M}$ will be of same size as when stating, and finally after reading a letter in $\Sigma]\!]'$ the stack of $\mathcal{M}$ will consist of one less elements, even if the last letter was a recovering one. We can thus define a mapping $\zeta : [\![\Sigma' \to P$ that returns for each opening letter $\mathbb{a}'$ the stack symbol p that is on the top of the stack when $\mathcal{M}$ finishes processing $h(\mathbb{a}')$. Then $\delta^{\mathfrak{l}}_{\mathcal{M}'}(q, \mathbb{a}') = (\delta^{\mathfrak{l}}(q, h(\mathbb{a}'))|_1,\ \zeta(\mathbb{a}'))$ and for $\mathbb{a}' \in \dot{\Sigma}' \cup \Sigma]\!]'$ we define simply $\dot{\delta}_{\mathcal{M}'}(q, \mathbb{a}') = \tilde{\delta}(q, h(\mathbb{a}'))$, and $\delta^{\mathfrak{d}}_{\mathcal{M}'}(q, \mathbb{a}', p) = \tilde{\delta}(q, h(\mathbb{a}'), p)$ where $\tilde{\delta}$ is the natural extension of δ to words. Finally, $\delta^{\varepsilon}_{\mathcal{M}'}(q, p) = \delta^{\varepsilon}(q, p)$. ▫

C.6 Observations regarding homomorphism and inverse homomorphism

Some observations regarding homomorphism and inverse homomorphism. In general it may be that $h(h^{-1}(L)) \neq L$. For instance, let L be the language accepted by the CNA at the bottom of Fig. 4 in the middle. That is, the language over $\hat{\Sigma} = \{[\![, (\!|, a, |\!),]\!]\}$ where each weakly matched infix encapsulated in $(\!|\, |\!)$ or $[\![\,]\!]$ has even and odd number of a's in its outer level, respectively. Let $\hat{\Sigma}' = \{[\![,]\!], (\!|, |\!), b\}$ and $h : \hat{\Sigma}' \to \hat{\Sigma}$ be defined as follows: $h([\![) = [\![a$, $h(]\!]) = a]\!]$, $h((\!|) = (\!|a$, $h(|\!)) = a|\!)$, $h(b) = aa$. Then $h^{-1}(L)$ has no words with $[\![$ or $]\!]$ since h enforces the number of $a's$ in between to be even. Consequently $h(h^{-1}(L))$ has no words with $[\![$ or $]\!]$ and is thus different from L.

Similarly it can be shown that it may be that $h^{-1}(h(L)) \neq L$, or moreover that, $h^{-1}(h(L)) \not\subseteq L$. For instance, take $\hat{\Sigma} = \hat{\Sigma}' = \{[\![, a,]\!]\}$, let L be the set of rooted weakly-matched words over Σ where the number of a's in every outer level of a rooted weakly-matched word is odd. Let $h([\![) = [\![$, $h(]\!]) =]\!]$ and $h(a) = \varepsilon$. Then $h(L)$ is the set of all rooted weakly-matched words over $\{[\![,]\!]\}$ and $h^{-1}(h(L))$ is the set of all weakly balanced words with no restriction on the number of $a's$.

C.7 Proof of Theorem 8

Theorem 8 states that emptiness of NCNAs can be solved in polynomial time. Inclusion, universality and equivalence of NCNAs are EXPTIME-complete. Here is a proof.

Proof. The *emptiness* problem for (general) pushdown automata is solvable in polynomial time. Since CNAs are a special type of pushdown automata, we can decide on their emptiness using the same procedure. This procedure can be

applied to non-deterministic (or deterministic) CNAs. Having a procedure for emptiness we can decide on inclusion using complementation and intersection, since $L_1 \subseteq L_2$ iff $L_1 \cap L_2^c$ is empty (where L^c denotes the complement of L) and CNAs are closed under all Boolean operation (Thm. 4). Since the construction used in the proof assumed deterministic CNAs, and the determinization construction (Thm. 2) involved an exponential blow up, we can conclude that inclusion can be solved in EXPTIME. Deciding *equivalence* follows from inclusion; and *universality* from emptiness and complementation. The lower bound follows from the lower bound of the respective problems for nested words.

D Proofs and Definitions of Section 7

A *context-free grammar* over an alphabet Σ is a tuple $G = (\mathcal{V}, S, Prod)$, where $\mathcal{V}$ is a finite set of variables, $S \in \mathcal{V}$ is the start variable, and $Prod$ is a finite set of productions of the form $X \longrightarrow \alpha$ where $X \in \mathcal{V}$ and $\alpha \in (\mathcal{V} \cup \Sigma)^*$. The semantics of the grammar G is defined by the derivation relation $\Longrightarrow$ over $(\mathcal{V} \cup \Sigma)^*$: for every production $X \longrightarrow \alpha$ and for all words $\beta, \beta' \in (\mathcal{V} \cup \Sigma)^*$ we have $\beta X \beta' \Longrightarrow \beta\alpha\beta'$. The language $\mathcal{L}(G)$ of the grammar G is the set of all words $w \in \Sigma^*$ such that $S \Longrightarrow^* w$ (i.e. the set of words that can be derived from G by a finite sequence of derivations).

D.1 Proof of Theorem 10

Theorem 10 states that a language L is derived by a CNW-grammar iff L is recognized by a CNA. Here is a proof.

Proof. Let $(\mathcal{V}, S, Prod)$ be a CNW-grammar. We define a non-deterministic blind CNA recognizing its derived language as follows. The set of states is $\mathcal{V}$, the set of stack symbols is $\bot \cup (\Sigma \times \mathcal{V})$. The initial state is S. The final frontiers are those whose state is a nullable variable (i.e. a variable X with production $X \longrightarrow \varepsilon$). The transition relations are defined as follows.

1. If $X \longrightarrow \dot{a}\, Y$ is a production then $Y \in \dot{\delta}(X, \dot{a})$
2. If $X \longrightarrow a_c\rangle\, Y$ is a production then $Y \in \delta^{\rangle}(X, a_c\rangle, \bot)$
3. If $X \longrightarrow \langle_c a\, Y$ is a production then $(Y, \bot) \in \delta^{\langle}(X, \langle_c a)$
4. If $X \longrightarrow \langle_c a\ Y\ b_c\rangle\, Z$ is a production then $(Y, (b_c\rangle, Z)) \in \delta^{\langle}(X, \langle_c a)$
5. If $X \longrightarrow \varepsilon$ is a production then $Z \in \delta^{\rangle}(X, b_c\rangle, (b_c\rangle, Z))$ for every Z.

The first rule corresponds to consuming an internal letter. The second rule corresponds to pending returns and the third to pending calls, which may occur either at the end of the word (when deriving a rule from a variable in $\mathcal{V}^{\langle}$) or within a weakly matched c'-rooted word where the encapsulating color c' is different than the color c of the opening call (i.e. when deriving a variable in $\mathcal{V}^{c'}$). The forth rule correspond to a weakly-matched word, and records on the stack what the closing letter should be and to which state the CNA should transition when it is read. The fifth rule says that the automaton should move to the variable

recorded on the stack, if the recorded closing letter occurred and the variable for the current state was consumed.

Let $\mathcal{B} = (Q, P, \{(q_0, p_0)\}, F, \delta^{(}, \dot{\delta}, \delta^{)})$ be a blind CNA for L. (It is not hard to see that any CNA can be converted into an equivalent one with single initial frontier. A proof is given in the appendix.) We build a CNW-grammar $(\mathcal{V}, S, Prod)$ deriving it as follows. The CNW-grammar variables $\mathcal{V}$ are partitioned to $\mathcal{V}^{(} \cup \mathcal{V}^{)} \cup \mathcal{V}^{c_1} \cup ... \cup \mathcal{V}^{c_k}$ where $\mathcal{V}^{(} = \{X_q \mid q \in Q\}$, $\mathcal{V}^{)} = \{Y_q \mid q \in Q\}$ and $\mathcal{V}^{c_i} = \{Z^{c_i}_{q,q''} \mid q, q'' \in Q\}$ for every $c_i \in C$. Intuitively, X_q variables correspond to $\mathcal{B}$ being in state q and expecting no pending returns, Y_q variables correspond to $\mathcal{B}$ being in state q and expecting no pending calls; and variables $Z^c_{q,q''}$ correspond to $\mathcal{B}$ being in state q and reaching q'' when finishing to process a weakly-balanced c-rooted word.

The start variable is X_{q_0}. The productions are defined as follows.

1. For every $q \in Q$ and $c \in C$ we have $Z^c_{q,q} \longrightarrow \varepsilon$.
2. For every $q \in F$ we have $X_q \longrightarrow \varepsilon$ and $Y_q \longrightarrow \varepsilon$.
3. If $q' \in \dot{\delta}(q, \dot{a})$ then $X_q \longrightarrow \dot{a} X_{q'}$, $Y_q \longrightarrow \dot{a} Y_{q'}$ and $Z^c_{q,q''} \longrightarrow \dot{a} Z^c_{q',q''}$ for every $q'' \in Q$ and $c \in C$.
4. If $(q', p') \in \delta^{(}(q, (\!|_c a)$ and $q''' \in \delta^{)}(q'', b_c|\!), p')$ then $X_q \longrightarrow (\!|_c a \; Z^c_{q',q''} \; b_c|\!) \, X_{q'''}$, $Y_q \longrightarrow (\!|_c a \; Z^c_{q',q''} \; b_c|\!) \, Y_{q'''}$ and $Z^{c'}_{q,r} \longrightarrow (\!|_c a \, Z^{c'}_{q',q''} \, b_c|\!) \, Z^{c'}_{q''',r}$ for every $c' \in C$, $r \in Q$.
5. If $q' \in \delta^{)}(q, a_c|\!), p)$ then $X_q \longrightarrow a_c|\!) \, X_{q'}$.
6. If $(q', p') \in \delta^{(}(q, (\!|_c a)$ then $Y_q \longrightarrow (\!|_c a \, Y_{q'}$ and $Z^{c'}_{q,q''} \longrightarrow (\!|_c a \, Z^{c'}_{q',q''}$ for every $c' \in C$, $q'' \in Q$.

The first rule corresponds to case when a derivation of weakly-matched c-rooted sub-word was complete. The second rule correspond to when derivation of a leading or a trailing or a recovered sub-word is complete. The third rule corresponds to consuming an internal letter. The forth rule summarizes derivation of words with a weakly-matched c-rooted prefix. The fifth rule correspond to consuming a pending closing letter, and the sixth to consuming a pending opening letter. ▫