Colored Nested Words

Rajeev Alur, Dana Fisman

University of Pennsylvania

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.

Note

This is an extended version of the paper that appeared in LATA'16. The version in the proceedings has an error in Theorem 6: Regular languages of nested words as defined in the proceedings version are not closed under reversal. The current version defines regular languages of colored nested words (and colored nested words automata) differently. Under the current definition regular languages of colored nested words are closed under reversal as well. We would like to thank Sarai Sheinvald for pointing to the problem in the proceedings version.

1. Introduction

Nested words, introduced in [Alur and Madhusudan (2004)], 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

```
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 }
```

Figure 1: A procedural program.

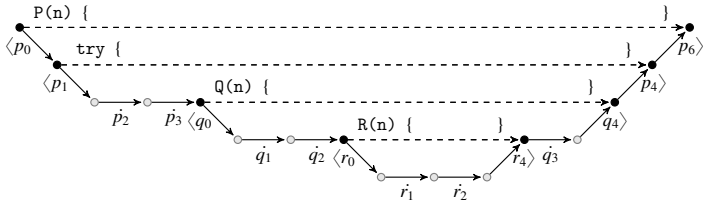


Figure 2: An illustration of an execution of the code of the program in Figure 1 where function calls are captured hierarchically.

non-deterministic counterparts; the class is closed under the following operations: union, intersection, complementation, concatenation, Kleene-*, prefixes, suffixes, reversal, 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. [Mozafari et al. (2012); Debarbieux et al. (2013); Madhusudan and Viswanathan (2009)]) and model checking of procedural programs (see e.g. [Alur and Chaudhuri (2010); Alur et al. (2006); Chaudhuri and Alur (2007); Thomo and Venkatesh (2011); Alur et al. (2011); Driscoll et al. (2011)]).

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. 1. An example execution is illustrated in Fig. 2. 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. 2 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. 1, we can define the first component of the alphabet

[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.

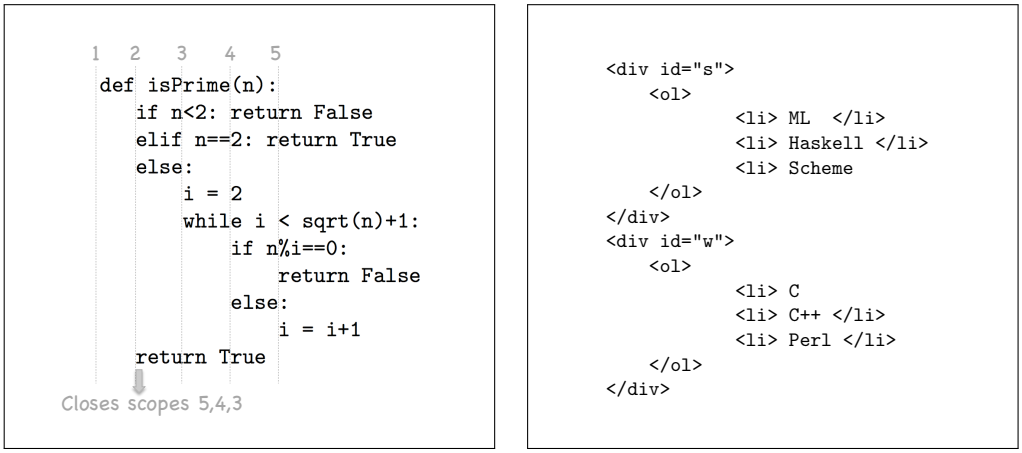


Figure 3: On the left, a Python program, on the right an excerpt from an HTML document

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 word in Fig. 2 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 [Alur et al. (2016)] 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. 3. 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.

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.

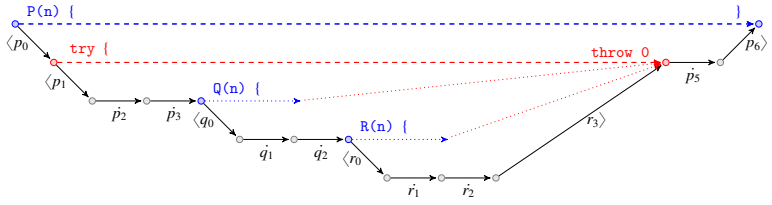


Figure 4: a colored nested word corresponding to an execution of the code in Fig. 1 where an exception is thrown.

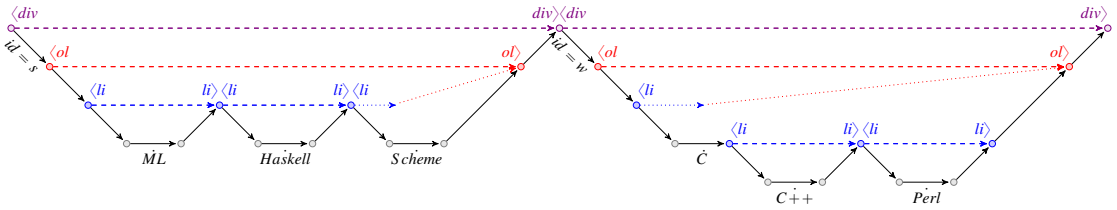


Figure 5: a colored nested word corresponding to the HTML excerpt in Fig. 3.

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* (or *rank*). The colors are assumed to be elements of a finite linearly ordered set. 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 of higher rank. This models catching thrown exceptions, closing as many blocks as needed, or recovering from unmatched tags. Fig. 4 depicts a *colored nested words* corresponding to an execution of the program in Fig. 1 where an exception is thrown. We assume a set of two colors $\{1, 2\}$ and that the exception tags `try` and `throw` and `catch` are of color 2 and function call and returns are of color 1. Fig 5 depicts the nested word corresponding to the excerpt HTML of Fig. 3. Here we assume a set of three colors $\{1, 2, 3\}$ such that $\langle div$ and $div\rangle$ are colored 3, $\langle ol$ and $ol\rangle$ are colored 2, and $\langle li$ and $li\rangle$ are colored 1.

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/push transitions. The ε-pop 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 counterparts. It is closed under the following operations: union, intersection, complementation, concatenation, Kleene-*, prefixes, suffixes, reversal, ho-

momorphism and inverse homomorphism. The following problems are decidable: emptiness, membership, language inclusion and language equivalence. We conclude with a grammar characterization.

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 [Nowotka and Srba (2007)], synchronized grammars [Caucal and Hassen (2008)], and Floyd grammars/automata [Crespi-Reghizzi and Mandrioli (2012)] (and some interest in languages accepted by higher order pushdown automata, e.g. [Hague et al. (2008)], 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. 4 or implicitly using words over an augmented alphabet. We start with the implicit representation. Let A be a finite set of symbols, and C a linearly ordered finite set such as $\{1, 2, \ldots, k\}$. *Colored nested words* are defined 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* or *rank* 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 $\langle a, i, +\rangle$ and $\langle a, i, -\rangle$, can be abbreviated as $(\!|_i a$ and $a_i|\!)$, 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 $i \in C$ we use $(\!|_i \Sigma$ and $\Sigma_i|\!)$ for the sets $A \times \{i\} \times \{+\}$ and $A \times \{i\} \times \{-\}$, respectively. Finally, we use $\hat{\Sigma}$ for $(\!|\Sigma \cup \dot{\Sigma} \cup \Sigma|\!)$, and $\mathbb{a}$, $\mathbb{b}$, $\mathbb{c}$, and $\mathbb{w}$, $\mathbb{u}$, $\mathbb{v}$ for letters and words in $\hat{\Sigma}$, respectively.

It is sometime convenient to use different types of parenthesis, and predefine the order between them. For instance we can decide that $(\!|$ and $|\!)$ have color 2 and that $[\![$ and $]\!]$ have color 1 and then write $(\!|a\, \dot{b}\, [\![c\, \dot{b}\, d]\!]\, e|\!)$ instead of $(\!|_1 a\, \dot{b}\, [\![_1 c\, \dot{b}\, d_1]\!]\, e_1|\!)$. In the sequel we will often use these types of brackets with this assignment of colors.

Explicit Representation. In the above representation the hierarchical structure of the colored nested word is implicit. There is an alternative formulation of colored nested words as graphs meeting certain criteria. A colored nested word of length n can be represented explicitly by a tuple $(w, \kappa, \leftrightharpoons, \leftleftharpoons, \rightrightharpoons, \leftarrow, \rightarrow)$ 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; $\leftrightharpoons$, $\leftleftharpoons$, $\rightrightharpoons$ 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 $\leftrightharpoons$, $\leftleftharpoons$, $\rightrightharpoons$, $\leftarrow$, $\rightarrow$

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 edge with source $i-1$ and target i to the i-th letter of w (an A-symbol), as shown in Fig. 4.

We refer to $\leftrightharpoons$ as *matched* edges, to $\leftharpoondown$ and $\rightharpoondown$ as *recovered calls* and *recovered returns*, resp., and to $\leftarrow$ and $\rightarrow$ as *pending calls* and *pending returns*, resp. 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 $\rightsquigarrow \in \{\leftrightharpoons, \leftharpoondown, \rightharpoondown, \leftarrow, \rightarrow\}$ then $i < j$.
2. Edges do not cross: if $i \rightsquigarrow j$ and $i' \rightsquigarrow' j'$ for some $\rightsquigarrow, \rightsquigarrow' \in \{\leftrightharpoons, \leftharpoondown, \rightharpoondown, \leftarrow, \rightarrow\}$ then it is not the case that $i < i' < j < j'$.
3. Matched edges agree on the color: if $i \leftrightharpoons j$ then $\kappa(i) = \kappa(j)$.
4. Recovered calls and returns are properly colored: if $i \leftharpoondown j$ then $\kappa(i) < \kappa(j)$ and if $i \rightharpoondown j$ then $\kappa(i) > \kappa(j)$.
5. Source positions may not be shared, unless in recovered/pending returns: if $i \rightsquigarrow j$ and $i' \rightsquigarrow' j'$ for some $\rightsquigarrow, \rightsquigarrow' \in \{\leftrightharpoons, \leftharpoondown, \leftarrow\}$ then $i \neq i'$.
6. Target positions may not be shared, unless in recovered/pending calls: if $i \rightsquigarrow j$ and $i' \rightsquigarrow' j'$ for some $\rightsquigarrow, \rightsquigarrow' \in \{\leftrightharpoons, \rightharpoondown, \rightarrow\}$ then $j \neq j'$.

A colored nested word is said to be *well-matched* if it has no pending calls/returns, and no recovered calls/returns. It is said to be *weakly-matched* if it has no pending calls/returns (but it may have recovered calls/returns). 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. 2 and 4), 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. 3, 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.

In Fig. 4 and Fig. 5 we saw examples of colored nested words with matching edges $\leftrightharpoons$, these are the monochromatic horizontal edges in the figure; and of recovered calls $\leftharpoondown$, these are the bi-chromatic edges that start horizontally and break upwards, e.g. the target of linear edge $\dot{p}_3$ to source of the linear edge $\dot{p}_5$ in Fig. 4. But we haven't seen examples of pending returns $\rightarrow$, pending calls $\leftarrow$ and, recovered returns $\rightharpoondown$. Pending calls and returns, may seem less natural, but they play an important role when considering prefixes of words, or suffixes of words, as well as concatenation. For instance we can partition $\mathbb{w} = (\!|a\,\dot{b}\,[\![c\,\dot{b}\,d]\!]\,e|\!)$ into two words $\mathbb{u} = (\!|a\,\dot{b}$ and $\mathbb{v} = [\![c\,\dot{b}\,d]\!]\,e|\!)$ such that $\mathbb{w} = \mathbb{u}\mathbb{v}$. We have that $\mathbb{u}$ has a pending call, and $\mathbb{v}$ has a pending return. To be able to reason about prefixes, suffixes and concatenation, we would like all of these words to be legitimate.

For a similar reason, while recovering returns may not seem natural, we would like to allow them in order to have closure under reversal. For instance the reversal of $\mathbb{w} = a|\!)\,b]\!]\,\dot{c}\,(\!|d\,e|\!)\,\dot{f}\,[\![g\,(\!|h\,i]\!]\,[\![j$ which has a recovered call '$(\!|h$', is $\mathbb{w}^R = j]\!]\,[\![i\,h|\!)\,g]\!]\,\dot{f}\,(\!|e\,d|\!)\,\dot{c}\,[\![b\,(\!|a$ which has a recovered return '$h|\!)$'. The explicit representations of $\mathbb{w}$ and $\mathbb{w}^R$ are given in Fig. 6. By convention matched edges are dashed, and recovered/pending edges are dotted.

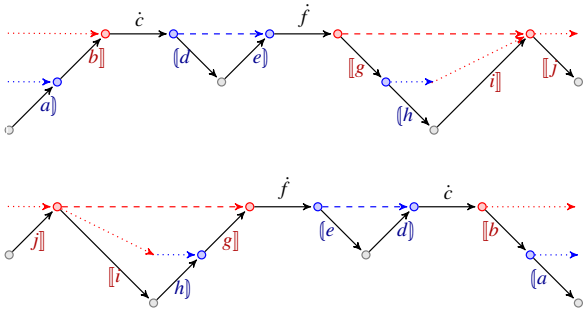


Figure 6: a colored nested word $\mathbb{w} = a|\!) \, b]\!] \, \dot{c} \, (\!|d \, e|\!) \, \dot{f} \, [\![g \, (\!|h \, i]\!] \, [\![j$ and its reverse $\mathbb{w}^R = j]\!] \, [\![i \, h|\!) \, g]\!] \, \dot{f} \, (\!|e \, d|\!) \, \dot{c} \, [\![b \, (\!|a$.

We can define a mapping from the explicit to the implicit representation by assigning closing tags to linear edges whose target is a target of an hierarchical edge, and opening tags to linear edges whose sources are a source of an hierarchical edge, and assigning other edges as internal. This map would be a bjiection. The reverse map is more easily explained using the automata model for colored nested words and is thus differed to Section 4. The fact that the map is a bijection follows from Lemma 2 given in that section, which states that the operations on the stack of a colored nested word automaton are totally determined by the colored nested word.

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

Assume our colored alphabet is $\hat{\Sigma} = A \cup A \times C \times \{+,-\}$. We can define the uncolored alphabet $\tilde{\Sigma} = \langle A_C \cup A_C \cup A_C \rangle$ where $A_C = (A \cup \{_\}) \times C$, $\langle A_C$ stands for $A_C \times \{+\}$ and $A_C \rangle$ stands for $A_C \times \{-\}$. A letter $\mathbb{a} \in \hat{\Sigma}$ can be mapped to a letter $\tilde{\mathbb{a}}$ in $\tilde{\Sigma}$ as follows. An opening letter $(\!|_c a$ can be mapped to $\langle(a,c)$, an internal letter $\dot{a}$ to itself, and a closing letter $a_c|\!)$ can be mapped to $(a,c)\rangle$. We will use letters of the form $\langle(_,c)$ and $(_,c)\rangle$ to fill the gap of "missing" opening and 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.

Given a colored nested word $\mathbb{w}$ we say that $\mathbb{w}'$ is a *completion* of $\mathbb{w}$ if $\mathbb{w}$ is a sub-string of $\mathbb{w}'$ and any letter of $\mathbb{w}'$ that is not in $\mathbb{w}$ is closing/opening an unmatched opening/closing letter of $\mathbb{w}$. There are various ways in which one can complete $\mathbb{w}$ and have the resulting word well-colored. Consider the alphabet $\{\leqslant, \geqslant, [\![,]\!], (\!|, |\!)\}$ where $\leqslant, \geqslant$ are colored 3, $[\![,]\!]$ are colored 2 and $(\!|, |\!)$ are colored 1, and let $\mathbb{w} = [\![\mathbb{u}_1 (\!|\mathbb{u}_2 (\!|\mathbb{u}_3]\!]$ where $\mathbb{u}_1, \mathbb{u}_2, \mathbb{u}_3$ are well colored. The mapping $f()$ that we define will result in $f(\mathbb{w}) = \mathbb{w}' = [\![\mathbb{u}_1 (\!|\mathbb{u}_2 (\!|\mathbb{u}_3 |\!)]\!]$, though $\mathbb{w}'' = [\![\mathbb{u}_1 (\!|\mathbb{u}_2 |\!) (\!|\mathbb{u}_3 |\!)]\!]$, for instance is also a well colored completion of $\mathbb{w}$. Intuitively, the reason $\mathbb{w}'$ is the correct completion is that $]\!]$ here recovers both the first and second $(\!|$. Put otherwise, the explicit representation for

$\mathbb{w}''$ will cause a crossing edge, violating requirement 2, while the explicit representation of $\mathbb{w}'$ is fine. The mapping also needs to respect the order of colors. Consider $\mathbb{v} = \mathbb{u}_1(\!|\mathbb{u}_2 \geqslant \mathbb{u}_3|\!]$. Both $\mathbb{v}' = \leqslant \mathbb{u}_1(\!|\mathbb{u}_2|\!) \geqslant [\![\mathbb{u}_3]\!]$ and $\mathbb{v}'' = [\![\mathbb{u}_1(\!|\leqslant \mathbb{u}_2 \geqslant \mathbb{u}_3|\!)]\!]$ are well-colored completions of $\mathbb{v}$, but the one we want to map to is $\mathbb{v}'$ since in $\mathbb{v}''$ the letter $\geqslant$ is recovered by the pair $[\![]\!]$ which is of smaller color, violating the 4-th requirement of the explicit characterization.

Definition 1. Let $\mathbb{w}$ be a colored nested word over $\hat{\Sigma}$. We let $f(\mathbb{w}) = \mathbb{w}'$ be the word over $\tilde{\Sigma}$ satisfying the following criteria:

1. $\mathbb{w}'$ is a well-colored completion of $\mathbb{w}$.
2. an added closing letter is located as right as possible.
3. an added opening letter is located as left as possible.
4. an added letter of color j cannot be encapsulated within opening and closing letters of color $i \leqslant j$.

where "as right/left as possible" means without violating the other requirements.

Definition 2. A language L of colored nested words is *regular* if the language $f(L) = \{f(\mathbb{w}) \mid \mathbb{w} \in L\}$ is a 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 $f(\mathbb{w})$ to a nested word automaton and process it instead of $\mathbb{w}$. But such a transducer machine $\mathcal{M}$ won't be a finite state transducer, nor it will be a nested words transducer (NWT) [Raskin and Servais (2008); Staworko et al. (2009); Filiot et al. (2010, 2011); Filiot and Servais (2012)]. Intuitively, since it may need to map a return letter to several return letters (or to none at all), 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, the push and pop operations are completely determined by the read word. For CNAs, when a symbol is pushed to the stack, it is automatically colored by the color of the opening letter. Formally on reading $(\!|_j a$ a symbol in $P \times \{j\}$ is pushed. When reading a closing letter $b\, |\!)_j$ the CNA will pop from the stack symbol after symbol as long as their color is $i < j$ until reaching the most recent stack symbol whose color k is bigger or equal to j, and make its final move on this letter. If that symbol is colored j it is popped as well, otherwise it is not. 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 j, and until

a symbol colored $k \geqslant j$ is visible on the stack, but it may not apply an ε-transition at any other time. Note also that on reading $b_{j}\rangle\!\rangle$ in case the top symbol on the stack is colored with a color $k > j$ nothing is popped.

Let C be a linearly ordered set of colors, e.g. $C = \{1, 2, \ldots, k\}$. We let $c_\perp$ denote a color greater than all color in C and let $K = C \cup \{c_\perp\}$. We use Γ to denote stack pairs, i.e. symbols in $P \times K$. For $c \in K$ we use $\Gamma_c, \Gamma_{<c}, \Gamma_{>c}$ and $\Gamma_{\geqslant c}$ for $P \times \{c\}$, $P \times \{c' \mid c' < c\}$, $P \times \{c' \mid c' > c\}$ and $P \times \{c' \mid c' \geqslant c\}$, respectively. A *configuration* of the automaton is a string γq where q is a state and $\gamma \in \Gamma_{c_\perp}\Gamma^*$. The *frontier* of a configuration $s = \gamma q$, denoted $\mathit{frnt}(\gamma q)$ is the triple $(q, (p, c))$ where (p, c) is the top symbol of γ. We use the term frontiers also for arbitrary pairs in $Q \times \Gamma$.

Definition 3 (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, $I \subseteq Q \times \Gamma_{c_\perp}$ is a set of initial frontiers, $F \subseteq Q \times \Gamma$ is a set of final frontiers. The transition relation is split into four components $\dot{\delta}, \delta^{(\!|}, \delta^{|\!)}, \delta^{\varepsilon)}$ where the last three components can be seen as further refined by the color. Letters in $(\!|_c\Sigma$ and $\dot{\Sigma}$ are processed by $\delta^{(\!|_c}$ and $\dot{\delta}$, respectively. Letters in $\Sigma_c|\!)$ are processed by both $\delta^{c|\!)}$ and $\delta^{\varepsilon_c|\!)}$. The types of the different δ's are as follows:

- $\delta^{(\!|_c} : Q \times (\!|_c\Sigma \to 2^{Q \times \Gamma_c}$,
- $\dot{\delta} : Q \times \dot{\Sigma} \to 2^Q$,
- $\delta^{c|\!)} : Q \times \Sigma_c|\!) \times \Gamma_{\geqslant c} \to 2^Q$ and
- $\delta^{\varepsilon_c|\!)} : Q \times \Gamma_{<c} \to 2^Q$.

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

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

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 word $\mathbb{w}$ if there exists an accepting run on $\mathbb{w}$. We also use $(q, p, c) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q', p', c')$ if $\mathcal{A}$ starting from configuration (q, p, c) and reading $\mathbb{w}$ may reach a configuration whose frontier is (q', p', c'). Thus $\mathbb{w}$ is accepted by $\mathcal{A}$ if $(q, p, c) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q', p', c')$ for some $(q, p, c) \in I$ and $(q', p', c') \in F$. 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. 7 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

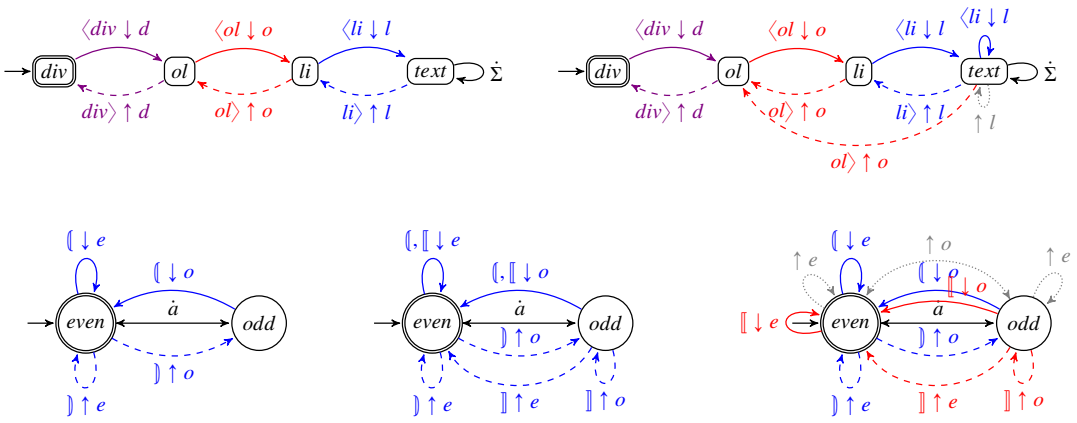


Figure 7: Some examples of CNAs.

the top symbol of the respective color is p. To ease distinction between push and pop transitions, pop edges are dashed. Finally, ε-pop transitions use grey dotted edges. The assignment of colors is as follows: $\langle div, div\rangle$ have color 3, $\langle ol, ol\rangle$, $[\![,]\!]$ have color 2, and $\langle li, li\rangle$, $(,)$ have color 1.

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.

The following theorem states that the class of languages recognized by deterministic CNAs are exactly the set of regular languages of colored nested words as defined in Sec. 3.

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

We will differ the proof of this theorem to Section 5 after we introduce some variations on the definition of CNAs and show that they are equivalent to the definition given here.

The following lemma states that the height of the stack and the colors on the stack are totally determined by the read word. That is, any two runs of a non-deterministic CNA will agree on the height and colors of the stack, as will different runs of different CNAs, on a same given word.

Lemma 2. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two* CNAs *over the same augmented alphabet. Let $s_0 s_1, \ldots, s_n$ and $s'_0 s'_1, \ldots, s'_n$ be runs of $\mathcal{C}$ and $\mathcal{C}'$ on a given word w. For $0 \leqslant i \leqslant n$, let $s_i = \gamma_i q_i$ and $s'_i = \gamma'_i, q'_i$.*

Then for every $0 \leqslant i \leqslant n$ *we have* $|\gamma_i| = |\gamma'_i|$ *and for every* $0 \leqslant j \leqslant |\gamma_i|$ *we have* $\gamma_i[j]$ *and* $\gamma'_i[j]$ *are of the same color.*[2]

It follows that a CNA can be seen as traversing the DAG of an explicit representation of colored nested word, and annotating the target node of each linear edge by the automaton state it reaches after reading the corresponding word, and annotating the source of each hierarchical edge by the symbol pushed to the stack when the corresponding letter is read. When traversing the target of a hierarchical edge the CNA can observe the latter annotation, which is equivalent to discovering the symbol on the top of the stack after all symbols that were popped due to the current read closing letter are removed from the stack.

Given a colored nested word in implicit representation we can deduce the explicit representation by following the stack operations the CNA goes through when reading it, which by Lemma 2 are well defined per each word. The linear labeling and coloring follow exactly that of the implicit word. For the hierarchical edges, a matching edge ($\leftrightharpoons$) is added on every closing tag, from the opening tag of the same color for which all symbols of the stack above it are of smaller color, if such exists; otherwise, a pending return ($\rightharpoonup$) is added to that closing tag; a recovered call edge ($\leftharpoondown$) is added to all symbols of the stack with color smaller than that of the closing tag, such that all symbols above it are also of smaller color; a recovered return ($\rightleftharpoons$) is added on closing tags if the top symbol of the stack, is of higher color; and last, pending calls ($\leftharpoonup$) are added to all symbols that are left on the stack when the run ends.

5. Equivalent Models

One of the great properties of regular languages is that their deterministic and non-deterministic models have the same expressive power. We will see in Section 5.2 that this is true for regular languages of colored nested words as well.

Another interesting result shows that a model where the CNA cannot read the symbols popped from the stack on its ε-pop transitions has the same expressive power, despite the fact that the content of the stack that it cannot inspect may be unboundedly large. We call this model a *blind* CNA and prove this in Section 5.3.

We start (in Section 5.1) by considering another variation of the model, one that also has ε-push transitions. This model is in a sense more symmetrical, and it is convenient to use in various proof, in particular the regularity theorem (Thm. 1).

5.1. A model with ε-push transitions

Colored nested word automata as presented in Def. 3 have ε-pop transitions but not ε-push transitions. The ε-pop transitions are introduced to allows us to pop symbols from the stack in the event they are recovered by a closing letter of bigger color. Their popping mimics reading the missing closing letters. On the event of reading a closing letter that is recovered by an opening letter of bigger color a CNA simply changes the state without popping anything from the stack as it normally does when reading a closing letter. We can think of a model that first allows the automaton to push a letter to stack to mimic the missing opening letter and then, the $\delta^{)}$ operation will pop that symbol on consuming the read letter as it normally does.

[2] Where $\gamma[j]$ refers to the j'th letter of γ.

Definition 4 (ECNA). An ECNA is a tuple $\mathcal{A} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)}, \delta^{(\varepsilon})$ where the components $\delta^{(}, \delta^{)}, \delta^{\varepsilon)}, \delta^{(\varepsilon}$ are refined by color. Transitions $\delta^{(\varepsilon_c} : Q \to 2^{Q \times \Gamma_c}$ are ε-push transitions that are executed when the ECNA reads a closing letter $a_{c]}$ and the symbol at the top of the stack after removing all symbols of color $c' < c$ has color $c'' > c$. All other components but $\delta^{)}$ are as in a CNA. A $\delta^{c)}$ function maps $Q \times \Sigma_{c]} \times \Gamma_c$ to 2^Q. That is, the difference between $\delta^{c)}$ in CNA and ECNA is that the former considers elements in $\Gamma_{\geqslant c}$ and the latter is restricted to Γ_c. The evolution of the stack is defined the same for letters in $\dot{\Sigma} \cup (\!|\Sigma$ and as follows for letters in $\Sigma|\!)$:

$$\eta(\gamma q, a_{c]}) = \left\{ \gamma' g q'' \;\middle|\; \begin{array}{l} \exists k \geqslant 0,\ \ q_0, \ldots, q_k,\ \ p_0, \ldots, p_k,\ \ c_0, \ldots, c_k \\ \text{s.t. } q_k = q,\ \ c_0 \geqslant c,\ \ c_k, \ldots, c_1 < c, \\ \gamma = \gamma'(p_0, c_0)(p_1, c_1) \ldots (p_k, c_k), \\ \forall 0 \leqslant i < k :\ q_i \in \delta^{\varepsilon_c)}(q_{i+1}, (p_{i+1}, c_{i+1})) \text{ and} \\ \text{if } c_0 = c \text{ then } g = \epsilon \text{ and } q'' \in \delta^{c)}(q_0, a_{c]}), (p_0, c_0)) \text{ and} \\ \text{if } c_0 > c \text{ then } g = (p_0, c_0) \text{ and } \exists q', p' \text{ s.t.} \\ (q', (p', c)) \in \delta^{(\varepsilon_c}(q_0) \text{ and } q'' \in \delta^{c)}(q', a_{c]}), (p', c)) \end{array} \right\}$$

Note that here as well, while ε-push transitions are available, they do not add non-determinism: it is prescribed exactly when they can and should be taken.

The following theorem states that ECNA and CNA recognize the same set of languages.

Theorem 3. *An* ECNA *can be converted into an equivalent* CNA*, and vice versa.*

PROOF. The proof of the first direction follows from the fact that a $\delta^{(\varepsilon_c}$-transition is always immediately followed by a $\delta^{c)}$ transition and thus the two can be summarized by one $\delta^{c)}$ transition in the case of a CNA. Formally, let $\mathcal{E} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)}, \delta^{(\varepsilon})$ be an ECNA. We construct from it the following CNA $\mathcal{C} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}_{\mathcal{C}}, \delta^{\varepsilon)})$ where $\delta^{c)}_{\mathcal{C}} = \delta^{c)} \cup \delta'^{c)}$ and $q'' \in \delta'^{c)}(q, a_{c]}), (p', c))$ if $\exists q', p'.\ (q', (p', c)) \in \delta^{(\varepsilon_c}(q)$ and $q'' \in \delta^{c)}(q', a_{c]}), (p', c,))$. Clearly $\mathcal{E}$ and $\mathcal{A}$ accept the same set of words.

For the second direction, assume we have a CNA $\mathcal{C} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)})$. We build from it an ECNA $\mathcal{E} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta'^{)}, \delta'^{\varepsilon)}, \delta'^{(\varepsilon})$ where $\delta'^{c)}$ is obtained from $\delta^{c)}$ by omitting the transitions in which the stack symbol $\gamma \in \Gamma_{>c}$; instead if $q' \in \delta^{c)}(q, a_{c]}), (p, c'))$ and $c' > c$ we add the transition $(q, (p, c)) \in \delta'^{(\varepsilon_c}(q)$ and the transition $q' \in \delta'^{\varepsilon_c)}(q, (p, c))$. It is easy to see that $\mathcal{C}$ accepts the same words as $\mathcal{E}$.

We can now prove Theorem 1 stating that the CNA machine model recognizes exactly regular language of colored nested words.

PROOF OF THEOREM 1. According to Def. 2 a language of colored nested words L is regular, iff $f(L)$ is a regular language of nested words. Let $\mathcal{C}$ be an ECNA, over $\hat{\Sigma}$. By tracking a run of the ECNA on a given word w, and adding a $\langle(_, j)$ letter transition whenever $\mathcal{C}$ makes a $\delta^{(\varepsilon_j}$ transition, and adding a $(_, j)\rangle$ letter whenever $\mathcal{C}$ makes a $\delta^{\varepsilon_j)}$ transition, we can obtain $f(w)$ as provided in Def. 1. Indeed, a sequence of letters $(_, j_1)\rangle, (_, j_2)\rangle, \ldots, (_, j_k)\rangle$ will be added to match the respective opening letters just before encountering a closing letter of color k bigger than j_i for $i \in [1..j]$, which is the rightmost positions these closing letters can be without violating the requirements. Likewise, a letter $\langle(_, j)$ will be added in the event an unmatched j-colored closing letter is read, in which case it is added to the left of all the opening symbols popped from the stack, and just to the right of an opening letter of bigger color (or the beginning of the word), which is the leftmost position that does not violate the requirements. Note that in the event of

consecutive unmatched closing letters $a_{j_a}\rangle, b_{j_b}\rangle, \ldots k_{j_k}\rangle$ the ECNA $\mathcal{C}$ first makes an $\delta^{(\varepsilon}_{j_a}$ transition and immediately pops the added symbol, then makes a $\delta^{(\varepsilon}_{j_b}$ transition and so on, so the added letters appear in the reverse order $\langle(_, j_k), \ldots \langle(_, j_b), \langle(_, j_a)$.

Let $\mathcal{C} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)}, \delta^{(\varepsilon})$ be an ECNA. We can define from $\mathcal{C}$ a nested word automaton $\mathcal{N} = (Q, P \times C, I, F, \delta^{(}_{\mathcal{N}}, \dot{\delta}, \delta^{)}_{\mathcal{N}})$ over $\tilde{\Sigma}$ as follows. If $(q', p') \in \delta^{(}(q, (_j a)$ then $(q', p') \in \delta^{(}_{\mathcal{N}}(q, \langle(a, j))$. In addition, if $(q', p') \in \delta^{(\varepsilon}_{j}(q)$ then $(q', p') \in \delta^{(}_{\mathcal{N}}(q, \langle(_, j))$. Similarly, if $q' \in \delta^{)}(q, a_{j}\rangle), (p, j')))$ then $q' \in \delta^{)}_{\mathcal{N}}(q, (a, j)\rangle, (p, j'))$. In addition, if $q' \in \delta^{\varepsilon)}_{j}(q, (p, j'))$ then $q' \in \delta^{)}_{\mathcal{N}}(q, (_, j)\rangle, (p, j'))$. Clearly, $\mathcal{A}$ will recognize $f(L)$ and thus $f(L)$ is a regular language of nested words.

For the other direction, assume $f(L)$ is a regular language of nested words. Then there exists a nested word automaton (NWA) $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = f(L)$. Since $\mathcal{A}$ recognizes $f(L)$ we can assume that on reading a word, if a violation of the requirements of Def. 1 is observed, then $\mathcal{A}$ goes to a sink rejecting state. In particular, since $f(L)$ is well-colored, we can assume $\mathcal{A} = \mathcal{A}' \times \mathcal{A}_c$ where $\mathcal{A}_c$ is the fixed NWA defined in Sec. 3 making sure words are well colored. Equivalently, we can assume the set of stack symbols Γ of A is of the form $P \times C$, and it uses the color component as $\mathcal{A}_c$ does. Assume $\mathcal{A} = (Q, \Gamma, I, F, \delta^{(}, \dot{\delta}, \delta^{)})$. Recall that the alphabet of $\mathcal{A}$ is $\tilde{\Sigma}$ as defined in Sec. 3. We use $\langle\Sigma_$ and $\Sigma_\rangle$ for $\{\langle(_, c) \mid c \in C\}$ and $\{(_, c)\rangle \mid c \in C\}$ resp. We say that a word w over $\tilde{\Sigma}$ has no pending calls (resp. no pending returns) if the word w' obtained after omitting all the $\langle\Sigma_$ letters (resp. $\Sigma_\rangle$ letters) is a well matched nested word.

We would like to build from $\mathcal{A}$ a CNA $\mathcal{C}$ accepting L. The idea is that as long as the word has no $\langle\Sigma_$ letters, $\mathcal{C}$ can mimic $\mathcal{A}$'s transitions one by one: when $\mathcal{A}$ pushes/pops a letter so does $\mathcal{C}$ where if the popped letter is in $\Sigma_\rangle$ then $\mathcal{C}$ uses a ε-pop transition for that. The challenge is coping with $\langle\Sigma_$ letters, since while $\mathcal{A}$ pushes them on the read, a CNA (or ECNA) learns about them being missing only when encountering the recovered closing letter. To cope with this $\mathcal{C}$ uses its non-determinism to guess the $\langle\Sigma_$ letters of the word. The guesses are being made at the initial states and on every push of a regular letter (in $\langle(\Sigma \times C)$).

Formally, we define the CNA $\mathcal{C} = (Q, P, I_C, F, \delta^{(}_C, \dot{\delta}_C, \delta^{)}_C, \delta^{\varepsilon)}_C)$ where $I_C = \{(q, p, c) \mid \exists q_0 \in I, \gamma \in \Gamma^*, w$ with no pending returns s.t. $\bot q_0 \overset{w}{\Longrightarrow}_{\mathcal{A}} \bot\gamma(p, c)q\}$ and the transitions are defined as follows:

1. $\dot{\delta}_C(q, \dot{a}) = \{q' \mid q' \in \dot{\delta}(q, \dot{a})\}$
2. $\delta^{c)}_C(q, a_c\rangle), (p, c)) = \{q' \mid q' \in \delta^{)}(q, (a, c)\rangle, (p, c))\}$
3. $\delta^{\varepsilon)}_C(q, (p, c')) = \{q' \mid q' \in \delta^{)}(q, (_, c')\rangle, (p, c')),\ c' < c\}$
4. $\delta^{(c}_C(q, (_c a) = \{(q', (p, c)) \mid (q', (p, c)) \in \delta^{(}(q, \langle(a, c))\}$
5. $\delta^{(c}_C(q, (_c a) =$
   $\{(q', (p', c)) \mid \exists\gamma \in \Gamma^*, w$ with no pending returns s.t. $\bot q \overset{\langle(a,c)w}{\Longrightarrow}_{\mathcal{A}} \bot\gamma(p', c')q'\}$

We can show by induction that there exists a run $s_0 s_1 s_2 \ldots s_n$ of $\mathcal{C}$ on $\mathbb{w}$ if and only if there exists a run $t_0 t_1 t_2 \ldots t_m$ of $\mathcal{A}$ on $f(\mathbb{w})$ and a set of indexes $i_0 < i_1 < i_2 < \ldots < i_n$ such that for every $j \in [0..n]$ we have $s_j = t_{i_j}$ and $i_m = n$. It follows that $\mathcal{C}$ recognizes L as required. □

5.2. Deterministic CNA*s are as expressive as non-deterministic* CNA

Before providing the deterministic model, we explore a couple of additional variations that come in handy in other proofs. It is often 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) or a CNA whose initial set of states is a singleton. The following two lemmas state that these restrictions do not compromise the expressive power. The proofs of these lemmas use standard techniques and are given in the appendix for completeness.

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 $I \subseteq Q \times \{p_0\} \times \{c_\perp\}$. In this case we often regard to I as a subset of Q.

Lemma 4. *Every* CNA *can be converted into an equivalent* SCNA.

Lemma 5. *Every* CNA *can be converted into an equivalent* CNA *with a single initial frontier.*

The main result of this section is that as is the case in finite automata and nested word automata, non-determinism does not add expressive power.

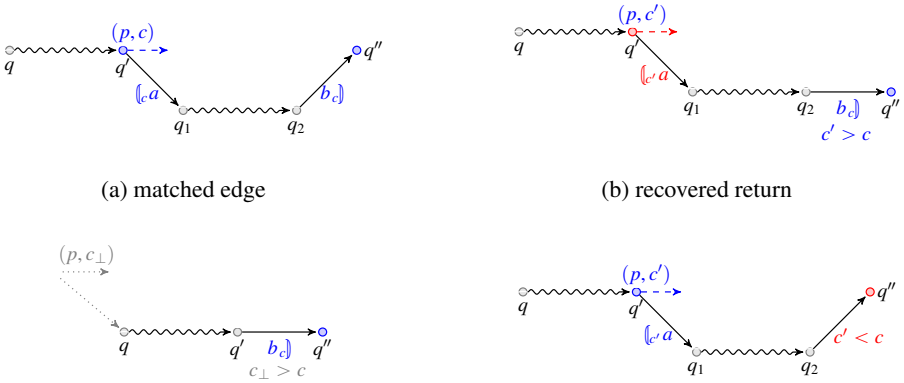
Theorem 6. DCNA*s have the same expressive power as* NCNA*s.*

PROOF. The determinization construction is a generalization of the subset construction following that of [Alur and Madhusudan (2009)].[3] Consider an NCNA $\mathcal{N} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)})$. By Lemma 4 we can assume $\mathcal{N}$ is an SCNA. 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 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 $c' < c$ 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 $c' \geqslant c$ 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 frontier $(S, g) \in Q' \times \Gamma$ is accepting if there exists a pair $(q, q') \in S$ such that $(q', g) \in F$. The transition relation is defined as follows:

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



(a) matched edge

(b) recovered return

(c) pending return

(d) recovered call

Figure 8: Illustration of the different cases considered in construction of transitions in the proof of Thm. 6

$$\delta_{\mathcal{D}}^{c\rrbracket}(S,(S',a',c),b_c\rrbracket) = \left\{ (q,q'') \;\middle|\; \begin{array}{l} \exists q_1,q_2,q' \in Q,\ p \in P. \\ (q_1,q_2) \in S,\ (q,q') \in S'. \\ (q_1,(p,c)) \in \delta^{\llbracket c}(q',\llbracket_c a), \\ q'' \in \delta^{c\rrbracket}(q_2,b_c\rrbracket),(p,c)) \end{array} \right\} \qquad \text{(see Fig. 8 (a))}$$

$$\bigcup \left\{ (q_1,q'') \;\middle|\; \begin{array}{l} \exists q_1,q_2,\in Q,\ p \in P,\ c' > c. \\ (q_1,q_2) \in S,\ (q,q') \in S', \\ (q_1,(p,c')) \in \delta^{\llbracket_{c'}}(q',\llbracket_{c'} a), \\ q'' \in \delta^{c\rrbracket}(q_2,b_c\rrbracket),(p,c)) \end{array} \right\} \qquad \text{(see Fig. 8 (b))}$$

$$\delta_{\mathcal{D}}^{c_\perp\rrbracket}(S,b_c\rrbracket,(p,c_\perp)) = \left\{ (q,q'') \;\middle|\; \begin{array}{l} \exists q' \in Q.\ (q,q') \in S, \\ q'' \in \delta^{c_\perp\rrbracket}(q',b_c\rrbracket)(p,c_\perp)) \end{array} \right\} \qquad \text{(see Fig. 8 (c))}$$

$$\delta_{\mathcal{D}}^{\varepsilon\rrbracket}(S,(S',a',c')) = \left\{ (q,q'') \;\middle|\; \begin{array}{l} \exists q_1,q_2,q' \in Q,\ g \in \Gamma,\ c' < c\,. \\ (q_1,q_2) \in S,\ (q,q') \in S', \\ (q_1,(p,c')) \in \delta^{\llbracket_{c'}}(q',\llbracket_{c'} a'), \\ q'' \in \delta^{\varepsilon\rrbracket}(q_2,(p,c')) \end{array} \right\} \qquad \text{(see Fig. 8 (d))}$$

□

5.3. *Blind CNAs*

We now consider a model where upon reading a return letter $a_c\rrbracket$ the automaton does not have the privilege to read all the stack until the most recent symbol of color greater or equal to c. Instead it immediately jumps to the most recent stack symbol p colored c or greater, 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.[4] Dependent on the

[4]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 of lower color and arrive to the matching call, if such exists, and a greater call otherwise.

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 5 (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 CNAs. For $\delta^{)}$ we have that

$$\begin{aligned}&\eta(\gamma q, a_c\rangle) = \\ &\quad \{\gamma' q' \mid \gamma = \gamma'(p,c)\gamma'',\ \gamma'' \in \Gamma^*_{<c} \text{ and } q' \in \delta^{)}(q, a_c\rangle, (p,c))\} \cup \\ &\quad \{\gamma'(p',c')q' \mid \gamma = \gamma'(p',c')\gamma'',\ \gamma'' \in \Gamma^*_{<c},\ c' > c \text{ and } q' \in \delta^{)}(q, a_c\rangle, (p,c'))\}\end{aligned}$$

As in CNAs 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. 7 can be made blind by omitting the ε-transition. Simulating the CNA at the bottom right of Fig. 7 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 $\varphi(q,c)$ dictates to which state the CNA will get after popping all c'-stack symbols for $c' < c$ if the current state is q.

Theorem 7. *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* n^{kn+1} *states and* mn^{kn} *stack symbols.*

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 $\Sigma_c\rangle$. 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 $a_c\rangle$ and the top symbol at the stack after symbols with color smaller than c have been popped is (p,φ''). Then, since the current state of $\mathcal{B}$ is (q,φ) then after popping all these 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 $q'' = \delta^{)}(q', a_c\rangle)$. Upon reading $\langle_c a$, if $\delta^{(}(q, \langle_c a) = (q', p')$ 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 $\Sigma_c\rangle$, it will reach $\varphi(q,c)$ after processing all the c'-pairs for $c' < c$, and thus $\mathcal{B}$ accepts the same language as $\mathcal{A}$. □

6. Boolean Closure and Decision Problems

Since regular languages of colored nested words can be recognized by deterministic CNAs, as per Theorem 6, it is easy to see that they are closed under complementation. Closure under union and intersection is also straight forward, since as guranteed by Lemma 2, two CNAs running on the same word, preform exactly the same stack operations. Thus, running them in parallel can be achieved by a simple product construction.

Theorem 8. *Regular languages of colored nested words are closed under complementation, intersection and union.*

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

- $I_\times = \{(q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II}) \mid (q_\text{I}, p_\text{I}) \in I_\text{I},\ (q_\text{II}, p_\text{II}), \in I_\text{II}, \}$
- $\delta_\times^{(}((q_\text{I}, q_\text{II}), \mathbb{0}) = ((q'_\text{I}, q'_\text{II}), (p_\text{I}, p_\text{II}))$ if $\delta_i^{(}(q_i, \mathbb{0}) = (q'_i, p_i)$ for $i \in \{\text{I}, \text{II}\}$
- $\dot{\delta}_\times((q_\text{I}, q_\text{II}), \mathbb{0}) = (q'_\text{I}, q'_\text{II})$ if $\dot{\delta}_i(q_i, \mathbb{0}) = q'_i$ for $i \in \{\text{I}, \text{II}\}$
- $\delta_\times^{)}((q_\text{I}, q_\text{II}), \mathbb{0}, (p_\text{I}, p_\text{II})) = (q'_\text{I}, q'_\text{II})$ if $\delta_i^{)}(q_i, \mathbb{0}, p_i) = q'_i$ 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 $i \in \{\text{I}, \text{II}\}$
- $F_\cap = \{((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) \mid (q_i, p_i) \in F_i \text{ for } i \in \{\text{I}, \text{II}\}\ \}$
- $F_\cup = \begin{array}{l} \{((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) \mid (q_\text{I}, p_\text{I}) \in F_\text{I},\ (q_\text{II}, p_\text{II}) \in Q_\text{II} \times P_\text{II}, \} \cup \\ \{((q_\text{I}, q_\text{II}), (p_\text{I}, p_\text{II})) \mid (q_\text{I}, p_\text{I}) \in Q_\text{I} \times P_\text{I},\ (q_\text{II}, p_\text{II}) \in F_\text{II}, \} \end{array}$

□

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

Theorem 9. *Emptiness of* NCNAs *can be solved in polynomial time. Inclusion, universality and equivalence of* NCNAs *are* EXPTIME*-complete.*

PROOF. The *emptiness* problem for 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. 8). Since the construction used in the proof assumed deterministic CNAs, and the determinization construction (Thm. 6) 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.

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 6 and then simulate it on the given membership query $\mathbb{w}$. 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 $O(|\mathbb{w}|)$ time and $O(d(\mathbb{w}))$ space where $d(\mathbb{w})$ is the hierarchical nesting depth of the colored nested word $\mathbb{w}$.

Theorem 10. *Membership of* NCNA*s can be solved in polynomial time. For a fixed* CNA $\mathcal{A}$ *and colored nested word* $\mathbb{w}$ *of length* ℓ *and depth* d*, the membership problem can be solved in time* $O(\ell)$ *and space* $O(d)$*.*

7. Additional Closure Properties

We shall see that regular languages of colored nested words enjoy additional closure properties. In particular, in Section 7.1 we shall see that they are closed under concatenation, finite iterations of concatenation (i.e. Kleene-*) as well as under the operations of prefixes, suffixes and reversal. In Section 7.2 we shall see that they are closed under color-respecting substitution, homomorphism and inverse homomorphism.

7.1. Closure under words operations

Theorem 11. *Regular languages of colored nested words are closed under concatenation and Kleene-*.*

PROOF. As per Lemma 4 we assume there is a unique bottom of stack symbol $\bot = (p_\bot, c_\bot) \in \Gamma$. Recall that color $c_\bot$ is greater than any color of the considered alphabet.

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 CNAs accepting them, respectively. An NCNA $\mathcal{B}$ for their concatenation can work by guessing a split of the given word $\mathbb{w}$ into two words $\mathbb{w}_1$ and $\mathbb{w}_2$ 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 $\mathcal{A}_2$. Note that when simulating $\mathcal{A}_2$ it will never pop $\bot$ out of the stack since its color is bigger than any that of any read letter, thus $\mathcal{A}_2$ will never encounter stack symbols that $\mathcal{A}_1$ pushed to the stack.

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 $\mathbb{w}$ to $\mathbb{w}_1, \ldots, \mathbb{w}_k$ for some k such that for each i, $\mathbb{w}_i$ is in L. As in the case of concatenation, whenever the constructed NCNA $\mathcal{C}$ reaches a final frontier it can non-deterministically move to an initial frontier of $\mathcal{A}$. Again since $\bot$ is of color greater than that of any read letter, when simulating a run on $\mathbb{w}_j$ the CNA will never pop $\bot$ and never reach a stack symbol pushed upon simulating $\mathbb{w}_i$ for $i < j$. □

For a letter $\mathbb{a} \in \hat{\Sigma}$ we define its *dual* $\overline{\mathbb{a}}$ as follows: $\overline{\dot{a}} = \dot{a}$, $\overline{(\!|_c a} = a_c|\!)$, and $\overline{a_c|\!)} = (\!|_c a$. If $\mathbb{w} = \mathbb{a}_1\mathbb{a}_2 \ldots \mathbb{a}_n$ then $\overline{\mathbb{a}}_n \ldots \overline{\mathbb{a}}_2\overline{\mathbb{a}}_1$ is the *reverse* of $\mathbb{w}$, and is denoted $\mathbb{w}^R$. Note that $(\mathbb{w}^R)^R = \mathbb{w}$. For a set of colored nested words L we define L^R to be the set $\{\mathbb{v} \mid \exists \mathbb{w} \in L \text{ s.t. } \mathbb{v} = \mathbb{w}^R\}$ and refer to it as the *reverse* of L.

Theorem 12. *Regular languages of colored nested words are closed under the operations of reverse.*

PROOF. Let $\mathcal{A} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)}, \delta^{(\varepsilon})$ be an ECNA. We can build an ECNA $\mathcal{R}$ for $\mathcal{L}(\mathcal{A})^R$ by switching the roles of initial and accepting frontiers, and dualizing the transition relation. Formally, $\mathcal{R} = (Q, P, F, I, \delta^{(}_{\mathcal{R}}, \dot{\delta}_{\mathcal{R}}, \delta^{)}_{\mathcal{R}}, \delta^{\varepsilon)}_{\mathcal{R}}, \delta^{(\varepsilon}_{\mathcal{R}})$ where the transitions of $\mathcal{R}$ are defined as follows:

1. if $q' \in \dot{\delta}(q, \dot{a})$ then $q \in \dot{\delta}_{\mathcal{R}}(q', \dot{a})$
2. if $(q', p, c) \in \delta^{(c}(q, (\!|_c a)$ then $q \in \delta^{c)}_{\mathcal{R}}(q', a_c|\!)), p, c)$
3. if $q' \in \delta^{c)}(q, a_c|\!)), (p, c))$ then $(q, p, c) \in \delta^{(c}_{\mathcal{R}}(q', (\!|_c a)$
4. if $q' \in \delta^{\varepsilon)}(q, (p, c'))$ then $(q, p, c') \in \delta^{(\varepsilon}_{\mathcal{R}_{c'}}(q')$ for every $c' < c$
5. if $(q, p, c) \in \delta^{(\varepsilon}_{c}(q')$ then $q \in \delta^{\varepsilon)}(q', p, c)$

The first three items are straight forward. The fourth item deals with ε-pop transitions on color c. In this case the considered colors c' on the stack are smaller than c. This corresponds to $\mathcal{A}$ allowing an unmatched $(\!|_{c'} a$ to be rescued by some closing letter of color $c > c'$. Therefore, for every $c' < c$, $\mathcal{R}$ needs to allow an unmatched $a_{c'}|\!)$ from the state q' that $\mathcal{A}$ arrived at, and it should take it to q the state that $\mathcal{A}$ started at. The transition $(q, p, c') \in \delta^{(\varepsilon}_{\mathcal{R}_{c'}}(q')$ does exactly this. The rational behind the fifth item is similar.

We can show by induction on the length of the word that $(q, p, c) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q', p', c')$ iff $(q', p, c') \overset{\mathbb{w}^R}{\Longrightarrow}_{\mathcal{R}} (q, p, c)$. Indeed the base case follows from switching roles of initial and final frontiers, and the induction step follows from the dualization of the transitions as per items 1 to 5. □

A word $\mathbb{u}$ is a *prefix* of $\mathbb{w}$ if there exists $\mathbb{v}$ such that $\mathbb{w} = \mathbb{u}\mathbb{v}$. In this case we also say that $\mathbb{v}$ is a *suffix* of $\mathbb{w}$. For a set of colored nested words L we define *pref*(L) and *suff*(L) to be the sets of prefixes and suffixes of words in L, respectively.

Theorem 13. *Regular languages of colored nested words are closed under the operations of prefix and suffix.*

PROOF. We show closure under suffix. Closure for prefix follows from Thm. 12 since $\mathit{pref}(L) = (\mathit{suff}(L^R))^R$.

Let $\mathcal{A} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)})$ be a DCNA recognizing L. We first note that given a pair of frontiers (q, p, c) and (q', p', c') we can compute whether $\exists \mathbb{w} : (q, p, c) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q', p', c')$ by answering the emptiness question for $\mathcal{A}'$ which is obtained from $\mathcal{A}$ by making the initial frontiers $\{(q, p, c)\}$ and the finals $\{(q', p', c')\}$. Checking emptiness can be done in polynomial time as stated in Theorem 9. It follows that we can compute in polynomial time the set $R = \{(q, p, c) \mid \exists (q_0, p_0, c_0) \in I \text{ and } \exists \mathbb{w} : (q_0, p_0, c_0) \overset{\mathbb{w}}{\Longrightarrow}_{\mathcal{A}} (q, p, c)\}$ of frontiers that are reachable from an initial frontier. We construct an NCNA $\mathcal{C}$ that is equivalent to $\mathcal{A}$ in all components but the initial frontiers. The initial frontiers of $\mathcal{C}$ are the reachable frontiers R of $\mathcal{A}$.

If $\mathcal{C}$ accepts a word $\mathbb{v}$ it means that there is a word $\mathbb{u}$ that leads $\mathcal{A}$ to reachable frontier (q, p, c) from which $\mathcal{C}$ started, and thus $\mathcal{A}$ accepts $\mathbb{u}\mathbb{v}$. If $\mathbb{v}$ is a suffix of some word in $\mathcal{A}$, say $\mathbb{w} = \mathbb{u}\mathbb{v}$ is accepted by $\mathcal{A}$, then there is a run of $\mathcal{C}$ on $\mathbb{v}$ starting from the frontier $\mathcal{A}$ reaches after reading $\mathbb{u}$. Since the run follows that of $\mathcal{A}$ on $\mathbb{v}$ from the point it finished reading $\mathbb{u}$, and $\mathcal{A}$ accepts $\mathbb{w}$, $\mathcal{C}$ will accept $\mathbb{v}$. □

7.2. Closure under substitution, homomorphism and inverse homomorphism

Let $\hat{\Sigma}$ and $\hat{\Sigma}'$ be two colored alphabets. We can assume w.o.l.g. that they use the same set of colors C. For every $\mathbb{a} \in \hat{\Sigma}$ let $H(\mathbb{a})$ 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 $\mathbb{w}' \in H(\dot{a})$ is weakly-matched, (2) for every $(\!|_c a \in (\!|\Sigma$ any word $\mathbb{w}' \in H((\!|_c a)$ is of the form $(\!|_c b\, \mathbb{v}$ where $\mathbb{v}$ is weakly-matched (3) for every $a_c|\!) \in \Sigma|\!)$ any word $\mathbb{w}' \in H(a_c|\!))$ is of the form $\mathbb{v}\, b_c|\!)$ where $\mathbb{v}$ is weakly-matched. When H maps every $\mathbb{a}$ 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 $h^{-1}(L') = \{\mathbb{w} \mid \exists \mathbb{w}' \in L'.\ h(\mathbb{w}) = \mathbb{w}'\}$. 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, $d|_i = d_i$.

Theorem 14. *Regular languages of colored nested words are closed under color-respecting substitution and homomorphism.*

Proof. Since homomorphism is a special case of substitution we only need to show closure under under color-respecting substitution.

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 characterizable by a CNA. Given a CNA $\mathcal{M}$ for L, and CNAs $\mathcal{M}_{\mathbb{a}}$ for $H(\mathbb{a})$ for every $\mathbb{a} \in \hat{\Sigma}$ 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 $\mathcal{M}_{\mathbb{a}}$, and the third component records a letter $\mathbb{a}$ that is the current guess of processed letter. The stack symbols of $\mathcal{M}'$ is the set $(\Gamma_{\mathcal{M}} \cup \{_\}) \times (\cup_{\mathbb{a} \in \hat{\Sigma}} \Gamma_{\mathcal{M}_{\mathbb{a}}})$. The NCNA works as follows. Roughly speaking, $\mathcal{M}'$ starts by guessing a letter $\mathbb{a} \in \hat{\Sigma}$, 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 $\mathcal{M}_{\mathbb{a}}$ in the second component of the state and stack. When $\mathcal{M}_{\mathbb{a}}$ reaches an accepting frontier $\mathcal{M}'$ may switch to guess a new letter $\mathbb{b}$ and continue with simulating $\mathcal{M}$ on $\mathbb{b}$ and $\mathcal{M}_{\mathbb{b}}$ on the next read letters.

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

Note that when $\mathbb{a} \in \dot{\Sigma}$, since $H(\mathbb{a})$ is weakly-matched, it will never encounter a foreign stack symbol, and the stack after processing $\mathcal{M}_{\mathbb{a}}$ will be the same as before processing it. When $\mathbb{a} \in (\!|\Sigma$, again no foreign stack symbol will be encountered and after processing $\mathcal{M}_{\mathbb{a}}$ the stack consists of one symbol more than before (whose first component corresponds to $\mathcal{M}$ and second to $\mathcal{M}_{\mathbb{a}}$). When $\mathbb{a} \in \Sigma|\!)$ after processing all letters but the last, $\mathcal{M}_{\mathbb{a}}$ 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 $\mathbb{a}$. If $\mathbb{a}$ 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 $\mathcal{M}_{\mathbb{a}}$ took into account the correct stack symbol(s) (where for $\mathcal{M}_{\mathbb{a}}$ it is the bottom of the stack, since that is what it would have encountered on the last letter of $H(\mathbb{a})$). □

Theorem 15. *Regular languages of colored nested words are closed 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 $\mathbb{a}' \in (\!|_c \Sigma'$ 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, of the same color as the symbol $\mathcal{M}'$ pushed. Similarly after reading a letter in $\dot{\Sigma}'$ the stack of $\mathcal{M}$ will be of same size as when starting, 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 $\delta^{(\!|}_{\mathcal{M}'}(q, \mathbb{a}') = (\delta^{(\!|}(q, h(\mathbb{a}'))|_1,\ \zeta(\mathbb{a}'))$ 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 $\delta^{|\!)}_{\mathcal{M}'}(q, \mathbb{a}', p) = \tilde{\delta}(q, h(\mathbb{a}'), p)$ where $\tilde{\delta}$ is the natural extension of δ to words. Finally, $\delta^{\varepsilon|\!)}_{\mathcal{M}'}(q, p) = \delta^{\varepsilon|\!)}(q, p)$. ▫

8. Grammar Characterization

In the following we provide a grammar characterization for regular languages of colored nested words. We first recall some basic definitions. A *context-free grammar* over an alphabet Σ is a tuple $G = (\mathcal{V}, S, \mathit{Prod})$, 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).

Definition 6. A grammar $(\mathcal{V}, S, \mathit{Prod})$ 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 $X \in \mathcal{V}$:

- $X \longrightarrow \varepsilon$
- $X^{|\!)} \longrightarrow \mathbb{a}\, Y^{|\!)(\!|}$ for $\mathbb{a} \in \Sigma|\!) \cup \dot{\Sigma}$
- $X^{|\!)} \longrightarrow (\!|_c a\ Y^c\ b_c|\!)\, Z^{|\!)(\!|}$
- $X^{(\!|} \longrightarrow \mathbb{a}\, Y^{(\!|}$ for $\mathbb{a} \in \dot{\Sigma} \cup (\!|\Sigma$
- $X^{(\!|} \longrightarrow (\!|_c a\ Y^c\ b_c|\!)\, Z^{(\!|}$
- $X^c \longrightarrow \dot{a}\, Y^{(\!|}$
- $X^c \longrightarrow \dot{a}\, Y^c$
- $X^c \longrightarrow (\!|_{c'} a\ Y^{c'}\ b_{c'}|\!)\, Z^c$
- $X^c \longrightarrow (\!|_{c'} a\ Y^c$ for $c' < c$
- $X^c \longrightarrow a_{c'}|\!)\ Y^c$ for $c' < c$

Intuitively, variables in $\mathcal{V}^{|\!)}$ and $\mathcal{V}^{(\!|}$ derive words with no pending calls and no pending returns, respectively, and variables in $\mathcal{V}^{c_i}$ derive weakly matched c_i-rooted words. In general a colored nested word $\mathbb{w}$ can be seen as the concatenation of three words $\mathbb{w}_1\mathbb{w}_2\mathbb{w}_3$ (some of which may be empty) such that the root variable deriving $\mathbb{w}_1$ is in $\mathcal{V}^{|\!)}$, the root variable deriving $\mathbb{w}_2$ is in $\mathcal{V}^c$ and root variable deriving $\mathbb{w}_3$ is in $\mathcal{V}^{(\!|}$. The grammar characterization of (uncolored) 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 define the boundaries for allowed recovered calls/returns of color $c' < c$.

Theorem 16. *A language L is derived by a* CNW*-grammar iff L is recognized by a* CNA.

Proof. Let $(\mathcal{V}, S, \mathit{Prod})$ be a CNW-grammar. We define an ECNA recognizing its derived language as follows. We extend the set of colors is C with an additional color $c_\perp$ greater than any other

color in C. The set of states is $\mathcal{V}$, the set of stack symbols is $(\Sigma \times \mathcal{V}) \cup (\bot \times C)$. Here we implicitly assume variables in $\mathcal{V}^c$ are colored c and variables in $\mathcal{V}^{(}$ and $\mathcal{V}^{)}$ are colored $c_\bot$. We use $\mathcal{V}^{>c}$ for the union of the sets $\mathcal{V}^{c'}$ for $c' > c$. The initial frontier is $(S, (\bot, c_\bot))$. 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 $X \longrightarrow (\!|_c a\ Y\, b_c|\!) Z$ is a production then $(Y, (b_c|\!|, Z)) \in \delta^{(}(X, (\!|_c a)$
3. If $X \longrightarrow \varepsilon$ is a production then $Z \in \delta^{)}(X, b_c|\!|), (b_c|\!|, Z))$ for every Z.
4. If $X \longrightarrow (\!|_c a\, Y$ is a production then $(Y, (\bot, c)) \in \delta^{(}(X, (\!|_c a)$ and
   $V^{c'} \in \delta^{\varepsilon)}(V^{c'}, (\bot, c))$ for every $V^{c'} \in \mathcal{V}^{>c}$.
5. If $X \longrightarrow a_c|\!|)\, Y$ is a production then $Y \in \delta^{)}(X, a_c|\!|), (\bot, c))$ and
   $(V^{c'}, (\bot, c)) \in \delta^{(\varepsilon}(V^{c'})$ for every $V^{c'} \in \mathcal{V}^{>c}$.

The first rule corresponds to consuming an internal letter. The second rule correspond to a c-rooted word, and records on the stack what the closing letter should be and to which state the ECNA should transition when it is read. The third 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. The fourth rule corresponds to pending/recovered calls and the fifth to pending/recovered returns. Both can occur within a c-rooted word, and the forth (fifth) can also occur at the end (beginning) of the word (resp.). Thus, they add both push/pop transitions as well as ε-pop/push transitions that can be performed from states corresponding to color $c' > c$.

Let $\mathcal{E} = (Q, P, \{(q_0, p_0)\}, F, \delta^{(}, \dot{\delta}, \delta^{)})$ be an ECNA for L. 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 $\mathcal{E}$ being in state q and expecting no pending returns, Y_q variables correspond to $\mathcal{E}$ being in state q and expecting no pending calls; and variables $Z^c_{q,q''}$ correspond to $\mathcal{E}$ 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 $(q', (p, c)) \in \delta^{(}(q, (\!|_c a)$ and $q''' \in \delta^{)}(q'', b_c|\!|), (p, c))$ 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 $q' \in \delta^{\varepsilon_{c'})}(q, (p, c))$ then $Y_q \longrightarrow (\!|_c a\, Y_{q'}$ and
   $Z^{c'}_{q,q''} \longrightarrow (\!|_c a\, Z^{c'}_{q',q''}$ for every $c' > c$, $q'' \in Q$.
6. If $(q', (p, c)) \in \delta^{(\varepsilon}(q)$ then $X_q \longrightarrow a_c|\!)\, X_{q'}$ and
   $Z^{c'}_{q,q''} \longrightarrow a_c|\!)\, Z^{c'}_{q',q''}$ for every $c' > c$, $q'' \in Q$.

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 weakly-matched c-rooted words. The fifth rule correspond to consuming a pending/recovered opening letter, and the sixth to consuming a pending/recovered closing letter. □

9. Conclusions

We have augmented the data model of nested words with a notion of *colors* (or *ranks*). The augmented model which we refer to as *colored nested words* allows capturing nested words seemingly impaired with abnormal termination. Examples of such data include executions of programs with exceptions and programs written in programming languages using whitespace to delimit program blocks. We argue that colored nested words cannot be transformed to properly matched nested words using nested words transducers, thus calling for a new automata model.

We define two automata models over colored nested words, CNA and *blind* CNA (BCNA). Both CNAS and BCNAS can be viewed as labeling the input data — the explicit graph representation. Each linear edge is labeled with the state the automaton is at when scanning it and each hierarchical edge is labeled by the stack symbol the automaton pushes when scanning its entrance point, where the stack symbol is colored by the color of the opening letter. In the case of a blind CNA we view the recovered pending calls as disconnected from the recovering point (and thus the automaton does not get to inspect the pushed symbols while popping them), whereas in the case of (sighted) CNAS we view the recovered pending edges as connected to the recovering point as in Fig. 4 (and thus the automaton does get to inspect the pushed symbols while popping them). We show that the two models have the same expressive power in spite of the fact that there can be unboudendly many stack symbols that the blind CNA cannot inspect while the sighted CNA can.

The grammar characterization of colored nested words offers an alternative view on the way colored nested words generalize nested words. 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 recovered calls/returns of any color smaller than c.

The motivation for introducing this model is to allow program verification of procedural code with exceptions. In a sense it also provides a theoretical model for handling software exceptions, and might be of use in discussions about variations of software exceptions mechanisms, their pros and cons.

References

Alur, R., Bouajjani, A., Esparza, J., 2016. Model checking of procedural programs. In: Handbook of Model Checking. Springer, to Appear.

Alur, R., Chaudhuri, S., 2010. Temporal reasoning for procedural programs. In: VMCAI. pp. 45–60.

Alur, R., Chaudhuri, S., Madhusudan, P., 2006. A fixpoint calculus for local and global program flows. In: POPL. pp. 153–165.

Alur, R., Chaudhuri, S., Madhusudan, P., 2011. Software model checking using languages of nested trees. ACM Trans. Program. Lang. Syst. 33 (5), 15.

Alur, R., Madhusudan, P., 2004. Visibly pushdown languages. In: STOC. pp. 202–211.

Alur, R., Madhusudan, P., 2009. Adding nesting structure to words. J. ACM 56 (3), http://robotics.upenn.edu/∼alur/Jacm09.pdf.

Caucal, D., Hassen, S., 2008. Synchronization of grammars. In: CSR. pp. 110–121.

Chaudhuri, S., Alur, R., 2007. Instrumenting C programs with nested word monitors. In: SPIN. pp. 279–283.

Crespi-Reghizzi, S., Mandrioli, D., 2012. Operator precedence and the visibly pushdown property. J. Comput. Syst. Sci. 78 (6), 1837–1867.
Debarbieux, D., Gauwin, O., Niehren, J., Sebastian, T., Zergaoui, M., 2013. Early nested word automata for xpath query answering on XML streams. In: CIAA'13. pp. 292–305.
Driscoll, E., Burton, A., Reps, T. W., 2011. Checking conformance of a producer and a consumer. In: SIGSOFT/FSE. pp. 113–123.
Filiot, E., Gauwin, O., Reynier, P.-A., Servais, F., 2011. Streamability of nested word transductions. In: Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS. pp. 312–324.
Filiot, E., Raskin, J.-F., Reynier, P.-A., Servais, F., Talbot, J.-M., 2010. Properties of visibly pushdown transducers. In: In Proc. 35th MFCS. pp. 355–367.
Filiot, E., Servais, F., 2012. Visibly pushdown transducers with look-ahead. In: SOFSEM 2012: Conf. on Current Trends The. and Prac. of CS. pp. 251–263.
Hague, M., Murawski, A. S., Ong, C. L., Serre, O., 2008. Collapsible pushdown automata and recursion schemes. In: LICS. pp. 452–461.
Madhusudan, P., Viswanathan, M., 2009. Query automata for nested words. In: MFCS. pp. 561–573.
Mozafari, B., Zeng, K., Zaniolo, C., 2012. High-performance complex event processing over xml streams. In: SIGMOD Conference. pp. 253–264.
Nowotka, D., Srba, J., 2007. Height-deterministic pushdown automata. In: MFCS. pp. 125–134.
Raskin, J.-F., Servais, F., 2008. Visibly pushdown transducers. In: Automata, Languages and Programming, ICALP 2008. pp. 386–397.
Staworko, S., Laurence, G., Lemay, A., Niehren, J., 2009. Equivalence of deterministic nested word to word transducers. In: In Proc. 17th FCT. pp. 310–322.
Thomo, A., Venkatesh, S., 2011. Rewriting of visibly pushdown languages for XML data integration. Theor. Comput. Sci. 412 (39), 5285–5297.

Appendix A. Proofs of lemmas of Section 5.2

Lemma 4 states that every CNA can be converted into an equivalent SCNA. Here is its proof.

PROOF OF LEMMA 4. Let $\mathcal{A} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\epsilon)})$ be a CNA. We define an SCNA $\mathcal{B} = (Q_{\mathcal{B}}, P_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}}, \delta^{(}_{\mathcal{B}}, \dot{\delta}_{\mathcal{B}}, \delta^{)}_{\mathcal{B}}, \delta^{\epsilon)}_{\mathcal{B}})$ as follows. Let $p_\perp$ be a fresh stack symbol, and recall that $c_\perp$ is bigger than all colors in C. The stack symbols $P_{\mathcal{B}}$ are $P \cup \{p_\perp\}$, thus $\Gamma_{\mathcal{B}} = P_{\mathcal{B}} \times C$. The set of states $Q_{\mathcal{B}}$ is $Q \times \Gamma_{\mathcal{B}}$. We use $g_\perp$ for $(p_\perp, c_\perp)$ and g, g', g'' for arbitrary elements of $\Gamma_{\mathcal{B}}$. The idea is that $\mathcal{B}$ records in the state the initial stack symbol. If $\mathcal{B}$ encounters $g_\perp$ 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 $\{((q,g), g_\perp) \mid (q,g) \in I\}$. The final frontiers are $F_{\mathcal{B}} = \{((q,g'),g) \mid (q,g) \in F\} \cup \{((q,g), g_\perp) \mid (q,g) \in F \cap I\}$. For the transition relation we have:

- $(q',g) \in \dot{\delta}_{\mathcal{B}}((q,g),\dot{a})$ if $q' \in \dot{\delta}(q,\dot{a})$
- $((q',g),g') \in \delta^{(}_{\mathcal{B}}((q,g),(\!|_c a)$ if $(q',g') \in \delta^{(}(q,(\!|_c a)$
- $(q',g) \in \delta^{)}_{\mathcal{B}}((q,g),a_c|\!),g')$ if $g' \neq g_\perp$ and $q' \in \delta^{)}(q,a_c|\!),g')$
- $(q',g) \in \delta^{)}_{\mathcal{B}}((q,g),a_c|\!),g_\perp)$ if $q' \in \delta^{)}(q,a_c|\!),g)$
- $(q',g) \in \delta^{\epsilon)}_{\mathcal{B}}((q,g),g')$ if $g' \neq g_\perp$ and $q' \in \delta^{\epsilon)}(q,g')$
- $(q',g) \in \delta^{\epsilon)}_{\mathcal{B}}((q,g),g_\perp)$ if $q' \in \delta^{\epsilon)}(q,g)$

□

Lemma 5 states that every CNA can be converted into an equivalent CNA with a single initial frontier. Here is its proof.

Proof of Lemma 5. By Lemma 4 we can convert the given CNA into a CNA $\mathcal{A}$ with $\mathcal{A} = (Q, P, I, F, \delta^{(}, \dot{\delta}, \delta^{)}, \delta^{\varepsilon)})$ where $I \subseteq Q \times \{p_\perp\}$ for some $p_\perp \in \Gamma$. 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 $F_{\mathcal{B}} = F \cup \{(q_I, p_\perp)\}$. 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 $\delta^{)}_{\mathcal{B}}(q_I, a_c|\!), p) = \cup_{q \in I} \delta^{)}(q, a_p|\!), p)$. □