

# A Note on Nested Words

Andreas Blass\*      Yuri Gurevich†

## Abstract

For every regular language of nested words, the underlying strings form a context-free language, and every context-free language can be obtained in this way. Nested words and nested-word automata are generalized to motley words and motley-word automata. Every motley-word automaton is equivalent to a deterministic one. For every regular language of motley words, the underlying strings form a finite intersection of context-free languages, and every finite intersection of context-free languages can be obtained in this way.

## 1 Introduction

In [1], Rajeev Alur and P. Madhusudan introduced and studied nested words, nested-word automata and regular languages of nested words. A nested word is a string of letters endowed with a so-called nested relation. Alur and Madhusudan prove that every nested-word automaton is equivalent to a deterministic one. In Section 2, we recall some of their definitions. We quote from the introduction to [1].

“The motivating application area for our results has been software verification. Given a sequential program  $P$  with stack-based control flow, the execution of  $P$  is modeled as a nested word with nesting edges from calls to returns. Specification of the program is given as a nested word automaton  $A$ , and verification corresponds to checking whether every nested word generated by  $P$  is

---

\*Math Dept, University of Michigan, Ann Arbor, MI 48109; ablass@umich.edu

†Microsoft Research, Redmond, WA 98052; gurevich@microsoft.com

accepted by  $A$ . Nested-word automata can express a variety of requirements such as stack-inspection properties, pre-post conditions, and interprocedural data-flow properties.”

In Section 3, we show that all context-free properties, and only context-free properties, can be captured by a nested-word automaton in the following precise sense. For every regular language of nested words, the underlying strings form a context-free language, and every context-free language can be obtained in this way. We continue the quotation from [1].

“If we were to model program executions as words, all of these properties are non-regular, and hence inexpressible in classical specification languages based on temporal logics, automata, and fixpoint calculi (recall that context-free languages cannot be used as specification languages due to nonclosure under intersection and undecidability of key decision problems such as language inclusion).”

Intersections of context-free languages naturally arise in applications. Think of multi-threaded programs for example. In Section 4, we generalize nested words to motley words: strings with several nested relations. We introduce motley-word automata and use the nested-word-automaton determinization procedure to show that every motley-word automaton is equivalent to a deterministic one. We show that every finite intersection (that is the intersections of finitely many) of context-free languages, and only such intersections, can be captured by a motley-word automaton in the following precise sense. For every regular (that is accepted by a motley-word automaton) language of motley words, the language of underlying strings is a finite intersection of context-free languages, and every intersection of context-free languages can be obtained in this way.

## 2 Preliminaries

To make this note self-contained, we recapitulate here some definitions from [1].

### 2.1 Nested words

A *nested relation*  $\nu$  over  $\{1, 2, \dots, k\}$  is a binary relation satisfying the following condition: if  $\nu(i, i')$  and  $\nu(j, j')$  and  $i \leq j$  then either  $i < i' < j < j'$  or else  $i < j < j' < i'$ . If  $\nu(i, j)$  then  $i$  is a *call* position and the *call-predecessor* of  $j$ , and  $j$  is a *return* position and the *return-successor* of  $i$ .

A *nested word* over an alphabet  $\Sigma$  is a pair  $(a_1 \dots a_k, \nu)$  where each  $a_i \in \Sigma$  and  $\nu$  is a nested relation over  $\{1, 2, \dots, k\}$ . A position  $i$  that is neither a call position nor a return position is an *internal* position of  $w$ .

$NW(\Sigma)$  is the set of nested  $\Sigma$ -words. A language of nested  $\Sigma$ -words is a subset of  $NW(\Sigma)$ .

### 2.2 NW automata

An *nw automaton*  $A$  over an alphabet  $\Sigma$  is a quadruple  $(Q, Q_{in}, \delta, Q_f)$  similar to a usual nondeterministic automaton with set  $Q$  of states, set  $Q_{in} \subseteq Q$  of initial states and set  $Q_f \subseteq Q$  of final states, except that  $\delta$  is not a single transition relation but a triple  $\langle \delta_c, \delta_i, \delta_r \rangle$  where

- $\delta_c \subseteq Q \times \Sigma \times Q$  is the transition relation for call positions,
- $\delta_i \subseteq Q \times \Sigma \times Q$  is the transition relation for internal positions, and
- $\delta_r \subseteq Q \times Q \times \Sigma \times Q$  is the transition relation for return positions.

The automaton  $A$  is *deterministic* if it satisfies the following conditions:

- There is exactly one initial state.
- $\delta_c$  is the relational form of a function from  $Q \times \Sigma$  to  $Q$ . In other words, if  $(p, a, q)$  and  $(p, a, q')$  belong to  $\delta_c$  then  $q = q'$ .
- $\delta_i$  is the relational form of a function from  $Q \times \Sigma$  to  $Q$ .
- $\delta_r$  is the relational form of a function from  $Q \times Q \times \Sigma$  to  $Q$ .

A *run* of  $A$  over an nested word  $(a_1 \dots a_k, \nu)$  is a sequence  $q_0, q_1, \dots, q_k$  of states such that  $q_0$  is an initial state, if  $i$  is a call position then  $(q_{i-1}, a_i, q_i) \in \delta_c$ , if  $i$  is an internal position then  $(q_{i-1}, a_i, q_i) \in \delta_i$ , and if  $i$  is a return position with call-predecessor  $j$  then  $(q_{i-1}, q_{j-1}, a_i, q_i) \in \delta_r$ . The run  $q_0, q_1, \dots, q_k$  is *accepting* if  $q_k$  is a final state.  $A$  *accepts* a word  $w$  if it has an accepting run on  $w$ . The *language*  $L(A)$  of  $A$  is the set of nested words accepted by  $A$ .

A language (that is a set) of nested words is *regular* if there is an nw automaton  $A$  such that  $L = L(A)$ . Two nw automata  $A, B$  are equivalent if  $L(A) = L(B)$ . Alur and Madhusudan prove that every nw automaton  $A$  is equivalent to a deterministic nw automaton [1, Theorem 1].

### 3 Nested-Word Automata and Context-Free Languages

In this section, we show that,

- for every regular language of nested words, the underlying strings form a context-free language, and
- every context-free language can be obtained in this way.

**Definition 1.**  $A$  is *call explicit* if  $\delta_c$  and  $\delta_i$  are disjoint.

**Lemma 2.** *For every nested-word automaton  $A = (Q, Q_{in}, \delta, Q_f)$ , there is a call-explicit nw automaton  $A'$  such that  $L(A') = L(A)$ . Furthermore, if  $A$  is deterministic then so is  $A'$ .*

*Proof.* The desired  $A' = (Q', Q'_{in}, \delta', Q'_f)$  where  $Q', Q'_{in}$  and  $Q'_f$  are  $Q \times \{0, 1\}$ ,  $Q_{in} \times \{0\}$  and  $Q_f \times \{0\}$  respectively. Further:

$$\begin{aligned} \delta'_c &= \{((p, d), a, (q, 1)) : (p, a, q) \in \delta_c \wedge d \in \{0, 1\}\} \\ \delta'_i &= \{((p, d), a, (q, 0)) : (p, a, q) \in \delta_i \wedge d \in \{0, 1\}\} \\ \delta'_r &= \{((p_1, d_1), (p_2, d_2), a, (q, 0)) : (p_1, p_2, a, q) \in \delta_r \wedge d_1, d_2 \in \{0, 1\}\} \end{aligned}$$

Clearly  $\delta'_c$  and  $\delta'_i$  are disjoint, and  $A'$  is deterministic if  $A$  is so. Every accepting run  $q_0, \dots, q_k$  of  $A$  on  $(a_1 \dots a_k, \nu)$  gives rise to an accepting run  $(q_0, d_0), \dots, (q_k, d_k)$  where  $d_i = 1$  if and only if  $i$  is a call position. And every accepting run  $(q_0, d_0), \dots, (q_k, d_k)$  of  $A'$  on  $(a_1 \dots a_k, \nu)$  gives rise to an accepting run  $q_0, \dots, q_k$  of  $A$ .  $\square$

**Definition 3.** The *projection*  $P(w)$  of a nested word  $w = (x, \nu)$  is the string  $x$ . The *projection*  $P(L)$  of a language  $L$  of nested words is the language  $\{P(w) : w \in L\}$ .

**Proposition 4.** *The projection of a regular nw language is context-free.*

*Proof.* Let  $L = L(A)$  where  $A$  is an nw automaton  $(Q, Q_{in}, \delta, Q_f)$ . We construct a (nondeterministic) pushdown automaton  $B$  with  $L(B) = P(L)$  that accepts on empty stack and final state. The sets of states, initial states and final states of  $B$  are  $Q$ ,  $Q_{in}$  and  $Q_f$  respectively. The stack alphabet of  $B$  is  $Q \cup \{\perp\}$  where  $\perp$  is the bottom-of-the-stack symbol. The transition function  $\Delta = \Delta_c \cup \Delta_i \cup \Delta_r$  where the components are as follows.

$$\begin{aligned}\Delta_i &= \{(p, a, p', q) : (p, a, q) \in \delta_i \wedge p' \in Q\} \\ \Delta_c &= \{(p, a, p', q, +p) : (p, a, q) \in \delta_c \wedge p' \in Q\} \\ \Delta_r &= \{(p, a, p', q, -) : (p, p', a, q) \in \delta_r \wedge p' \in Q\}\end{aligned}$$

The intended meaning of an instruction  $(p, a, p', q)$  is this: if the current state is  $p$ , the input symbol is  $a$  and the top stack symbol is  $p'$  then to go state  $q$  (and move one-letter to the right on the input word). The additional “ $+p$ ” means: push  $p$  onto the stack. The additional “ $-$ ” means: pop the stack.

An accepting run  $q_0, \dots, q_k$  of  $A$  on  $(a_1 \dots a_\ell, \nu)$  gives rise to an accepting run  $(q_0, U_0) \dots (q_k, U_k)$  of  $B$  on  $a_1 \dots a_k$  where  $U_0, \dots, U_k$  are stack contents defined inductively.  $U_0 = \perp$ . Let  $i > 0$ . If  $i$  is an internal position then  $U_i = U_{i-1}$ . If  $i$  is a call position then  $U_i = q_{i-1}U_{i-1}$ . Suppose that  $u$  is a return position with call-predecessor  $j$ . The number of call positions  $< i$  exceeds the number of return positions  $< i$ , and so  $U_{i-1} \neq \perp$ .  $U_i$  is obtained from  $U_{i-1}$  by popping the top symbol. It is easy to check that if  $(j, i) \in \nu$  then the top symbol of  $U_{i-1}$  is  $q_{i-1}$ . It is easy to see that the run  $(q_0, U_0) \dots (q_k, U_k)$  is indeed accepting.

Every accepting run  $(a_0, U_0) \dots (a_k, U_k)$  of  $B$  on  $a_1 \dots a_k$  gives rise to an accepting run  $q_0, \dots, q_k$  of  $A$  on a nested word  $(a_1 \dots a_\ell, \nu)$  where  $\nu$  consists of pairs  $(j, i)$  such that  $B$  pushes a symbol at step  $j$  and pops it at step  $i$ .  $\square$

**Proposition 5.** *Every context-free language is the projection of some regular nw language.*

*Proof.* Let  $L$  be a context-free language over an alphabet  $\Sigma$ . Without loss of generality,  $L$  is proper, that is it does not contain the empty word  $\epsilon$ . Indeed

if  $L' = L - \{\epsilon\}$  is the projection of a regular nw language  $M'$  then  $L$  is the projection of the regular nw language  $M' \cup \{\epsilon\}$ .

Since  $L$  is proper, there is a context-free grammar  $G$  for  $L$  that is in quadratic Greibach normal form [2, Theorem 3.2] which means the following. Let  $V$  be the set of the variables of  $G$ . Every production of  $G$  has the form  $X \rightarrow a$  or  $X \rightarrow aY$  or  $X \rightarrow aYZ$  where  $a \in \Sigma$  and  $X, Y, Z \in V$ .

We construct a particular nondeterministic pushdown automaton  $A$  that accepts  $L$ . Every instruction of  $A$  moves it one-letter to the right on the input string. The state set of  $A$  is  $V$ . The stack alphabet of  $A$  is  $V \cup \{\perp\}$  where  $\perp$  is the bottom-of-the-stack symbol. The initial state of  $A$  is the axiom  $S$  of  $G$ .

- Every production  $P = X \rightarrow aY$  of  $G$  gives rise to instructions  $(X, a, X', Y)$  of  $A$ : if the current state is  $X$ , the current input letter is  $a$  and the top stack symbol is  $X'$ , then go to state  $Y$  (without altering the stack). Here  $X'$  ranges over  $V \cup \{\perp\}$ .
- Every production  $P = X \rightarrow aYZ$  of  $G$  gives rise to one instruction  $(X, a, X', Y, +Z)$  of  $A$ : if the current state is  $X$ , the current input letter is  $a$  and the top stack symbol is  $X'$  then go to state  $Y$  and push  $Z$  onto the stack. Here  $X'$  ranges over  $V \cup \{\perp\}$ .
- Every production  $P = X \rightarrow a$  of  $G$  gives rise to instructions  $(X, a, Y, Y, -)$  of  $A$ : if the current state is  $X$ , the current input letter is  $a$  and the top stack symbol is  $Y$ , then pop the stack and go to state  $Y$ . Here  $Y$  ranges over  $V$ .

It is easy to see  $A$  indeed accepts  $L$ .

It remains to construct an nw automaton  $B$  with  $L(A) = P(L(B))$ . The desired  $B$  is

$$(V \times (V \cup \{\perp\}), \{S\} \times \{\perp\}, \delta, V \times \{\perp\})$$

where  $\delta$  is as follows. Intuitively a state  $(X, Y)$  of  $B$  means that  $A$  is in state  $X$  and the top stack symbol if  $Y$ .

- Every instruction  $(X, a, X', Y)$  of  $A$  gives one  $\delta_i$  instruction  $((X, X'), a, (Y, X'))$ .
- Every instruction  $(X, a, X', Y, +Z)$  of  $A$  gives one  $\delta_c$  instruction  $((X, X'), a, (Y, Z))$ .

- Every instruction  $(X, a, X', X', -)$  of  $A$  gives  $\delta_r$  instructions  $((X, X'), (Y, Y'), a, (X', Y'))$  where  $Y, Y' \in V$ .

We check that  $L(A)$  is the projection of  $L(B)$ . Let  $x = a_1 \dots a_k$ . First suppose that  $x \in L(A)$ . Consider an accepting computation  $(X_0, U_0), \dots, (X_k, U_k)$  of  $A$  on  $x$ . Here  $X_i, U_i$  are the state and the stack of  $A$  after reading the initial  $i$  letters of  $x$ . In particular  $X_0 = S$  and  $U_0 = U_k = \perp$ . Let  $X'_i$  be the top symbol of  $U_i$ . Let  $\nu$  be the set of pairs  $(j, i)$  such  $A$  pushes a symbol onto the stack during the  $j^{\text{th}}$  step and pops it during the  $i^{\text{th}}$  step. Then  $(X_0, X'_0), \dots, (X_k, X'_k)$  is an accepting run of  $B$  on  $(x, \nu)$ .

Second suppose that  $B$  accepts some  $(x, \nu)$  and let  $(X_0, X'_0), \dots, (X_k, X'_k)$  be an accepting run of  $B$  on  $(x, \nu)$ . By induction on  $i$ , we construct stack words  $U_0, \dots, U_k$  such that the top of  $U_i$  is  $X'_i$  and  $(X_0, U_0), \dots, (X_k, U_k)$  is an accepting run of  $A$  on  $x$ .  $U_0 = X'_0 = \perp$ . Let  $i > 0$ . If  $i$  is an internal position of  $(x, \nu)$  then  $U_i = U_{i-1}$ . If  $i$  is a call position then  $U_i = X'_{i-1}U_{i-1}$ . And if  $i$  is a return position then  $U_i$  is the result of popping the top symbol of  $U_{i-1}$ .  $\square$

## 4 Motley Words and Intersections of Context-Free Languages

We introduce the notion of motley words by generalizing the definition of nested words to allow for several nested relations. We introduce motley-word automata, or simply motley automata, and we use the nested-word-automaton determinization procedure of [1] to show that every motley automaton is equivalent to a deterministic one. We show that every finite intersection of context-free languages, and only such intersections, can be captured by motley automata in the following precise sense. For every regular (that is accepted by a motley automaton) language of motley words, the language of underlying strings is a finite intersection of context-free languages, and every finite intersection of context-free languages can be obtained in this way.

Fix an alphabet  $\Sigma$ . It is convenient to view a  $\Sigma$ -word  $a_1 \dots a_n$  as a vertex-labeled directed graph. The  $n$  vertices  $1, \dots, n$  are labeled with letters  $a_1, \dots, a_n$  respectively. And there are  $n - 1$  edges  $(1, 2), (2, 3), \dots, (n - 1, n)$ . The edges are unlabeled.

It is convenient to view a nested  $\Sigma$ -word  $\{a_1 \dots a_n, \nu\}$  as a digraph described above together with additional  $\nu$ -labeled edges  $(i, j)$  such that  $\nu(i, j)$  holds. Think of  $\nu$  as a color. Then  $\nu$ -labeled edges are  $\nu$ -colored. Think of unlabeled edges as uncolored.

**Definition 6.** A *motley  $\Sigma$ -word*  $w$  of dimension  $d$  is a  $\Sigma$ -word endowed with  $d$  nested relations. More explicitly,  $w$  is a tuple  $(a_1 \dots a_n, \nu_1, \dots, \nu_d)$  where each  $a_i \in \Sigma$  and  $\nu_1, \dots, \nu_d$  are nested relations on  $\{1, 2, \dots, n\}$ . Every nested relation is viewed as an edge-color. Hence the adjective *motley*.

Since the alphabet  $\Sigma$  is fixed, we may omit mentioning it explicitly.

**Definition 7.** A *motley automaton*  $A$  of dimension  $d$  is a direct product  $A_1 \times \dots \times A_d$  of  $d$  nw automata  $A_1, \dots, A_d$ . Since nw automata are in general non-deterministic, so are motley automata. A motley automaton  $A_1 \times \dots \times A_d$  is *deterministic* if every nw automaton  $A_k$  is so.

**Definition 8.** A *run* of  $A$  on a  $d$ -dimensional motley word  $(a_1, \dots, a_n, \nu_1, \dots, \nu_d)$  is a sequence

$$(q_0^1, \dots, q_0^d), (q_1^1, \dots, q_1^d), \dots, (q_n^1, \dots, q_n^d)$$

of states of  $A$  such that every  $(q_0^k, q_1^k, \dots, q_n^k)$  is a run of  $A_k$  on the nested word  $(a_1, \dots, a_n, \nu_k)$ . The run of  $A$  is *accepting* if every one of the  $d$  constituent runs is accepting.  $A$  *accepts* a  $d$ -dimensional motley word  $w$  if it has an accepting run on  $w$ . The *language*  $L(A)$  of  $A$  is the set of  $d$ -dimensional motley words accepted by  $A$ . Two motley automata  $A$  and  $B$  are *equivalent* if  $L(A) = L(B)$ .

**Theorem 9.** *For every motley automaton  $A$  there is a deterministic motley automaton  $B$  equivalent to  $A$ .*

*Proof.*  $A$  is the direct product  $A_1 \times \dots \times A_d$  of some nw automata  $A_1, \dots, A_d$ . By Theorem 1 in [1], every  $A_k$  is equivalent to a deterministic nw automaton  $B_k$ . It is easy to see that the direct product  $B_1 \times \dots \times B_d$  is equivalent to  $A$ .  $\square$

**Definition 10.** A *motley language* of dimension  $d$  is a language of  $d$ -dimensional motley words. A  $d$ -dimensional motley language  $L$  is *regular* if there is a  $d$ -dimensional motley automaton  $A$  such that  $L = L(A)$ . The *projection*  $P(w)$  of a motley word  $w = (x, \nu_1, \dots, \nu_d)$  is the  $\Sigma$ -word  $x$ . The *projection*  $P(L)$  of a motley language  $L$  of motley words is the language  $\{P(w) : w \in L\}$ .

**Lemma 11.** *Let  $A_1, \dots, A_d$  be nw automata and  $A$  be the motley automaton  $A_1 \times \dots \times A_d$ . Then  $P(L(A)) = \bigcap_k P(L(A_k))$ .*

*Proof.* Consider an arbitrary  $\Sigma$ -word  $x$ . First suppose that  $x \in P(L(A))$ . By the definition of projections, there exist nw relations  $\nu_1, \dots, \nu_d$  such that  $A$  accepts the motley word  $(x, \nu_1, \dots, \nu_d)$  and so there is an accepting run  $\rho$  of  $A$  on  $w$ . But then the constituent runs of automata  $A$  on nested words  $(x, \nu_k)$  are all accepting. Therefore  $x$  belongs to every  $P(L(A_k))$ .

Second suppose that  $x$  belongs to every  $P(L(A_k))$ . Then there are nested relations  $\nu_1, \dots, \nu_k$  such that every  $A_k$  accepts the nested word  $(x, \nu_k)$ . Let  $(q_0^k, q_1^k, \dots, q_n^k)$  be an accepting run of  $A_k$  on  $(x, \nu_k)$ . These  $d$  runs give rise to an accepting run

$$(q_0^1, \dots, q_0^d), (q_1^1, \dots, q_1^d), \dots, (q_n^1, \dots, q_n^d)$$

of  $A$  on  $(x, \nu_1, \dots, \nu_d)$ . Thus  $x \in P(L(A))$ . □

**Theorem 12.** *The projection of any regular motley language is a finite intersection of context-free languages. And the other way round, every finite intersection of context-free languages is the projection of some regular motley language.*

*Proof.* Let  $L$  be a regular motley language and  $d$  be the dimension of  $L$ . There exists a  $d$ -dimensional motley automaton  $A$  such that  $L = L(A)$ . The automaton  $A$  is the direct product of nw automata  $A_1, \dots, A_d$ . By Lemma 11,  $P(L) = \bigcap_k P(L(A_k))$ . According to the previous section, every  $L(A_k)$  is context free. Thus  $P(L)$  is a finite intersection of context free languages.

Let  $L = \bigcap_{k=1}^d L_k$  where every  $L_k$  is a context-free language. According to the previous section, there are nw automata  $A_1, \dots, A_d$  such that  $L_k = L(A_k)$ . Let  $A$  be the motley automaton  $A_1 \times \dots \times A_k$ . By Lemma 11,  $P(L(A)) = \bigcap_{k=1}^d P(L(A_k)) = L$ . Thus  $L$  is the projection of the regular motley language  $L(A)$ . □

## References

- [1] Rajeev Alur and P. Madhusudan, “Adding Nesting Structure to Words”, Tenth International Conference on Developments in Language Theory, 2006.
- [2] Jean-Michel Autebert, Jean Berstel and Luc Boasson, “Context-Free Languages and Push-Down Automata”, In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, volume I, chapter 3. Springer Verlag, Berlin Heidelberg, 1997