Closure Properties in the Class of Multiple Context Free Groups.

Robert P. Kropholler and Davide Spriano

September 11, 2017

Abstract

We show that the class of groups with \( k \)-multiple context free word problem is closed under amalgamated free products over finite subgroups. We also show that the intersection of two context free languages need not be multiple context free.

1 Introduction

Multiple context free languages (MCFLs) are a generalizations of context free languages introduced by linguists to better model natural languages [Pol84], but they can be used as a tool to better understand some classes of groups.

Given a presentation for a group \( G \), it is a natural question to ask whether two words represents the same element in \( G \). Using the elementary fact that \( g = h \leftrightarrow gh^{-1} = 1 \), this is equivalent to establish whether a given product of generators represents the identity element. One of the most successful strategy to tackle this question is to consider the set of all words that represents the trivial element, the so-called word problem, and study it via language theoretical instruments. A remarkable result of Muller-Schupp [MS81], which relies on results of Stallings and Dunwoody [Sta71, Dun85] shows that the class of groups that have context free word problem coincides with the class of virtually free groups. This motivates the study of groups whose word problem is multiple context free (MCF groups), hoping that the weakened restrictions on the word problem will encode a larger and more interesting class of groups.

The class of MCFLs is strictly larger than the class of CF languages, for example the language \( \{ a^n b^n c^n : n \in \mathbb{N} \} \) is MCF but not CF. However, it wasn’t clear until [Sal15], where it is proved that \( \mathbb{Z}^2 \) is MCF, that the difference can be seen on the level of groups, namely that there are groups with a word problem that is MCF, but not CF.

A recent result of Ho [Ho17] shows that the word problem for \( \mathbb{Z}^n \) is MCF, generalizing the result of Salvati [Sal15].

This leads to some natural questions about the closure properties of MCF groups: is the class of MCF groups closed under direct product, free product
or subgroups? Seki et al. [SMFK91] gives us the closure under subgroups and supergroups of finite index. Gilman, Kropholler and Schleimer [GKS17] showed that \( F_2 \times F_2 \) is not an MCF group, thus showing that the class is not closed under direct product.

In this paper we will prove the following result.

**Theorem 1.1.** Let \( G_1 \) and \( G_2 \) be groups whose word problem with multiple context free. Let \( H_i \) be a finite subgroup of \( G_i \), such that \( H_1 \cong H_2 \cong H \). Then \( G = G_1 \ast_H G_2 \) has a multiple context free word problem.

Since the class groups with regular word problem coincides with the class of finite groups, one could rephrase this result as saying that the class of MCF groups is closed under amalgamation over regular groups. This result is not true substituting regular groups with CF groups. Indeed, \( F_2 \times F_2 = (\mathbb{Z}_2 \ast_{\mathbb{Z}} \mathbb{Z}_2) \ast_{F_2} (\mathbb{Z}_2 \ast_{\mathbb{Z}} \mathbb{Z}_2) \).

### Acknowledgements

We greatly thank Bob Gilman for introducing us to the subject and making this project possible. This work was started at MSRI, Berkeley, where research is supported by the National Science Foundation under Grant No. DMS-1440140. The second author would like to thank UC Berkeley for inviting him as a visiting scholar. The first author would like to thank Alessandro Sisto for inviting me to complete this work at the ETH. Finally, we would like to thank Neil Fullarton for his invaluable work with a stapler.

### 2 Background

We are interested in the study of formal languages. In this section, we will give an introduction to formal languages and MCFLs. For a more comprehensive treatment, we refer to [HU69].

**Definition 2.1.** Given a finite set \( \Sigma \), \( \Sigma^* \) is the *free monoid over \( \Sigma \)*, i.e. the set of all finite words in \( \Sigma \) with the concatenation operation. We will denote with \( \varepsilon \) the trivial element of \( \Sigma^* \), namely the empty word.

**Definition 2.2.** Given a finite set \( \Sigma \), we say that a set \( L \subseteq \Sigma^* \) is a *language* over \( \Sigma \).

Since the definition of language is very broad, we will restrict our attention to languages that have a nice description. The reader should think of this as the same meta-distinction between continuous functions \( \mathbb{R} \rightarrow \mathbb{R} \) and continuous functions that can be phrased in terms of elementary functions.

Hence we want to prescribe a general recipe that will allow us to produce languages.
Chomsky grammars and hierarchy.

Definition 2.3. A Chomsky grammar $G$ is a tuple $(\Sigma, N, \delta, S)$ where $\Sigma$ and $N$ are (disjoint) finite sets, $S \in N$ and $\delta$ is a finite subset of $((\Sigma \cup N)^* \setminus \Sigma^*) \times (\Sigma \cup N)^*$. Namely, if $(x, y) \in \delta$, then $x$ contains at least one symbol of $N$. We call $\Sigma$ the set of terminals of $G$, $N$ the set of non terminals, $S$ the starting symbol and $\delta$ the production rules.

Notation. We will often use the following conventions: the elements of $\Sigma$ will be denoted by lower case letters (ex. \{a, b, c\}), the elements of $N$ by upper case letters (ex. \{A, B, S\}), and elements $\tau = (aB, BccA)$ of $\delta$ as $\tau: aB \rightarrow BccA$.

Given a grammar $G = (\Sigma, N, S, \delta)$, it is always possible to associate a (possibly empty) language $L(G) \subset \Sigma^*$. We will describe inductively the language $L(G)$.

Definition 2.4. Let $G = (\Sigma, N, S, \delta)$ be a grammar. We want to describe a subset $D(G) \subseteq (\Sigma \cup N)^*$ of derivable words.

- $S$ is derivable,
- for $u, v, w \in (\Sigma \cup N)^*$, if $uvw$ is derivable and the rule $v \rightarrow x$ is an element of $\delta$, then $uxw$ is derivable. In particular, we say that $uxw$ is derivable from $uvw$.

We say that a derivation (for $w_k$) is a chain of words $S = w_1, \ldots, w_k$ such that $w_{i+1}$ is derivable from $w_i$. The language associated to the grammar $G$ is the intersection $L(G) = D(G) \cap \Sigma^*$, namely all the derivable words that consists only of terminals symbols.

Example 2.5. Let $G = (\{a, b, c\}, \{A, B, S\}, S, \delta)$ be a grammar, where $\delta$ consist of the following rules:

- $\tau_1: S \rightarrow AB$,
- $\tau_2: A \rightarrow aAb$,
- $\tau_3: B \rightarrow ABc$,
- $\tau_4: A \rightarrow \epsilon$,
- $\tau_5: B \rightarrow \epsilon$.

To generate the language $L(G)$, we will try to understand the derivable words. We start with the symbol $S$. The only rule we can apply at the first step is $\tau_1$, yielding $AB$. Then we can substitute $A$ with $aAb$, using rule $\tau_2$, getting $aAbB$. Applying $\tau_2$ $k$ more times gives $a^k Ab^k B$. Rule $\tau_4$ gives $a^k b^k B$. Now, if we apply rule $\tau_3$, we will get $a^k b^k ABC$. We can repeat the process above and get some word of the form $a^{k_1} b^{k_1} \cdots a^{k_n} b^{k_n} Bc^m$. After applying rule $\tau_5$, we would get $a^{k_1} b^{k_1} \cdots a^{k_n} b^{k_n} c^m$, which is a string composed of non terminals only.

We now give a classification of some grammars.

Definition 2.6. A Chomsky grammar $G = (\Sigma, N, \delta, S)$ is called:
regular if all the elements of $\delta$ have the form $X \rightarrow wY$, where $X \in N$, $Y \in N \cup \{\varepsilon\}$, and $w \in \Sigma^*$;

context free if all the elements of $\delta$ have the form $X \rightarrow w$, where $X \in N$ and $w \in (\Sigma \cup N)^*$;

recursively enumerable otherwise.

The language $L(G)$ is regular (respectively context free or recursively enumerable) if $G$ is.

The intuitive idea that one should have about the above definition is the following: a derivation in a regular language consists of substituting the last letter of a word with a new string of letters. A derivation in a context free language consists of substituting a single letter (but not necessarily the last one) of a word with a new string of letters. The last case covers all other possibilities.

The gap between being context free and being recursively enumerable seems (and in fact is) very big. The class of multiple context free languages (MCFLs) that we are going to describe, is one of the classes that properly lives in this gap, namely properly contains context free languages, and is properly contained in the class of recursively enumerable languages [SMFK91].

As before, we are going to describe a grammar that defines the class of MCFLs. It should be noted that this will not be a Chomsky grammar. We start with the definition of linear rewriting function. The idea is very simple, but the definition may look a bit convoluted. Intuitively, a linear rewriting function is a function that “paste words together”, possibly adding some string of letters. For instance, if $a, b$ are letters and $v, w$ words, a linear rewriting function is $(v, w) \mapsto waabvb$.

Definition 2.7. Fix a finite alphabet $\Sigma$, and let $X = \{x_1, \ldots, x_n\}$ be a finite (possibly empty) set of variables. A rewriting on the variables $\{x_1, \ldots, x_n\}$ is a word $w \in (X \cup \Sigma)^*$. We say that a rewriting $w$ is linear if each element of $X$ occurs at most once.

Given a rewriting $w$, we can associate to it the function $f_w: (\Sigma^*)^n \rightarrow \Sigma^*$ that associates to each tuple $(u_1, \ldots, u_n)$ the word obtained substituting in $w$ each occurrence of $x_i$ with $u_i$. If $n = 0$, then $(\Sigma^*)^0 = \{\varepsilon\}$ and $f_w$ is the constant function $w$. A rewriting function is linear if it comes from a linear rewriting.

We say that a function $f: (\Sigma^*)^n \rightarrow (\Sigma^*)^m$ is a (multiple) rewriting function if it is a rewriting function in each component. A (multiple) rewriting function coming from rewritings $w_1, \ldots, w_m$ is linear if $w_1 \cdots w_m$ is linear.

Note that being linear in each component is not enough for a multiple rewriting function to be linear. In fact, the whole word $w_1 \cdots w_m$ must be linear, this implies that each variable $x_i$ appears in at most one of the $w_j$. In order to simplify notation, from now on we will call multiple rewriting functions simply rewriting functions.

Definition 2.8. A stratified set is a set $N$ equipped with a function $\| \cdot \|: N \rightarrow \mathbb{N} \setminus \{0\}$. The function $\| \cdot \|$ is called a dimension.
Definition 2.9. A multiple context free grammar (MCFG) on an alphabet \( \Sigma \) is a tuple \((\Sigma, N, S, F)\) satisfying the following:

- \( \Sigma \) is a finite set of terminals.
- \( N \) is a finite stratified set of non terminals.
- \( S \in N \) is the starting symbol such that \( |S| = 1 \).
- \( F \) is a finite set of elements of the form \((A, f, B_1, \ldots, B_s)\), where \( A, B_1, \ldots, B_s \) are elements of \( N \), and \( f : (\Sigma^*)^{|B_1|+\cdots+|B_s|} \to (\Sigma^*)^{|A|} \) is a linear rewriting function.

Given an element \( \tau = (A, f, B_1, \ldots, B_s) \) of \( F \), we will denote it by \( \tau = A \to f(B_1, \ldots, B_s) \).

We say that the grammar is \( k \)-MCF if \( |A| \leq k \) for all \( A \in N \).

As in the case of Chomsky grammars, given a MCFG \( H \), we want to associate a language \( L(H) \) to it.

Definition 2.10. Let \( H = (\Sigma, N, S, F) \) be a MCFG, and let \( A \in N \). We inductively define \( D_H(A) \subseteq (\Sigma^*)^{|A|} \) as follows: for each \( \tau \in F \):

- if \( \tau = A \to f(\varepsilon) \), then \( f(\varepsilon) \in D_H(A) \);
- if \( \tau = A \to f(B_1, \ldots, B_s) \) and \( y_1 \in D_H(B_1), \ldots, y_s \in D_H(B_s) \), then \( f(y_1, \ldots, y_s) \in D_H(A) \).

Definition 2.11. For a MCFG \( H = (\Sigma, N, S, F) \), we define the language associated to \( H \) as \( D_H(S) \). We say that a language \( L \) is a multiple context free language if there is a MCFG \( H \) such that \( L = D_H(S) \).

It is a well known fact that the class of context free language is not closed under intersection. However, it is natural to ask if intersection of context free languages are contained in a larger class. A possible guess is the class of multiple context free languages. We will show, however, that this is not the case.

Proposition 2.12 \([\text{SMFK91}]\). If \( L \) is a \( k \)-MCFL and \( R \) is a regular language, then \( L \cap R \) is again a \( k \)-MCFL.

Lemma 2.13 \([\text{SMFK91}]\) \( \text{Lemma 3.3} \). The language \( \{a^n b^n : n \in \mathbb{N}\} \subseteq \{a, b\}^* \) is a \( k \)-MCFL, but not a \((k - 1)\)-MCFL, for each \( k \in \mathbb{N} \).

Proof. This is a consequence of the pumping lemma for MCF languages which is Lemma 3.2 in [SMFK91]. \( \Box \)

Proposition 2.14. The intersection of two context free languages need not be a MCFL.

Proof. It is easily seen that the following two are CFL:

- \( L_1 = \{a^{n_1} b^{n_2} \cdots a^{n_k} b^{n_k} : n_1, \ldots, n_k, k \in \mathbb{N}^*\} \subseteq \{a, b\}^* \);
- \( L_2 = \{a^{n_1} b^{n_2} a^{n_2} \cdots b^{n_k} : n_1, \ldots, n_k, k \in \mathbb{N}^*\} \subseteq \{a, b\}^* \).
Intersecting these two languages, we get $L = \{(a^n b^n)^k : n, k \in \mathbb{N}\}$. Consider the family $\{R_k\}$ of regular languages defined as $R_k = \{(a^*b^*)^k\}$. Intersecting $L$ with each element of $\{R_k\}$, Lemma 2.13 gives that $L$ is not $k$-MCFL for each $k$.

3 Grammars and automata

The goal of this section is to explain the relation between grammars and automata. In what follows, an automaton should be thought as a “computer with limitations”, namely as a machine that can do some operations, but does not possess the power (usually memory) of a Turing machine. As in the case of grammars, an automaton is naturally associated to a language. The intuitive explanation for this is the following: an automaton is associated to an algorithm that, given a word, either “accepts” or “rejects” it. The language associated to an automaton is the set of all “accepted” words.

In what follows, we fix a finite alphabet $\Sigma$, and all the definitions are understood to be dependent on $\Sigma$. Recall that a partial function $f : A \rightarrow B$ is a map of sets defined on a subset $C \subseteq A$, called the domain of $f$.

**Definition 3.1.** A storage type is a tuple $T = (C, P, F, C_I)$ satisfying the following: $C$ is a set, called the set of storage configurations; $P$ is a subset of the power set $P(C)$, and the elements of $P$ are called predicates; $F$ is a set of partial functions $f : C \rightarrow C$ called instructions; and $C_I \subseteq C$ is a set of initial configurations.

**Definition 3.2.** An automaton with storage is a tuple $M = (Q, T, I, \delta)$, where $Q$ is a finite set of states, $T = (C, P, F, C_I)$ is a storage type, $I$ is a tuple $I = (q_I, c_I, Q_F)$ where $q_I \in Q$ is the initial state, $Q_F \subseteq Q$ are the final states, and $c_I \in C_I$ is the initial storage configuration. Finally $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times P \times F \times Q$ is a finite set of transitions.

**Definition 3.3.** Given an automaton with storage $M = (Q, T, I, \delta)$, we define the graph realisation of $M$, denoted by $\Gamma(M)$, as the following oriented labelled graph:

- The vertices of $\Gamma(M)$ are the elements of $Q \times C$.
- To each $\tau = (q_1, \sigma, p, f, q_2) \in \delta$, we associate an oriented edge between the pair $((q_1, c_1), (q_2, c_2))$ if $c_1 \in p$, $f(c_1) = c_2$. In that case the label of this edge is $\sigma$.

Note that $f$ is a partial function, so with $f(c_1) = c_2$ we are also asking that $c_1$ is in the domain of $f$.

**Definition 3.4.** Let $\Sigma$ be an alphabet, and let $g : (\Sigma \cup \{\varepsilon\})^* \rightarrow \Sigma^*$ be the morphism of monoids that sends $\varepsilon$ to the empty word, and is the identity on all the other generators. Given a word $w \in \Sigma^*$ we say that a word $w' \in (\Sigma \cup \{\varepsilon\})^*$ is an $\varepsilon$-expansion of $w$ if $g(w') = w$. 

6
Definition 3.5. Given an automaton with storage $\mathcal{M}$ we define a language $L(\mathcal{M}) \subseteq \Sigma^*$ as follows. A word $w$ is in $L(\mathcal{M})$ if and only if there is an oriented path $\gamma$ in $\Gamma(\mathcal{M})$ starting from $(q_I, c_I)$ and ending in a vertex $(q, c)$ with $q \in Q_F$ such that the word formed by the labels of $\gamma$ is an $\varepsilon$-expansion of $w$.

In order to improve the readability of the above definitions, we will provide a fairy tale example to clarify the role of the various entities above.

Imagine there is a group of children playing a treasure hunt in a town. The town is finite (as towns tend to be) and each block of the town is one of the states $Q$. The children possess an extremely bad memory, but luckily each of them is equipped with a book to write notes. The set of all possible books open on all possible pages is the set of storage configurations $C$. The set $P$ can contain a single element for instance, say the set of all books open on a blank page.

Now suppose that there is a voice guiding the game in order to help the children find the treasure, and in particular every now and then is reading out loud some hint (the alphabet $\Sigma$). The voice represents the word $w$ in the alphabet. When a hint (letter) is read, the children will perform an action, and the possible actions are encoded in the set $\delta$.

At the start of the game, the children will all be in the central block of the city $(q_I)$, with an empty book open on the first page $(c_I)$, and the treasure will be buried in some blocks $(Q_F)$ of the city. The typical turn (an element of $\delta$) will work as follows: every child will check on which block they are standing on (element of $Q$), then listen to what the voice is saying ($\Sigma$), and look if there is something written on the book ($P$). After doing that, they may write something on the book ($F$), and go to a new block ($Q$). If at any time a child cannot perform an action, then he or she is disqualified from the game. When the voice stops giving hints, each child will start digging exactly where they stand and see if a treasure is found.

If at least one child has found a treasure, then the instructions were correct (and hence the word $w$ is accepted).

Let’s start with some famous automata in order to familiarize with the above concepts.

Definition 3.6. A trivial storage is a storage type $T = (C, P, F, C_I)$ with $C = \{C_I\}$, $P = \{C\}$ and $F = \{\text{id}\}$.

Definition 3.7. A finite state automaton (FSA) is an automaton with storage with trivial storage.

It is a very easy exercise to see that a FSA is completely described by a finite oriented graph with edges labeled by elements of $\Sigma$ (and not $\Sigma \cup \{\varepsilon\}$).

The following theorem forms a bridge between languages associated to grammars, and languages accepted by automata.

Theorem 3.8. [HU69] For a language $L \subseteq \Sigma^*$ the following are equivalent:

- $L$ is associated to a regular grammar;
· $L$ is accepted by a FSA.

**Definition 3.9.** A **push-down storage** over a finite alphabet alphabet $\Omega$ is a storage type $T = (C, P, F, C_I)$ where:

- $C = \Omega^*$.
- We define the set $\text{equals}_\omega$, as the set of words in $\Omega^*$ that end with $\omega$ (note that $\text{equals}_\varepsilon$ is the set $\{\varepsilon\}$). Then $P = \{\text{equals}_\omega : \omega \in \Omega \cup \{\varepsilon\}\}$.
- We define the function $\text{push}_\omega : \Omega^* \to \Omega^*$ that sends $x$ to $x\omega$. We also define a partial function $\text{pop}_\omega : \text{equals}_\omega \to \Omega^*$ that sends $x\omega$ to $x$. Then $F = \{\text{Id}\} \cup \{\text{pop}_\omega, \text{push}_\omega : \omega \in \Omega\}$.
- $C_I = \{\varepsilon\}$.

The intuitive idea behind the push-down storage is to have a stack of papers that can grow arbitrarily large, but the automaton can read only what is written on the top-most paper. This corresponds to the predicate $\text{equals}_\omega$. Then one can put another paper on top with the letter $\omega'$ ($\text{push}_\omega$) or remove the old one ($\text{pull}_\omega$). Note that the alphabet $\Omega$ is, in general, not the same as $\Sigma$.

**Definition 3.10.** A **push-down automaton** is an automaton with storage with push-down storage.

**Theorem 3.11.** [Cha62] For a language $L \subseteq \Sigma^*$ the following are equivalent:

- $L$ is associated to a context-free grammar;
- $L$ is accepted by a push-down automaton.

We now want to describe the last automaton we are interested in, namely the tree-stack automaton.

**Definition 3.12.** Let $S$ be a set. If $uv \in S^*$ we say that $u$ is a **prefix** for $uv$. Given a set $D \subseteq S^*$ we say that $D$ is prefix-closed if for each word $w \in D$, all the prefixes of $w$ are in $D$. Similarly, we say that $v$ is a **suffix** for $uv$.

**Definition 3.13.** Given an alphabet $\Omega$, an $\Omega$-**tree** is a partial function $T : N^* \to \Omega \cup \{\gamma\}$ such that $\text{domain}(T) \subseteq N^*$ is prefix-closed and $T^{-1}(\gamma) = \{\varepsilon\}$.

Note that, this corresponds to a rooted tree, in the usual graph-theory sense, where each edge is labeled by a natural number, the root is labeled by the symbol $\gamma$ and every other vertex is labeled by an element of $\Omega$.

**Definition 3.14.** An $\Omega$-**tree with a pointer** is a pair $(T, p)$ such that $T$ is an $\Omega$-tree and $p \in \text{domain}(T)$.

One should think of the pointer as a selected vertex of the tree. Figure 1 may provide some clarification.
Notation. Let $F : C \to X$ be a partial function, and let $c \notin \text{domain}(F)$. Then we define $F[c \mapsto x]$ as the partial function defined on $\text{domain}(F) \cup \{c\}$, that agrees with $F$ on $\text{domain}(F)$ and sends $c$ to $x$.

Definition 3.15. A \textit{tree-stack storage} over a finite alphabet alphabet $\Omega$ is a storage type $T = (C, P, F, C_I)$ where:

- $C = \{(T, p) : (T, p) \text{ is an } \Omega\text{-tree with pointer}\}$.
- For $\omega \in \Omega \cup \{\gamma\}$, we set $\text{equals}(\omega) = \{(T, p) \in C : T(p) = \omega\}$ and $\text{notequals}(\omega) = \{(T, p) \in C : T(p) \neq \omega\}$.

Then $P = \{\text{equals}(\omega), \text{notequals}(\omega) : \omega \in \Omega \cup \{\gamma\}\} \cup \{C\}$.

- For $n \in \mathbb{N}$ and $\gamma \in \Omega$, we define the following partial functions:
  - $\text{push}_n(\gamma) : \{(T, p) : pn \notin \text{domain}(T)\} \to C$ as the map $(T, p) \mapsto (T[pn \mapsto \gamma], pn)$.
  - $\text{up}_n : \{(T, p) : pm \in \text{domain}(T)\} \to C$ as the map $(T, p) \mapsto (T, pn)$.
  - $\text{down} : C - \text{equals}_\gamma \to C$ as the map that sends $(T, pm)$ to $(T, p)$, for $m \in \mathbb{N}$.
  - $\text{set}_\gamma : C - \text{equals}_\gamma \to C$ as the map that sends $(T, p)$ to $(T', p)$, where $T'$ is obtained by $T$ changing the value of $p$ to $\gamma$.

Then $F = \{\text{Id}, \text{push}_n(\gamma), \text{up}_n, \text{down}, \text{set}_\gamma : \gamma \in \Omega, n \in \mathbb{N}\}$.

- $C_I = \{(\varepsilon \mapsto \gamma, \varepsilon)\}$.

Notation. For a subset $F$ of $\Omega$, we will write $\text{notequals}(F)$ to indicate the finite union of $\{\text{equals}(\omega) : \omega \in \Omega - F\}$. In particular, with $(q, a, \text{notequals}(F), f, q')$ we will indicate the finite set of rules $\{(q, a, \text{equals}(\omega), f, q') : \omega \in \Omega - F\}$.

Definition 3.16. A \textit{tree-stack automaton} is an automaton with storage with tree-stack storage.
Definition 3.17. We say that a tree-stack automaton is \(k\)-restricted if for any \(p \in \mathbb{N}^+, \, n \in \mathbb{N}\) and any path in the graph realisation \(\Gamma(M)\), starting at \((q_I, c_I)\) the following holds. There are at most \(k\) edges of the form \((q_1, (T_1, p))\) to \((q_2, (T_2, pn))\), where \(q_1, q_2 \in Q\) and \(T_1, T_2\) are tree-stacks.

Theorem 3.18. \([Den16]\) For a language \(L \subset \Sigma^*\) the following are equivalent:

\(\cdot\) \(L\) is associated to a \(k\)-MCFG;
\(\cdot\) \(L\) is accepted by a \(k\)-restricted tree-stack automaton.

Definition 3.19. A tree-stack automata is cycle-free if for every non-trivial loop in the graph realisation \(\Gamma(M)\), there is at least one push, up or down command.

Lemma 3.20. \([Den16]\) Given a \(k\)-restricted tree-stack automaton \(M\) there exists a tree-stack \(k\)-restricted automaton \(M'\) such that \(L(M) = L(M')\) and \(M'\) is cycle-free.

4 Closure under free products

In this section we prove that the class of groups whose word problem is multiple context free is closed under free products. To do this we will show that given \(G_1\) and \(G_2\) with multiple context free word problem we can construct a tree-stack automaton which accepts the word problem for \(G_1 * G_2\).

Lemma 4.1. Let \(M\) be a tree-stack automaton accepting the language \(M\). Then there exists a tree-stack automaton \(M'\) such that \(L(M') = L\) and \(M'\) accepts a non-empty word only if the tree-stack storage is in the state \((T, \varepsilon)\) for some \(\Omega\)-tree \(T\).

Proof. We build a new automaton which accepts the same language as follows.

Add two extra states \(q_f, \bar{q}_f\) to our automaton. We add the following transitions to \(\delta\).

\[
\begin{align*}
(q, \varepsilon, C, \text{Id}, q_f), & \forall q \in Q_F \\
(q_f, \varepsilon, C, \text{down}, q_f) \\
(q_f, \varepsilon, \text{equals}_\gamma, q_f)
\end{align*}
\]

We change the set of accept states to \(\{\bar{q}_f\}\). The language accepted by this new automaton is the same language as before. It should be noted that the new automaton has a single accept state and if \(M\) was cycle-free, then so is \(M'\).

It will also be useful to know that the amount of time spent at any vertex in the tree-stack is uniformly bounded.

Definition 4.2. A run in a tree-stack automaton is a path in the graph realisation. This can be seen as a valid sequence of instructions.

An accepted run is a run which ends in an accepted state.
Lemma 4.3. If $M$ is a $k$-restricted cycle-free tree-stack automaton, then there is an $n$ such that, for each $p \in \mathbb{N}^*$ and each path in the graph realisation of $M$ starting at $(q_1, c_1)$, there are at most $n$ vertices in the run of the form $(q, (T, p))$, where $q$ and $T$ may vary.

Proof. Consider the two possibilities for entering a vertex of the form $(q, (T, p))$, where $p$ is fixed and $q$ and $T$ may vary. Either we have an edge $(q_1, (T_2, pm)) \rightarrow (q, (T, p))$ or $(q_2, (T_2, \tilde{p})) \rightarrow (q, (T, p))$, where $\tilde{p}l = p$ for some $l$. There are only $k$ possibilities of the second instance since the automaton is $k$-restricted.

In the first instance, there must have been an edge of the form $(q', (T', p)) \rightarrow (q'', (T'', pm))$ previously in the path. There are at most $k$ such edges by $k$-restrictedness. Since $\delta$ is finite there can only be a finite number of instructions that contain a push command. Therefore, there are a bounded number of choices for $m$.

We will not require the exact bound, however, it can be calculated. A good estimate is $k$(number of push commands)(length of the longest path in the automaton with no movement in the tree).

Let $G_1, G_2$ be groups with multiple context free word problem, we now create the automaton which will accept the word problem for $G_1 \ast G_2$. Ideally, one would like to take the “free product” of the automata. However, this will results in something infinite. The key idea is to do this at the level of the tree-stack storage only.

Theorem 4.4. If $G_1$ and $G_2$ are groups with multiple context free word problem, then $G_1 \ast G_2$ has multiple context free word problem.

Proof. Let $W_i$ be the word problem in $G_i$ and $W$ be the word problem in $G_1 \ast G_2$. Let $\mathcal{M}_i = (Q_i, T_i, I_i, \delta_i)$, where $T_i$ is a tree-stack storage over the alphabet $\Omega_i$ and $I_i = (q_i^f, c_i^f, Q_{F_i} = \{q_i^f\})$ be an automaton recognising the language $W_i$.

We will assume that these automata are $k$-restricted, cycle-free and accept a word if and only if the stack pointer is at the root. Let $n$ be the maximum of the two bounds obtained from Lemma 4.3 applied to $\mathcal{M}_1$ and $\mathcal{M}_2$.

We now define the automaton $\mathcal{M}$ that will recognise the language $W$. The states of $\mathcal{M}$ are $Q = Q_1 \sqcup Q_2 \sqcup \{S, F\}$, the storage type $T$ is the set of tree-stacks on the alphabet $\Omega = \Omega_1 \sqcup \Omega_2 \sqcup (Q \times \{\Box_1, \Box_2\})$. The initial state is $S$, with empty initial tree and the final state is $F$. The transitions are $\delta = \delta_1 \sqcup \delta_2 \sqcup \delta_3$ where $\delta_i'$ is $\delta_i$ where any $\gamma$ symbol is replaced with a set of instructions one for
each element of $Q \times \{\Box_i\}$ and

$$\delta_3 = \{(S, \varepsilon, \text{equals}(\gamma), \text{Id}, F),$$

$$(S, \varepsilon, C, \text{push}_1(S, \Box_1), q_1^1), (S, \varepsilon, C, \text{push}_2(S, \Box_2), q_2^1),$$

$$(q_1^1, \varepsilon, \text{equals}(S, \Box_1), \text{down}, S), (q_2^1, \varepsilon, \text{equals}(S, \Box_2), \text{down}, S)\} \cup$

$$\{(q, \varepsilon, \text{equals}(q', \Box_1)), \text{push}_1((q, \Box_2), q_1^1) :$$

$q \in Q_1, q' \in Q_2, i \in \{-1, \ldots, -n\}\} \cup$

$$\{(q, \varepsilon, \text{equals}(q', \Box_2)), \text{push}_2((q, \Box_1), q_2^1) :$$

$q \in Q_2, q' \in Q_1, i \in \{-1, \ldots, -n\}\} \cup$

$$\{(q_1^1, \varepsilon, \text{equals}(q', \Box_1)), \text{down}, q' : q' \in Q_2\} \cup$

$$\{(q_2^1, \varepsilon, \text{equals}(q', \Box_2)), \text{down}, q' : q' \in Q_1\} \cup$$

The reader should note that tree-stacks were defined with $\mathbb{N}$ and negative labels have been used above. One should note that $\mathbb{Z}$ is countable so the labels
can be made positive.

We want to show that $W = L(M)$.

The way the automaton above works is as follows. We start with our word and move to one of the automata $M_1$ or $M_2$, say $M_1$. We then read a word in $\Sigma_1$ and move in this automaton as usual. When we come to a letter from $\Sigma_2$ we move to the automaton $M_2$ recording the state $q \in Q_1$ where we left $M_1$ and opening a new branch on the tree. Later we will read a letter of $\Sigma_1$, if we do this at the final state of $M_2$ then we move back to $q$, otherwise we open a new branch and move to $q_1$ and continue this process.

An accepted run $\Lambda$ of the automaton will have the pointer start and end at the root of the tree-stack. Let $T_f$ be the final tree-stack for the run. We can colour the non-root vertices of $T_f$ red and blue as follows. Colour a vertex red if the label is from $\Omega_1 \cup (Q_2 \times \{\square_1\})$ and blue otherwise. Note that after each instruction there is a tree-stack which embeds, as a graph, into $T_f$. Since the only set commands are to be found in $\delta_1'$ and $\delta_2'$, one could colour a vertex upon creation, the above embedding will then be colour preserving.

There is a subtree $T_c \subset T_f$ of a single colour whose complement is connected.

For each instruction there are two possible pointers, these can be viewed as vertices of $T_f$. Let $\Theta$ be the instructions in $\Lambda$ such that both pointers are in $T_c$. We claim that all the elements of $\Theta$ are consecutive. This is because there are no up commands with negative labels, so once we leave $T_c$ there is no way to return. Note that $\Theta$ start at the initial state of one of the automaton and ends at the corresponding final state. In particular, it can be viewed as an accepted run in $M_i$ and the subword $v$ of the run $\Lambda$ associated to $\Theta$ is an element of $W_i$.

Using the above, $\Lambda$ decomposes as $\Lambda_1 \theta_1 \Theta \theta_2 \Lambda_2$, where $\theta_1 \in \delta_3$ is an instruction containing a push command and $\theta_2 \in \delta_3$ is an instruction containing a down command. However, to leave the tree $T_c$, $\theta_1$ and $\theta_2$ pair up, by which we mean that the state of the automaton and the pointer before $\theta_1$ and after $\theta_2$ are the
same. Also, the tree-stacks outside $T_c$ remains unchanged. Thus, $\Lambda_1 \Lambda_2$ is an accepted run of $M$. As a consequence, we have that the word $w$ corresponding to the run $\Lambda$ decomposes as $w_1vw_2$, where $v$ is an element of $W_i$ and $w_1w_2$ is accepted by $M$. By considering words that are trivial in $G_1 \ast G_2$, we have that if $w_1w_2$ is an element of $W$, then so is $w$.

For the base case, note that if $T_c = T_f$, then $w \in W$. Thus by induction on the number of maximal one-colored subtrees, $L(M)$ is a subset of $W$.

For the other direction, we will use induction on the free product length of the word $w \in W$. The free product length of $w$ is the $p$ such that $w = w_1w_2 \ldots w_p$ and if $w_i \in W_j$, then $w_{i+1} \notin W_j$.

It is clear that words of length 1 are in the language $L(M)$.

If $w = w_1 \ldots w_p$ has length $p$ and is an element of $W$, then there is an $i$ such that $w_i$ is an element of $W_j$. We will assume that $w_i \in W_1$. The run the machine will take is as follows, make the run for the word $w_1 \ldots w_i-1 w_{i+1} \ldots w_p$, which exists by induction hypothesis. At the point where the word $w_i$ is read we will open a new tree and move to the automaton $M_1$ following a run for this word.

This run will finish at the root of the new tree and then return to the automaton $M_2$ to continue the run where it left off.

To make sure that we can do this process we have to be able to push a new edge at the correct moment. This may not be possible if we have already pushed $n$ edges at this vertex. However, we assumed that the automaton $M_1$ can only spend a uniformly bounded amount of time at any vertex and we added more push commands than this bound. Thus, there will always be a run for the word $w_1 \ldots w_i-1 w_{i+1} \ldots w_p$, where we can make a push at the desired moment. \(\square\)

In fact in the proof we have shown a slightly stronger result.

**Corollary 4.5.** If $G_1$ and $G_2$ are groups whose word problem is $k$-MCF, then the word problem in $G_1 \ast G_2$ is $k$-MCF.

**Proof.** It is clear from the proof of Theorem 4.4 that the automaton constructed is $\max\{k_1, k_2\}$-restricted. Indeed, all the instruction that contains commands up are contained in $\delta_1' \cup \delta_2'$. Applying instructions contained in $\delta_1'$ will not move the pointer to a vertex of a different colour (where the colouring is defined as in the proof of Theorem 4.4). Thus, if a vertex is contained in the interior of a one-colored subtree, say the colour corresponding to $W_1$, then that vertex will satisfy the $k_1$-restriction condition. \(\square\)

## 5 Amalgamated Free Products

In this section we generalise the previous result to show that the class of groups with multiple context free word problem is closed under amalgamation over finite subgroups.

The idea is similar to the previous proof, there are however more details. We feel that the interested reader should understand the proof of Theorem 4.4.
which encapsulates most of the details in an easier setting. The key idea is the following:

**Proposition 5.1.** Let $G$ be a group with multiple context free word problem. Let $H$ be a finite subset of $G$. Then $\{w \in \Sigma^* : w \text{ represents an element of } H\}$ is a multiple context free language.

**Proof.** For each $h \in H$, let $v_h$ be a word representing $h^{-1}$ in $\Sigma$. Let $R = \{v_h : h \in H\}$. Since $H$ is a finite set, so is $R$. Let $R'$ be the set of (possibly empty) suffixes of words in $R$. Let $\mathcal{M} = (Q, T, I, \delta)$ be an automaton recognising the word problem in $G$ with start state $q_I$ and a single final state $q_f$, where $T$ is the set of tree-stacks over the alphabet $\Omega$. Assume that this automaton has been modified as in Lemma 4.1.

The idea is the following: let $w$ be the input word. We will build an automaton that will "guess" an element of $H$, say $h$, and then proceed to process the word $v_hw$ in $\mathcal{M}$. The way this is done, is by adding a "second variable" to the states. The second variable represents the new word that is inserted. If the second variable is empty, then the automaton acts exactly as before. Otherwise, if the automaton is in a state $(q, v)$, where $v = a_1 \cdots a_n$ is a (non trivial) word, the automaton acts as if it was in the state $q$ and the first letter of $v$ (that is, $a_1$) is read. Then the second variable becomes $a_2 \cdots a_n$.

More formally. We will build a new automaton $\mathcal{M}' = (Q', T', I', \delta')$ as follows. The set of states $Q'$ will be $(Q \times R') \sqcup \{S\}$. The storage $T'$ will be tree-stacks over $\Omega$. The set of transitions $\delta'$ will consist of four types of transformation:

$$
\begin{align*}
(S, \varepsilon, \text{equals}(\gamma), \text{Id}, (q_I, v)) & \quad \forall v \in R, \\
((q, v'), \varepsilon, p, f, (q', v')) & \quad \forall v' \in R' \text{ and } (q, \varepsilon, p, f, q') \in \delta, \\
((q, a_1 \cdots a_n), \varepsilon, p, f, (q', a_2 \cdots a_n)) & \quad \forall a_1 \cdots a_n \in R' \text{ and } (q, a_1, p, f, q') \in \delta, \\
((q, \varepsilon), \sigma, p, f, (q', \varepsilon)) & \quad \forall (q, \sigma, p, f, q') \in \delta.
\end{align*}
$$

The automaton will have start state $S$ and final state $(q_f, \varepsilon)$. \hfill \square

We stress once more that everything boils down to the fact that given an automaton $\mathcal{M}$ and a finite number of words $w_i \in \Sigma^*$, it is possible to insert a routine in the automaton that will mimic the behaviour of $\mathcal{M}$ when a word $w_i$ is read, that is, to "insert" $w_i$ in the processed string of letters. The way it is done, is by adding the various suffixes of the $w_i$ as a "second variable" to the states.

If $H$ is a normal subgroup $G$, then the word problem in $G/H$ is exactly the set of words representing elements of $H$. Thus we immediately get the following corollary.

**Corollary 5.2.** If $G$ is a groups with multiple context free word problem and $H$ is a finite normal subgroup of $G$, then $G/H$ has a multiple context free word problem.

We recalled the following result from [LS01].
Theorem 5.3 ([LS01] p.187, Theorem 2.6). Let $G = G_1 *_H G_2$ be an amalgamated product and let $c_1, \ldots, c_n$ be a sequence of elements of $G$ such that:

1. $n \geq 2$.
2. Each $c_i$ is in one of the factors $G_1$ or $G_2$.
3. The words $c_i, c_{i+1}$ come form different factors.
4. No $c_i$ is in $H$.

Then the product $c_1 \cdots c_n$ is non trivial in $G$.

With Proposition 6.1, we can prove our main theorem, as previously stated the idea is similar to Theorem 4.4 with a few extra details.

Theorem 5.4. Let $G_1$ and $G_2$ be groups whose word problem with multiple context free. Let $H_i$ be a finite subgroup of $G_i$, such that $H_1 \cong H_2 \cong H$. Then $G = G_1 *_H G_2$ has a multiple context free word problem.

Proof. Let $W_i$ be the word problem in $G_i$. Let $M_i$ be an automaton accepting the language $W_i$. Let $w_i^h$ be a word in $G_i$ representing the element $h \in H$. Let $F_i = \{w_i^h : h \in H\}$ with a bijection $\phi: F_1 \to F_2$ such that $\phi(w_1^h) = w_2^h$, let $\psi = \phi^{-1}$. Let $F_i'$ be the set of suffixes of words in $F_i$. Let $M_i'$ be the automaton recognising words in $H_i$ from Proposition 6.1 with states $Q_i \times F_i' \cup \{S_i\}$.

Let $W$ be the word problem in $G$. We build an automaton similar to Theorem 4.4 accepting the language $W$.

The idea is the following: suppose that the word $w = a_1 \cdots a_m$ is read. If all $a_i$ are contained in only one between $\Sigma_1$ and $\Sigma_2$, then the automaton will proceed as in the case of the original groups. So suppose this doesn’t happen. We can subdivide the word $w$ into (maximal) subwords that contain only elements of $\Sigma_1$ or $\Sigma_2$. This will give a sequence $c_1, \ldots, c_n$ of elements of $G$. Theorem 5.3 gives that $w =_G c_1 \cdots c_n$ represents the trivial element only if there is an $i$ such that $c_i$ represents an element of $H$. Let $u$ be the subword of $w$ associated to $c_i$.

Without loss of generality, we may assume that $u \in \Sigma_i$. By non-determinism, the automaton will guess the correct $i$ and the element $c_i \in H$. Then, using a procedure similar to Proposition 5.1, it will check if $u$ really represents $c_i$ and, if this is the case, mimic the run that obtained substituting $u$ with a word $v \in \Sigma_2$ representing $c_i$ in $G_2$. Note that for this last step it is crucial that $H$ is finite.

It is clear that the word $w$ will be accepted if and only if the automaton will accept the word obtained by $w$ substituting $u$ with $v$. By induction on the length of the sequence $c_1, \ldots, c_n$, we get the result.

More formally: the states of $M$ are $Q_1 \times F_1' \cup Q_2 \times F_2' \cup \{S_1, S_2, S, F\}$. The storage will be tree stacks over the alphabet $\Omega_1 \cup \Omega_2 \cup (Q_1 \times F_1) \cup (Q_2 \times F_2) \cup \{\square_1, \square_2\}$.

The transitions will consist the following:

1. $\{(S, \varepsilon, C, \text{push}_1(\square_1), (q_1^1, \varepsilon)), (S, \varepsilon, C, \text{push}_2(\square_2), (q_2^1, \varepsilon))\} \cup \{(S, \varepsilon, \text{equals}(y), \text{Id}, F)\}$
If the automaton is in a state \((q_i, \varepsilon)\), the rules are of the form \( (q_i, \varepsilon, \text{equals}(\square_1), \text{down}, S) \) or \( (q_i, \varepsilon, \text{equals}(\square_2), \text{down}, S) \).

1. \( \{(q_i, \varepsilon, \text{equals}(\square_1), \text{down}, S) : (q_i, \varepsilon, \text{equals}(\square_2), \text{down}, S) \} \)

2. \( \{(q_i, \varepsilon, \text{equals}(\square_1), \text{down}, S), (q_i, \varepsilon, \text{equals}(\square_2), \text{down}, S) \} \)

3. \( \{(q_i, \varepsilon, \text{notequals}((Q_2 \times F_2), \text{push}_1, ((q, w), (q_i, \phi(w)^{-1})) : q \in Q_1, w \in F_1, i \in \{-1, \ldots, -n\} \} \)

4. \( \{(q_i, \varepsilon, \text{notequals}((Q_1 \times F_1), \text{push}_i, ((q, w), (q_i, \psi(w)^{-1})) : q \in Q_2, w \in F_2, i \in \{-1, \ldots, -n\} \} \)

5. \( \{(q_i, \varepsilon, \text{equals}((q', w), \text{down}, (q', w)) : q' \in Q_2, w \in F_2 \} \)

6. \( \{(q_i, \varepsilon, \text{equals}((q', w), \text{down}, (q', w)) : q' \in Q_1, w \in F_1 \} \)

7. \( \{(q_i, \varepsilon, \text{equals}(\gamma), \text{Id}, (Q_1, v) \) \}

Before explaining in detail the rules, there is one key and central observation. If the automaton is in a state \((q, w)\) with \( w = a_1 \cdots a_n \neq \varepsilon \), then the only possible rules are of the form \( ((q, a_1 \cdots a_n), \varepsilon, p, f, (q', a_2 \cdots a_n)) \), where \((q, a_1, p, f, q')\) was a rule of \( M_i \) or \((q, a_1 \cdots a_n), \varepsilon, p, f, (q', a_1 \cdots a_n)\), where \((q, \varepsilon, p, f, q')\) was a rule of \( M_i \). That is, if there is a non empty word \( w \) at the second variable, the only possible rule that can be applied is one mimicking the behaviour of one of the original automata if the first letter of \( w \) was read. That is, the priority is always to deplete the second variable of the states.

The elements of the group (1) consist of the very final instruction and the two instructions that starts processing letters in one of the two alphabets \( \Sigma_i \).

The elements of the group (2) consist of the final move for a sub-run representing a trivial (sub)word. Indeed, they are triggered when the automaton is in state \( q_i \).

The elements of the groups (3) and (4) consists of the same type of rules, with the roles of \( G_1 \) and \( G_2 \) interchanged. The rules describe the following instruction (say for the group (3)): "At any moment where the stack pointer is not pointing an element of \( Q_2 \times F_2 \), and your state has empty second variable, you can guess that a sub-word that represents \( \phi(w) \) is starting, for some \( w \in H \). Then, you start a new branch and add \( \phi(w)^{-1} \) at the second variable". If the guess was correct, then eventually the automaton will return to the root of the new branch with state \((q_2, \varepsilon)\). Thus, it successfully processed a sub-word that represented \( \phi(w) \). In this case, the rules of group (5) apply. Indeed, remember that, at the beginning of the process, we pushed \((q, w)\) in the stack, to remember the state at which the automaton was (as in Theorem 4.4) and the word we were checking. Then, we put \( w \) in the second variable. What happened, is that we effectively substituted the sub-word representing \( \phi(w) \) with \( w \).

We will now give a precise proof of the Theorem. This automaton works similarly to the automaton in Theorem 4.4.

Let \( \Lambda \) be an accepted run for the automaton. Let \( T_f \) be the final tree-stack for this run. We colour the non-root vertices of \( T_f \) red and blue as in the proof of Theorem 4.4.
There is a subtree $T_c \subset T_f$ of a single colour whose complement is connected. Assume that $T_c$ is a tree with labels from $\Omega_1$. For each instruction there are two possible pointers, these can be viewed as vertices of $T_f$. Let $\Theta$ be the subset of the instruction in $\Lambda$ such that both pointers are in $T_c$. It can be seen as in the proof of Theorem 4.4 that all these instructions are consecutive. Since $\Theta$ starts and ends at the root, the word read while performing the instructions in $\Theta$ represents an element $v \in F_1$.

The run $\Lambda$ decomposes as a concatenation $\Lambda_1 \theta_1 \Theta \theta_2 \Xi \Lambda_2$, where $\theta_1$ and $\theta_2$ correspond to entering and leaving the tree $T_c$ and $\Xi$ is the run from $(q, \phi(v))$ to the first state $(q', \varepsilon)$.

Since the tree $T_c$ cannot be reentered we see that $\Lambda$ is a valid run if and only if there is a valid run of the form $\Lambda_1 \theta'_2 \Theta \Lambda_2$, where $\theta'$ is the same run as $\Xi$ running through the states $(q, \varepsilon)$ instead of $(q, w)$, one could see this as a run in $M_2$ corresponding to $\Xi$.

The original decomposition $\Lambda_1 \theta_1 \Theta \theta_2 \Xi \Lambda_2$ corresponds to a decomposition of $w$ as $u_1 v u_2$. The word corresponding to the run $\Lambda_1 \theta'_2 \Theta \Lambda_2$ is $u_1 \phi(v) u_2$.

It should be noted that the final tree for the run $\Lambda_1 \theta'_2 \Theta \Lambda_2$ will have one fewer red subtree.

For the base case note that if $T_c = T_f$, then we have a word in $W_1 \cup W_2$. Thus by induction on the number of maximal one-coloured subtrees, $L(M)$ is a subset of $W$.

We must now prove that this automaton accepts all words in $W$. We will use the free product length of a word once again. Let $w = w_1 \ldots w_k$ be a word of free product length $k$. If this word represents the trivial word, then there is a subword $w_j$ which represents and element of $H$. Let $u$ be the corresponding element of $F_1$. We can assume this word is in $\Sigma_1^*$. Let $v$ be an element of $F_2$ representing the same element as $w_j$.

The automaton will leave the automaton $M_2'$ from the state $(q, \varepsilon)$ to the automaton $M_1'$ starting at the state $(q_1', u)$. When the word $w_j$ is read the automaton will return to $M_2'$ at the state $(q, v)$. The automaton will then make a run in $M_2$ for the word $v$. Thus $w$ is in $L(M)$ if and only if $w' = w_1 \ldots w_{j-1} v w_{j+1} \ldots w_k$ is in $L(M)$. Since $w'$ has shorter free product length and it is clear that words of free product length 1 are in $L(M)$, we are done by induction.

\[ \square \]

References

[Cho62] Noam Chomsky. Context-free grammars and pushdown storage. 1962.

[Den16] Tobias Denkinger. An Automata Characterisation for Multiple Context-Free Languages. In Developments in Language Theory, pages 138–150. Springer, Berlin, Heidelberg, July 2016.

[Dun85] M. J. Dunwoody. The accessibility of finitely presented groups. Inventiones mathematicae, 81:449–458, 1985.
[GKS17] R. H. Gilman, R. P. Kropholler, and S. Schleimer. Groups Whose Word Problems are Not Semilinear. In Preparation, 2017.

[Ho17] Meng-Che "Turbo" Ho. The word problem of \( Z^n \) is a multiple context-free language. \textit{arXiv:1702.02926}, February 2017.

[HU69] John E. Hopcroft and Jeffrey D. Ullman. \textit{Formal Languages and Their Relation to Automata}. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1969.

[LS01] Roger C. Lyndon and Paul E. Schupp. \textit{Combinatorial group theory}. Classics in Mathematics. Springer-Verlag, Berlin, 2001. Reprint of the 1977 edition.

[MS81] David E. Muller and Paul E. Schupp. Context-free languages, groups, the theory of ends, second-order logic, tiling problems, cellular automata, and vector addition systems. \textit{Bull. Amer. Math. Soc. (N.S.)}, 4(3):331–334, May 1981.

[Pol84] C. Pollard. Generalized Phrase Structure Grammars, Head Grammars, and Natural Language, 1984.

[Sal15] Sylvain Salvati. MIX is a 2-MCFL and the word problem in \( Z^2 \) is captured by the IO and the OI hierarchies. \textit{J. Comput. Syst. Sci.}, 81(7):1252–1277, November 2015.

[SMFK91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. \textit{Theor. Comput. Sci.}, 88(2):191–229, October 1991.

[Sta71] John R. Stallings. \textit{Group Theory and Three-dimensional Manifolds}. Yale University Press, New Haven, January 1971.