doi:10.1093/comjnl/bxx128

Watson–Crick Context-Free Grammars: Grammar Simplifications and a Parsing Algorithm

NURUL LIYANA MOHAMAD ZULKUFLI*, SHERZOD TURAEV, MOHD IZZUDDIN MOHD TAMRIN AND AZEDDINE MESSIKH

Department of Computer Science, Kulliyyah of Information and Communication Technology, International Islamic University Malaysia, Kuala Lumpur, Malaysia
Corresponding author: liyanazulkufli@iium.edu.my

A Watson–Crick (WK) context-free grammar, a context-free grammar with productions whose right-hand sides contain nonterminals and double-stranded terminal strings, generates complete double-stranded strings under Watson–Crick complementarity. In this paper, we investigate the simplification processes of Watson–Crick context-free grammars, which lead to defining Chomsky-like normal form for Watson–Crick context-free grammars. The main result of the paper is a modified CYK (Cocke–Younger–Kasami) algorithm for Watson–Crick context-free grammars in WK-Chomsky normal form, allowing to parse double-stranded strings in $O(n^6)$ time.

Keywords: DNA Computing; formal languages; parsing algorithms; Watson–Crick grammars; Watson–Crick automata; grammar simplifications; context-free grammars

Received 13 October 2016; revised 8 December 2017; editorial decision 8 December 2017
Handling editor: Raphael Clifford

1. INTRODUCTION

DNA computing appears as a challenge to design new types of computing paradigms, which differ from classical counterparts in fundamental way, to solve wide spectrum of computationally intractable problems. DNA molecules are double-stranded structures composed of four nucleotides A (adenine), C (cytosine), G (guanine) and T (thymine), paired to A–T and C–G according to the so-called Watson–Crick complementary. DNA computing contains various formal language theoretical approaches of the recombinant behavior of DNA sequences under the effect of enzymatic activities. Different DNA operations motivate to introduce different formal language tools, such as recognition devices (automata) and generative devices (grammars), and to investigate structures and properties of biomolecular sequences.

Watson–Crick (WK) automata, one of the early DNA computing models, are introduced as an extension of finite automata with the addition of two reading heads on double-stranded sequences [1]. The symbols on corresponding positions from the two strands of the input are related by a complementarity relation, similar with the WK complementarity of DNA nucleotides. The two strands of the input are separately scanned from left to right by read-only heads controlled by a common state. Various restrictions and extensions can be made onto the basic model of WK automata to achieve more computational power, such as changing the way the reading head works, and providing the automata with special system like output and weight.

There are a number of variants of WK automata such as initial stateless WK finite automata, WK automata with a WK memory, WK transducers [2] and weighted WK automata [3] introduced. Paper [4] proposes parallel communicating WK automata systems, which exploit the massive parallelism trait of DNA molecules. The survey [5] covers a detailed information on WK automata.

The computational relations among WK automata and context-free grammars are studied in [6, 7]. A pioneering work [8], which uses this fundamental feature, proposes an analytic counterpart of WK finite automata called (static) WK regular grammars, which are regular grammars with double-stranded terminal substrings on the right-hand side of productions, and generate the languages of complete double-stranded strings. Papers [9–11] introduce dynamic variants of WK (regular, linear and context-free) grammars, study their generative capacities and closure properties.

1.1. Our contribution

In this work, we investigate the structural properties and simplification issues of WK context-free grammars. First, we discuss the derivation processes by WK context-free grammars, where derivations are not the same as those by Chomsky context-free grammars: two adjacent nonterminals may generate different complete double-stranded substrings 'together' depending on the positions from where each nonterminal starts building up the complete substring. As a result, various possible shapes of initial double-stranded substrings between nonterminals are discovered. Surprisingly, this dependency in derivations does not affect the application order of productions: we show that leftmost and rightmost derivations result in the same complete double-stranded string.

In order to facilitate the analysis of WK context-free grammars and languages, we define a 'Chomsky-like' normal form, called *WK-Chomsky normal form*, for a WK grammar by imposing some restrictions on the forms of productions of the grammar. Similar to usual context-free case, we consider three main transformations—removal of erasing productions, removal of chain productions and removal of useless nonterminals and productions—that convert an arbitrary WK context-free grammar into the grammar in WK-Chomsky normal form. Further, we develop a modified CYK algorithm for WK context-free grammars using WK-Chomsky normal form and show that the time complexity of this algorithm is $O(n^6)$.

The paper is organized as follows: in Section 2, we recall the basic notions and notations from the theory of formal languages that are used throughout the paper. In Section 3, we define WK context-free grammars. In Section 4, we show that an arbitrary WK context-free grammar can be converted into the grammar in WK-Chomsky normal form through transformation stages. In Section 5, we develop WK-CYK algorithm for WK context-free grammars and discuss its complexity issues. We conclude our paper with a summary and open problems in Section 6.

2. PRELIMINARIES

We assume that a reader is familiar with basic notions and notations of formal languages, grammars and DNA computing. For more details, the reader is referred to [2, 5, 12, 13].

In the paper, we use the following general notations: the inclusion is denoted by $\subseteq$ and the strict (proper) inclusion is denoted by $\subset$. The symbol $\varnothing$ denotes the empty set. The power set of a set X is denoted by 2^X, while the cardinality of a set X is denoted by $|X|$. The notation $[x, y]$ denotes a closed integer interval.

2.1. Grammars

Let Σ be an *alphabet* which is a finite non-empty set of symbols. A *string* over the alphabet Σ is a finite sequence of symbols from Σ. The *empty* string is denoted by λ. The set of all strings over the alphabet Σ is denoted by Σ^*. The set of non-empty strings over Σ is denoted by Σ^+, i.e. $\Sigma^+ = \Sigma^* - \{\lambda\}$. A subset of Σ^* is called a *language*. The *length* of a string $w \in \Sigma$, denoted by $|w|$. The *shuffle* of two strings $u, v \in \Sigma^*$, denoted by $u \Delta v$, is the set of all strings w of the form $w = u_1 v_1 u_2 v_2 \ldots u_k v_k$ with $u_i, v_i \in \Sigma^*$ for all $i \in [1, k]$, and $u_1 u_2 \ldots u_k = u$ and $v_1 v_2 \ldots v_k = v$.

A *context-free grammar* is a quadruple $G = (N, T, S, P)$, where N is an alphabet of *nonterminals*, T is an alphabet of *terminals*, with $N \cap T = \varnothing$, $S \in N$ is the *start symbol*, and $P \subseteq N \times (N \cup T)^*$ is a finite set of *productions*. We write $A \to \beta$ indicating the rewriting process of the strings based on the production $(A, \beta) \in P$. For a production $A \to \beta$, A is called its *left-hand side and* β *its right-hand side*. A production whose right-hand side is the empty string is called an λ-*production* (erasing production). To abbreviate productions $A \to \beta_1, A \to \beta_2, \ldots, A \to \beta_k \in P$, $k \geq 2$, with the same left-hand side, we use the shorthand $A \to \beta_1|\beta_2| \cdots |\beta_n$ where the vertical bar stands for 'or'.

$x \in (N \cup T)^*$ *directly derives* $y \in (N \cup T)^*$, written as $x \Rightarrow y$, if and only if $x = x_1 A x_2$ and $y = x_1 \beta x_2$ for some productions $A \to \beta \in P$ and $x_1, x_2 \in (N \cup T)^*$. This is also called a *derivation step*.

A string $u \in (N \cup T)^*$ *derives* a string $v \in (N \cup T)^*$, written as $u \Rightarrow^* v$, if either

- $u = v$ or
- there are strings $u_0, u_1, \ldots, u_n$ in $(N \cup T)^*$, called *sentential forms*, such that $u_0 = u$, $u_n = v$, and u_{i-1} directly derives u_i for all $i \in [1, n]$, i.e.,

$$u = u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n = v. \tag{1}$$

The sequence (1) is called the *derivation* of v from u. If $v \in T^*$, v is called a *terminal string*. A derivation is called *leftmost* (*rightmost*) if at every derivation step, the leftmost (rightmost) nonterminal of the sentential form is rewritten.

The language generated by the grammar G, denoted by $L(G)$, is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$.

A *derivation tree* or a *parse tree* is an ordered tree where the interior nodes of the tree are the left-hand sides of productions of a grammar, and all children of the nodes are their corresponding right-hand sides. The start symbol in the grammar is the root of the tree, while the terminals are the leaves.

Formally, a derivation tree can be defined as follows. Let $G = (N, T, S, P)$ be a context-free grammar and $S \Rightarrow^* w$ be a derivation in G. A *derivation tree* of $S \Rightarrow^* w$ is a directed, ordered tree whose nodes are labeled with symbols of $N \cup T \cup \{\lambda\}$ in such a way that

(1) the interior nodes are labeled with nonterminals of N,
(2) the root is labeled with the start symbol S,

(3) if $\beta_1, \beta_2, \ldots, \beta_n$ are labels of all children of a node labeled with nonterminal A, ordered from left to right, then $A \to \beta_1\beta_2\ldots\beta_n$ is a production of P.

The *yield* of a derivation tree is the string over $N \cup T$ constructed from the labels of the leaves by reading from left to right.

2.2. Watson–Crick (WK) grammars

Let $\rho \subseteq V \times V$ be a symmetric relation on an alphabet V. We denote by $V^* \times V^*$ the set of all pairs of strings over V. We write the elements $(x, y) \in V^* \times V^*$ in the form $\langle x/y \rangle$. The *length* of $\langle x/y \rangle$ is defined by $|x| + |y|$. We also use notations $[V/V]$ and $\langle V^*/V^* \rangle$ instead of $V \times V$ and $V^* \times V^*$, respectively. Strings of the form $\langle u/\lambda \rangle$ and $\langle \lambda/v \rangle$ are called *semi-empty* strings. For two strings $\langle u_1/v_1 \rangle$ and $\langle u_2/v_2 \rangle$ in $\langle V^*/V^* \rangle$, their concatenation $\langle u_1/v_1 \rangle \cdot \langle u_2/v_2 \rangle$ is defined as $\langle u_1u_2/v_1v_2 \rangle$.

Let $[V/V]_\rho = \{[a/b] : a, b \in V \text{ and } (a, b) \in \rho\}$. The set $WK_\rho(V) = [V/V]_\rho^*$, the set of all (complete) double-stranded strings (molecules), is called the *WK domain* associated to the alphabet V and the complementary relation ρ. The subscript ρ can be omitted if there is no danger of confusion.

A string $[a_1/b_1][a_2/b_2]\ldots[a_n/b_n] \in WK_\rho(V)$ is, for short, written as $[u/v]$ where $u = a_1a_2\ldots a_n$ and $v = b_1b_2\ldots b_n$. Then u, v are called *upper* and *lower strands*, respectively.

One can notice that $[u/v] = \langle u/v \rangle$ iff the strings u and v have the same length and the corresponding symbols in u and v are complementary in the sense of the relation ρ.

On the other hand, for a string $\langle u/v \rangle \in \langle V^*/V^* \rangle$, the positions of the symbols of strings u and v are not fixed with respect to each other (there is no relation between the symbols of u and v), $\langle u/v \rangle$ can represent different incomplete strings. For instance, string $\langle aa/a \rangle$ may represent $[a/a]\langle a/\lambda \rangle$, $\langle a/\lambda \rangle [a/a]$ or even $\langle a/\lambda \rangle \langle a/\lambda \rangle \langle \lambda/a \rangle$, $\langle \lambda/a \rangle \langle a/\lambda \rangle \langle a/\lambda \rangle$, etc.

For strings $\langle u_1/v_1 \rangle$ and $\langle u_2/v_2 \rangle$ in $\langle V^*/V^* \rangle$, we define their *shuffle* as the set $\langle u_1/v_1 \rangle ш \langle u_2/v_2 \rangle$ of all strings

$$\langle u_{1(1)}/v_{1(1)} \rangle \langle u_{2(1)}/v_{2(1)} \rangle \cdots \langle u_{1(k)}/v_{1(k)} \rangle \langle u_{2(k)}/v_{2(k)} \rangle \qquad (2)$$

where $u_{i(j)}, v_{i(j)} \in V^*$ for all $i \in [1, 2]$, $j \in [1, k]$, $k \geq 1$ and $u_i = u_{i(1)}\ldots u_{i(k)}$ and $v_i = v_{i(1)}\ldots v_{i(k)}$ for $i \in [1, 2]$.

We also define the *symbolic-shuffle* of strings $\langle u_1/v_1 \rangle$ and $\langle u_2/v_2 \rangle$, denoted by $\langle u_1/v_1 \rangle ш_\Sigma \langle u_2/v_2 \rangle$, as the subset of $\langle u_1/v_1 \rangle ш \langle u_2/v_2 \rangle$ such that for each shuffled string (2) belonging to this set, $|u_{i(j)}v_{i(j)}| = 1$ for all $i \in [1, 2]$ and $j \in [1, k]$.

For any strings $\langle u_1/\lambda \rangle$ and $\langle \lambda/v_2 \rangle$ or $\langle \lambda/v_1 \rangle$ and $\langle u_2/\lambda \rangle$, their shuffle consists of only one string, i.e. $\langle u_1/v_2 \rangle$ or $\langle u_2/v_1 \rangle$, respectively. Thus, any string $\langle u/v \rangle$ can be represented as a symbolically shuffled string from $\langle u/\lambda \rangle ш_\Sigma \langle \lambda/v \rangle$.

Further, we recall some definitions related to WK grammars defined by [14, 8].

Definition 2.1. *A Watson–Crick (WK for short) context-free grammar is a 5-tuple $G = (N, T, \rho, S, P)$, where N, T, S are defined as for a context-free grammar, ρ is a symmetric relation on T, and $P \subseteq N \times (N \cup \langle T^*/T^* \rangle)^*$ is non-empty finite set of productions. If*

$$P \subseteq N \times (\langle T^*/T^* \rangle N \cup \langle T^*/T^* \rangle),$$

then the grammar G is called regular, and if

$$P \subseteq N \times (\langle T^*/T^* \rangle N \langle T^*/T^* \rangle \cup \langle T^*/T^* \rangle),$$

then it is called linear.

We write $\langle T^*/T^* \rangle$, not $\langle T/T \rangle$, because in a double-stranded string, there could be a case where a production generates a double-stranded string with the empty string in either upper or lower strand.

Definition 2.2. *Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar. We say that $x \in (N \cup \langle T^*/T^* \rangle)^*$ directly derives $y \in (N \cup \langle T^*/T^* \rangle)^*$, denoted by $x \Rightarrow y$, if and only if*

$$x = \langle u_1/v_1 \rangle A \langle u_2/v_2 \rangle \quad \text{and}$$
$$y = \langle u_1/v_1 \rangle \beta \langle u_2/v_2 \rangle$$

where $A \in N$, $u_i, v_i \in (N \cup \langle T^*/T^* \rangle)^*$, $i \in [1, 2]$ and $A \to \beta \in P$.

If $\beta = \langle x_1/y_1 \rangle \gamma \langle x_2/y_2 \rangle$, where $x_i, y_i \in T^*$, $i \in [1, 2]$ and $\gamma \in (N \cup \langle T^*/T^* \rangle)^*$, then $y = \langle u_1x_1/v_1y_1 \rangle \gamma \langle x_2u_2/y_2v_2 \rangle$.

The transitive and reflexive closure of the relation $\Rightarrow$ is denoted by $\Rightarrow^*$.

Similar to context-free grammar, the definitions follow:

Definition 2.3. *A derivation in a WK context-free grammar is called leftmost (rightmost) derivation, denoted as $\Rightarrow_l$ ($\Rightarrow_r$), if at each step of the derivation, the leftmost (rightmost) nonterminal symbol is rewritten.*

Definition 2.4. *The language generated by a WK context-free grammar G is called as WK context-free language and defined as*

$$L(G) = \{u \colon [u/v] \in WK_\rho(T) \text{ and } S \Rightarrow^* [u/v]\}.$$

Remark 1. For simplicity, in this paper, we use the languages in the examples in the form of $L(G) = \{u\colon [u/u]$ instead of $L(G) = \{u\colon [u/v]$, and the relation $(a, a) \in \rho$ instead of $(a, b) \in \rho$.

3. DERIVATION TREES

In context-free grammars, a derivation, a transformation of nonterminals into terminal strings, can be represented graphically by derivation (parse) tree. In a similar manner, we can define the concept of derivation tree for WK context-free grammars. WK context-free grammars use nonterminals but double-stranded terminals, we need to clarify which double-stranded symbols are used as labels of tree nodes. We show that any WK context-free grammar can be transformed into an equivalent WK context-free grammar in which every terminal substring on the right-hand side of its productions can be decomposed into double-stranded symbols of the total length one.

DEFINITION 3.1. *A WK context-free grammar $G = (N, T, \rho, S, P)$ is said to be in terminal normal form if and only if each production in P has one of the following forms*:

$$A \to x_1 B_1 x_2 B_2 x_3 \ldots x_n B_n x_{n+1}, \quad n \geq 1,$$

or

$$A \to x$$

where $A, B_i \in N,\ 1 \leq i \leq n,$ *and* $x, x_i \in \langle T/\lambda\rangle^* \langle \lambda/T\rangle^*$, $1 \leq i \leq n+1$.

LEMMA 3.1. *For every WK context-free grammar G with $\lambda \notin L(G)$, there exists an equivalent WK context-free grammar G' in terminal normal form.*

Proof. Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar. For each production $A \to \beta \in P$, since $\beta \in (N \cup \langle T^*/T^*\rangle)^*$, it has one of the following forms:

$$\beta = \langle x_1/y_1\rangle B_1 \langle x_2/y_2\rangle B_2 \langle x_3/y_3\rangle \cdots \langle x_n/y_n\rangle B_n \langle x_{n+1}/y_{n+1}\rangle,$$

or

$$\beta = \langle x/y\rangle$$

where $B_i \in N,\ 1 \leq i \leq n$ and $x, x_i, y, y_i \in T^*,\ 1 \leq i \leq n+1$.

By the definition of symbolic-shuffle mentioned in Section 2.2, in general, if we have a string $\langle x/y\rangle \in \langle T^*/T^*\rangle$ such that $x = u_1 u_2 \ldots u_k$ and $y = y_1 y_2 \ldots y_l$, then for $k + l > 1$, $\langle u_1 u_2 \ldots u_k / v_1 v_2 \ldots v_l\rangle$ can be replaced with any symbolically shuffled string from

$$\langle u_1 u_2 \ldots u_k/\lambda\rangle \sqcup\!\sqcup_T \langle \lambda/v_1 v_2 \ldots v_l\rangle.$$

Thus, we can choose

$$\langle u_1/\lambda\rangle\langle u_2/\lambda\rangle \cdots \langle u_k/\lambda\rangle\langle \lambda/v_1\rangle\langle \lambda/v_2\rangle \cdots \langle \lambda/v_l\rangle,$$

for $u_s, v_r \in T, 1 \leq s \leq k$ *and* $1 \leq r \leq l$. □

DEFINITION 3.2. *Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar and $S \Rightarrow^* \langle u/v\rangle$ be a derivation in G. A derivation tree of $S \Rightarrow^* \langle u/v\rangle$ is a directed, ordered tree whose nodes are labeled with symbols of $N \cup \langle T/\lambda\rangle \cup \langle \lambda/T\rangle \cup \langle \lambda/\lambda\rangle$ in such a way that*

(1) *the root is labeled with the start symbol S,*
(2) *the interior nodes are labeled with nonterminals of N,*
(3) *if $x_1, x_2, \ldots, x_n$ are labels of the children of a node labeled with nonterminal A, ordered from left to right, then $A \to x_1 x_2 \ldots x_n$ is a production of P.*

EXAMPLE 1. Let $G = (\{S, A, B, C\}, \{a, b, c\}, \rho, S, P)$ be a WK context-free grammar in terminal normal form, where $\rho = \{(a, a), (b, b), (c, c)\}$ and P contains the following productions:

$$\begin{aligned}
&S \to ABC,\\
&A \to \langle a/\lambda\rangle A | \langle \lambda/a\rangle A | \langle \lambda/\lambda\rangle,\\
&B \to \langle b/\lambda\rangle B | \langle \lambda/b\rangle B | \langle \lambda/\lambda\rangle,\\
&C \to \langle c/\lambda\rangle C | \langle \lambda/c\rangle C | \langle \lambda/\lambda\rangle.
\end{aligned}$$

Then for instance, the leftmost derivation for $[abc/abc]$:

$$\begin{aligned}
S &\Rightarrow ABC \Rightarrow \langle a/\lambda\rangle ABC \Rightarrow \langle a/a\rangle ABC \Rightarrow \langle a/a\rangle BC\\
&\Rightarrow \langle ab/a\rangle BC \Rightarrow \langle ab/ab\rangle BC \Rightarrow \langle ab/ab\rangle C\\
&\Rightarrow \langle abc/ab\rangle C \Rightarrow \langle abc/abc\rangle C \Rightarrow [abc/abc],
\end{aligned}$$

and the rightmost derivation for $[abc/abc]$:

$$\begin{aligned}
S &\Rightarrow ABC \Rightarrow AB\langle c/\lambda\rangle C \Rightarrow AB\langle c/c\rangle C \Rightarrow AB\langle c/c\rangle\\
&\Rightarrow A\langle b/\lambda\rangle B\langle c/c\rangle \Rightarrow A\langle b/b\rangle B\langle c/c\rangle \Rightarrow A\langle bc/bc\rangle\\
&\Rightarrow \langle a/\lambda\rangle A\langle bc/bc\rangle \Rightarrow \langle a/a\rangle A\langle bc/bc\rangle \Rightarrow [abc/abc].
\end{aligned}$$

Figure 1 shows the derivation tree for $[abc/abc]$ regardless of its leftmost and rightmost derivations.

It is known that in a context-free grammar G, there exist the leftmost and rightmost derivation for a string $w \in L(G)$. The end result derived by the leftmost derivation is the same as the end result derived by the rightmost derivation.

REMARK 2. The 'end result' means the final string (or the target sub-sentential form at some point) from the derivation.

The next lemma shows that this property also holds for WK context-free grammars.

LEMMA 3.2. *Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar. For every string $[w/w] \in [T/T]^*$ generated by G, there exist a leftmost derivation $S \Rightarrow_l^* [w/w]$ and a rightmost derivation $S \Rightarrow_r^* [w/w]$.*

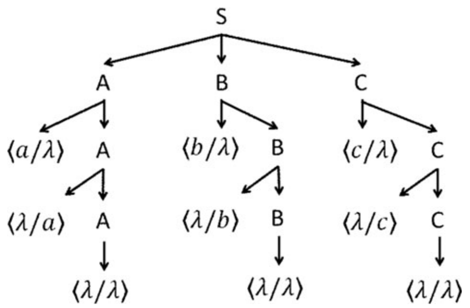

FIGURE 1. A derivation tree for the double-stranded string $[abc/abc]$ based on the WK context-free grammar in Example 1. The derivation tree is the same regardless of the leftmost and rightmost derivation.

Proof. Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar. If G is a WK linear grammar, the argument is trivial.

Suppose that a string $[w/w] \in [T/T]^*$ is obtained by the following derivation:

$$S \Rightarrow^* xABy \Rightarrow^* xA' \langle u_1/v_1\rangle \langle u_2/v_2\rangle B'y \tag{3}$$

$$= xA' \langle u_1u_2/v_1v_2\rangle B'y \Rightarrow^* [w/w] \tag{4}$$

where $\langle u_1/v_1\rangle, \langle u_2/v_2\rangle \in \langle T^*/T^*\rangle$, $A, B, A', B' \in N$ and $x, y \in (N \cup \langle T^*/T^*\rangle)^*$. Then, we distinguish the following two cases:

Case 1. In the substring of $[w/w]$ generated from $A' \langle u_1u_2/v_1v_2\rangle B'$, the symbols of u_1 are not related to the symbols of v_2 or to the symbols of the substring generated by B, and the symbols of u_2 are not related to the symbols of v_1 or to the symbols of the substring generated by A, i.e. the derivations from A and B are completely independent (see Figure 2(a)), which is similar to a usual context-free derivation. Thus, derivation (3) can be continued from A.

Case 2. In the substring of $[w/w]$ generated from $A' \langle u_1u_2/v_1v_2\rangle B'$, either some symbols of u_1 are related by ρ to symbols of v_2 or to symbols of the substring generated by B, or some symbols of u_2 are related by ρ to symbols of v_1 or to symbols of the substring generated by A, i.e. the derivations from A and B are dependent (Figure 2(b)).

Let us consider productions $A' \to \beta_1 \langle u_3/v_3\rangle$, $B' \to \langle u_4/v_4\rangle \beta_2 \in P$ where $x, x_i, y, y_i, \beta_1, \beta_2 \in (N \cup \langle T^*/T^*\rangle)^*$ $(i = [1, 4])$, $u_3, u_4, v_3, v_4 \in T^*$ that are used to generate $[w/w]$.

Then, there are two possible derivations from A' and B':

$$\begin{aligned} S &\Rightarrow^* xA' \langle u_1u_2/v_1v_2\rangle B'y \\ &\Rightarrow_l^* x\beta_1 \langle u_3/v_3\rangle \langle u_1u_2/v_1v_2\rangle B'y \\ &\Rightarrow_l^* x\beta_1 \langle u_3/v_3\rangle \langle u_1u_2/v_1v_2\rangle \langle u_4/v_4\rangle \beta_2 y \end{aligned} \tag{5}$$

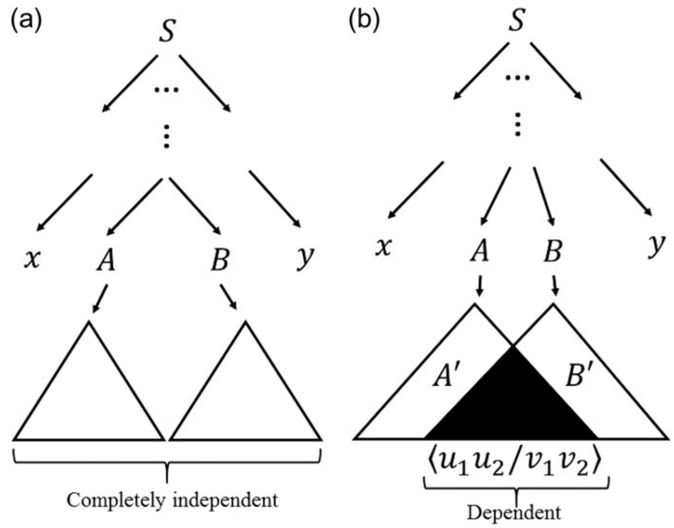

FIGURE 2. Different types of derivations in WK context-free grammars: (**a**) a tree with completely independent derivations from nonterminal symbols A and B; (**b**) the black triangle indicates the dependent part derived both from A and B.

and

$$\begin{aligned} S &\Rightarrow^* xA' \langle u_1u_2/v_1v_2\rangle B'y \\ &\Rightarrow_r^* xA' \langle u_1u_2/v_1v_2\rangle \langle u_4/v_4\rangle \beta_2 y \\ &\Rightarrow_r^* x\beta_1 \langle u_3/v_3\rangle \langle u_1u_2/v_1v_2\rangle \langle u_4/v_4\rangle \beta_2 y \end{aligned} \tag{6}$$

From (5) and (6), the same sub-sentential form $\beta_1 \langle u_3u_1u_2u_4/v_3v_1v_2v_4\rangle \beta_2$ is generated. That is, the order of production applications does not affect the end result of the derivation, i.e. the last line in derivation (5) and the last line in (5) are the same. Again, derivation (3) can be continued from A. □

4. GRAMMAR SIMPLIFICATIONS

Since the right-hand sides of productions in WK context-free grammars are unrestricted, it is difficult to study the properties and relations of grammars and languages. In this section, we consider the context-free grammar transformations (see [12, 15]) for their WK variants that transform an arbitrary WK context-free grammar into a grammar in Chomsky-like normal form, called WK-Chomsky normal form, which is useful to develop parsing algorithms for WK context-free grammars. In the following lemmas, we mostly adapt the proof arguments of the lemmas and theorems on grammar transformations and simplifications given in [12, 15] for WK variants.

LEMMA 4.1. (Substitution Rule). *Let* $G = (N, T, \rho, S, P)$ *be a WK context-free grammar. Let* $A \to uBv \in P$ *where* $u, v \in (N \cup \langle T^*/T^*\rangle)^*$, *and for* $B \neq A$,

$$B \to \beta_1|\beta_2|\cdots|\beta_n \in P$$

where $\beta_i \in (N \cup \langle T^*/T^* \rangle)^*$, $1 \le i \le n$ *and* $n > 0$. *Then, the WK context-free grammar* $G' = (N, T, \rho, S, P')$ *with*

$$P' = P - \{A \to uBv\} \\ \cup \{A \to u\beta_1 v|u\beta_2 v|\cdots|u\beta_n v\}$$

is an equivalent grammar to G, *i.e.* $L(G') = L(G)$.

Proof. The inclusion $L(G') \subseteq L(G)$ is obvious since the application of each production $A \to u\beta_i v$, $1 \le i \le n$, in a derivation in G' can be replaced with the consecutive derivation steps $A \Rightarrow uBv \Rightarrow u\beta_i v$ in G.

Suppose that a terminal string $[w/w]$ is derived in G using a production $A \to uBv$:

$$S \Rightarrow^* x_1Ax_2 \Rightarrow x_1uBvx_2 \Rightarrow^* x_1u'\beta_i v'x_2 \Rightarrow^* [w/w]. \quad (7)$$

By Lemma 3.2, the order of applications of productions in a derivation in a WK context-free grammar are independent, the derivation (7) can be rewritten as

$$S \Rightarrow^* x_1Ax_2 \Rightarrow x_1uBvx_2 \Rightarrow x_1u\beta_i vx_2 \Rightarrow^* [w/w]. \quad (8)$$

Thus, the derivation (8) can be replaced with

$$S \Rightarrow^* x_1Ax_2 \Rightarrow x_1u\beta_i vx_2 \Rightarrow^* [w/w]$$

in G'. □

DEFINITION 4.1. *Any production of WK context-free grammar of the form* $A \to \langle \lambda/\lambda \rangle$ *is called* λ*-production, and any nonterminal symbol* A *with derivation* $A \Rightarrow^* [\lambda/\lambda]$ *is called nullable.*

REMARK 3. Note that $A \to \langle u/\lambda \rangle$, $|u| \ne \lambda$ and $A \to \langle \lambda/v \rangle$, $|v| \ne \lambda$, are not λ-productions.

LEMMA 4.2. (*Removing* λ*-productions*). *Let* $G = (N, T, \rho, S, P)$ *be a WK context-free grammar with* $\lambda \notin L(G)$. *Then there exists an equivalent WK context-free grammar* $G' = (N, T, \rho, S, P')$ *without* λ*-productions.*

Proof. First, we construct set N_{NULL} of all nullable nonterminals of G:

(1) For all $A \to \langle \lambda/\lambda \rangle$ in P, add A to N_{NULL}.
(2) Repeat the following step until no nonterminal is added to N_{NULL}:
For all $B \to A_1A_2\ldots A_k \in P$ where all $A_i \in N_{NULL}$, $1 \le i \le k$, add B to N_{NULL}
(3) Construct P' as follows:
Add each production

$$A \to w_1w_2\ldots w_n$$

where $w_i \in N \cup \langle T^+/\lambda \rangle \cup \langle \lambda/T^+ \rangle \cup \langle T^+/T^+ \rangle$, $1 \le i \le n$, and all productions generated by replacing nullable nonterminals with λ in all possible combinations to P' unless all w_i are nullable.

Then, G' also generates $L(G)$. □

DEFINITION 4.2. *Any production of a WK context-free grammar of the form* $A \to B$, *where* A *and* B *are nonterminals, is called a unit-production.*

Since unit-productions involve only nonterminals, using the same arguments of the proofs of Lemma 4.3.2 and Theorem 4.3.3 from [15] or Theorem 6.4 from [12], one can show that the following lemma also holds.

LEMMA 4.3. (Removing unit-productions). *Let* $G = (N, T, \rho, S, P)$ *be a WK context-free grammar without* λ*-productions. Then, there exists an equivalent WK context-free grammar* $G' = (N', T, \rho, S, P')$ *without unit-productions, where* $N' \subseteq N$.

DEFINITION 4.3. *Let* $G = (N, T, \rho, S, P)$ *be a WK context-free grammar. A nonterminal* $A \in N$ *is called useful if there exists at least one derivation*

$$S \Rightarrow^* xAy \Rightarrow^* [w/w]$$

where $x, y \in (N \cup \langle T^*/T^* \rangle)^*$ *and* $w \in L(G)$. *A nonterminal which is not useful is called useless, and any production involving a useless nonterminal in it is called a useless production.*

LEMMA 4.4. (Removing useless productions). *Let* $G = (N, T, \rho, S, P)$ *be a WK context-free grammar. Then there exists an equivalent WK context-free grammar* $G' = (N', T, \rho, S, P')$ *without useless nonterminals and productions.*

Proof. In order to construct G', first, we should identify useless nonterminals of G unreachable from the start symbol, and second, find the useless nonterminals that do not generate any complete or incomplete terminal string $\langle u/v \rangle \in \langle T^+/T^+ \rangle \cup \langle T^+/\lambda \rangle \cup \langle \lambda/T^+ \rangle$. Since, the first part involves only nonterminals, we can use the same proof arguments of Lemma 4.4.5 from [15].

The difference of the form of a double-stranded strings in both complete and incomplete cases from single-stranded strings makes us consider the second case in some detail.

Suppose that $G_1 = (N_1, T, \rho, S, P_1)$ be a WK context-free grammar without nonterminals unreachable from S. We construct the grammar $G' = (N', T, \rho, S, P')$ from G_1 by removing the nonterminals that cannot derive any complete or incomplete terminal string in the following steps:

(1) Let N' be the empty set.
(2) Repeat until no more nonterminals are put into N': For each $A \in N$, if $A \to \alpha \in P_1$ where $\alpha \in (N' \cup \langle T^*/T^*\rangle)^*$, put A into N'.
(3) Define $P' = \{A \to \alpha \in P_1 | \alpha \in (N' \cup \langle T^*/T^*\rangle)^*\}$. □

However, different from the conntext-free grammars, the processes in removing useless productions cannot guarantee that the string produced by a WK context-free grammar is a complete double-stranded string.

Summarizing the results above, we get the following theorem.

THEOREM 4.1. *Let L be a WK context-free language where $\lambda \notin L$. Then, there exists a WK context-free grammar G without λ-productions, unit-productions, and useless productions such that $L(G) = L$.*

We show an example:

EXAMPLE 2. Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar where P contains the following productions:

$$\begin{array}{ll} S \to SS, & S \to \langle a/a\rangle S\langle b/b\rangle, \\ S \to \langle a/\lambda\rangle S, & S \to \langle a/\lambda\rangle A, \\ A \to \langle b/a\rangle A, & A \to \langle b/a\rangle B, \\ B \to \langle \lambda/b\rangle B, & B \to \langle \lambda/\lambda\rangle, \\ B \to S, & C \to CD. \end{array}$$

Then, the equivalent WK context-free grammar G' with the following productions can be constructed after removing λ-productions, unit-productions, and useless productions:

$$\begin{array}{ll} S \to SS, & S \to \langle a/a\rangle S\langle b/b\rangle, \\ S \to \langle a/\lambda\rangle S, & S \to \langle a/\lambda\rangle A, \\ A \to \langle b/a\rangle A, & A \to \langle b/a\rangle B, \\ A \to \langle b/a\rangle, & \\ B \to \langle \lambda/b\rangle B, & B \to \langle \lambda/b\rangle, \\ B \to SS, & B \to \langle a/a\rangle S\langle b/b\rangle, \\ B \to \langle a/\lambda\rangle S, & B \to \langle a/\lambda\rangle A. \end{array}$$

5. WK-CHOMSKY NORMAL FORM

In this section, we show that Chomsky normal form can also be constructed for WK context-free grammars.

DEFINITION 5.1. *A WK context-free grammar $G = (N, T, \rho, S, P)$ is said to be in WK-Chomsky normal form if all productions are of the form*

- $A \to BC$
- $A \to \langle u/v\rangle$, *or*
- $S \to \langle \lambda/\lambda\rangle$

where $A \in N$, $B, C \in N - \{S\}$ and $\langle u/v\rangle \in \langle T/\lambda\rangle \cup \langle \lambda/T\rangle$.

LEMMA 5.1. *For every WK context-free grammar G, there exists an equivalent WK context-free grammar G' in WK-Chomsky normal form.*

Proof. Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar. Without loss of generality, we assume that G is in terminal normal form without λ-productions (except $S \to \langle \lambda/\lambda\rangle$), unit-productions and useless productions. From G, we construct an equivalent WK context-free grammar $G' = (N', T, \rho, S, P')$ in WK-Chomsky normal form. We set $N_T = \{T_a^u, T_a^d | a \in T\}$ and

$$P_1 = \{T_a^u \to \langle a/\lambda\rangle | a \in T\} \cup \{T_a^d \to \langle \lambda/a\rangle | a \in T\}.$$

We construct the set P_2 of productions from P as follows. Let $A \to \beta \in P$.

(1) If $|\beta| = 0$, then this is only production $S \to \langle \lambda/\lambda\rangle$, and we add this production to P_2.
(2) If $|\beta| = 1$, then $\beta \in \langle T/\lambda\rangle \cup \langle \lambda/T\rangle$, and we also add this production to P_2.
(3) Let $|\beta| \geq 2$ and $\beta = x_1x_2\ldots x_k$ where $x_i \in N \cup \langle T/\lambda\rangle \cup \langle \lambda/T\rangle$. For each $1 \leq i \leq k$, we set

$$X_i = \begin{cases} x_i & \text{if } x_i \in N, \\ T_a^u & \text{if } x_i = \langle a/\lambda\rangle \in \langle T/\lambda\rangle, \\ T_a^d & \text{if } x_i = \langle \lambda/a\rangle \in \langle \lambda/T\rangle, \end{cases}$$

and add the following new productions to P_2:

$$A \to X_1Y_1, \;\; Y_1 \to X_2Y_2, \ldots, Y_{k-2} \to X_{k-1}X_k$$

where $Y_1, Y_2, \ldots, Y_{k-2}$ are new nonterminals.

We define N' as the set of all nonterminals of N and all new nonterminals introduced above, and $P' = P_1 \cup P_2$. Then, it is not difficult to see that $L(G) = L(G')$. □

Example 3 illustrates the transformation of a WK context-free grammar into WK-Chomsky normal form.

EXAMPLE 3. Let $G = (N, T, \rho, S, P)$ be a WK context-free grammar in terminal normal form where P consists of the following productions:

$$\begin{array}{ll}
S \to SS, & S \to \langle a/\lambda\rangle\langle\lambda/a\rangle S\langle b/\lambda\rangle\langle\lambda/b\rangle, \\
S \to \langle a/\lambda\rangle S, & S \to \langle a/\lambda\rangle A, \\
A \to \langle b/\lambda\rangle\langle\lambda/a\rangle A, & A \to \langle b/\lambda\rangle\langle\lambda/a\rangle B|\langle b/\lambda\rangle\langle\lambda/a\rangle, \\
B \to \langle\lambda/b\rangle B, & B \to \langle\lambda/b\rangle, \\
B \to BB, & B \to \langle a/\lambda\rangle\langle\lambda/a\rangle S\langle b/\lambda\rangle\langle\lambda/b\rangle, \\
B \to \langle a/\lambda\rangle S, & B \to \langle a/\lambda\rangle A.
\end{array}$$

To transform G into a WK context-free grammar in WK-Chomsky normal form, first, we introduce nonterminals $T_a^u, T_b^u, T_a^d, T_b^d$, and obtain the following productions:

$$\begin{array}{ll}
S \to SS, & S \to T_a^u T_a^d S T_b^u T_b^d, \\
S \to T_a^u S, & S \to T_a^u A, \\
A \to T_b^u T_a^d A, & A \to T_b^u T_a^d B \\
A \to T_b^u T_a^d, & \\
B \to T_b^d B, & B \to \langle\lambda/b\rangle, \\
B \to BB, & B \to T_a^u T_a^d S T_b^u T_b^d, \\
B \to T_a^u S, & B \to T_a^u A, \\
T_a^u \to \langle a/\lambda\rangle, & T_b^u \to \langle b/\lambda\rangle, \\
T_a^d \to \langle\lambda/a\rangle, & T_b^d \to \langle\lambda/b\rangle.
\end{array}$$

Next, we introduce additional nonterminals Y_i, $1 \le i \le 8$, to construct productions of WK-Chomsky normal form:

$$\begin{array}{llll}
S \to SS, & S \to T_a^u Y_1, & Y_1 \to T_a^d Y_2, & Y_2 \to SY_3, \\
Y_3 \to T_b^u T_b^d, & S \to T_a^u S, & S \to T_a^u A, & \\
A \to T_b^u Y_4, & Y_4 \to T_a^d A, & A \to T_b^u Y_5, & Y_5 \to T_a^d B, \\
A \to T_b^u T_a^d, & B \to T_b^d B, & B \to \langle\lambda/b\rangle, & \\
B \to BB, & B \to T_a^u Y_6, & Y_6 \to T_a^d Y_7, & Y_7 \to SY_8, \\
Y_8 \to T_b^u T_b^d, & B \to T_a^u S, & B \to T_a^u A, &
\end{array}$$
$$T_a^u \to \langle a/\lambda\rangle, T_b^u \to \langle b/\lambda\rangle, T_a^d \to \langle\lambda/a\rangle, T_b^d \to \langle\lambda/b\rangle.$$

6. A MODIFIED CYK ALGORITHM

Since the structure of productions of a WK context-free grammar is similar to the structure of those of a context-free grammar, we can attempt to adjust parsing (membership) algorithms for context-free grammars for their WK variants. In this section, we consider CYK (J. Cocke, D. Younger, T. Kasami) algorithm [16–18] for WK context-free grammars in WK-Chomsky normal form.

The CYK algorithm uses bottom-up dynamic programming approach to determine whether a given string w can be generated by a given context-free grammar G in Chomsky normal form. The strategy of the algorithm is to construct the sets of nonterminals from which each substring of length from one to $|w|$ can be generated. Thus, the algorithm in Step i constructs $(|w| - i + 1)$ sets for the substrings of length i, where $1 \le i \le |w|$, and in each step, the number of the sets decrease by one.

Based on CYK algorithm, we introduce a modified CYK algorithm, implemented to WK context-free grammar in WK-Chomsky normal form, called WK-CYK algorithm. Before delving into the algorithm in detail, we shall discuss the specifics of substrings of double-stranded strings.

The definition of substrings of a certain length for incomplete or complete double-stranded strings is considerably different from single-stranded strings. We have to consider the usual substrings from each strand and any possible combinations of substrings from both strands. For instance, if $w = \langle abc/a\rangle$, then we obtain the following substrings

Length 1: $\langle a/\lambda\rangle$, $\langle b/\lambda\rangle$, $\langle c/\lambda\rangle$, $\langle\lambda/a\rangle$
Length 2: $\langle ab/\lambda\rangle$, $\langle bc/\lambda\rangle$, $\langle a/a\rangle$, $\langle b/a\rangle$, $\langle c/a\rangle$
Length 3: $\langle abc/\lambda\rangle$, $\langle ab/a\rangle$, $\langle bc/a\rangle$
Length 4: $\langle abc/a\rangle$

Moreover, we have to consider both orders of two nonterminals generating substrings involving combinations of upper and lower strands. For instance, if $A \Rightarrow^* \langle ab/\lambda\rangle$ and $B \Rightarrow^* \langle\lambda/a\rangle$, then for the substring $\langle ab/a\rangle$, we need to find nonterminals producing AB and BA, thus, the order of generating substrings $\langle ab/\lambda\rangle$ and $\langle\lambda/a\rangle$ is irrelevant.

We can see another difference of double-stranded substrings from single-stranded substrings in their indexing according to the positions of the symbols. Let $w = x_1x_2\ldots x_n$ be a single-stranded string. Then, the substring $x_i\ldots x_j$ can be denoted by $x_{i,j}$. In the CYK algorithm, the set of nonterminals generating $x_{i,j}$ can be denoted accordingly by $X_{i,j}$, which contains the nonterminals that can generate the pairs of nonterminals one from each of $X_{i,k}$ and $X_{k+1,j}$ in this order for all k's between i and j.

Let $w = [x_{11}\ldots x_{1n}/x_{21}\ldots x_{2n}]$ be a double-stranded string. Consider a substring $\langle x_{1i}\ldots x_{1j}/x_{2k}\ldots x_{2l}\rangle$ of w. We use the notations $x_{i:j,k:l}$ and $X_{i:j,k:l}$ to denote the substring and the corresponding set of nonterminals generating this substring, i.e.

$$X_{i:j,k:l} = \{A \in N | A \Rightarrow^* \langle x_{1i}\ldots x_{1j}/x_{2k}\ldots x_{2l}\rangle\}.$$

If $x_{1i}\ldots x_{1j} = \lambda$ or $x_{2k}\ldots x_{2l} = \lambda$, then we use notations $x_{0:0,k:l}$ ($X_{0:0,k:l}$) or $x_{i:j,0:0}$ ($X_{i:j,0:0}$), respectively. Nonterminal A is in $X_{i:i,0:0}$ or in $X_{0:0,i:i}$ if and only if there is a production $A \to \langle x_{1i}/\lambda\rangle$ in P or $A \to \langle\lambda/x_{2i}\rangle$ in P, respectively.

On the other hand, nonterminal A is in $X_{i:j,k:l}$ if and only if there is a production $A \to BC$ such that

$$B \Rightarrow^* x_{i:s,k:t} \quad \text{and} \quad C \Rightarrow^* x_{s+1:j,t+1:l}$$

for some $i \leq s < j$ and $k \leq t < l$, or

$$B \Rightarrow^* x_{i:j,0:0} \quad \text{and} \quad C \Rightarrow^* x_{0:0,k:l},$$

or

$$B \Rightarrow^* x_{0:0,k:l} \quad \text{and} \quad C \Rightarrow^* x_{i:j,0:0}.$$

Thus, if $0 < i \leq j$ and $0 < k \leq l$, we define the set $X_{i:j,k:l}$ as

$$\begin{aligned} X_{i:j,k:l} = & \bigcup_{s\in[i,j-1],\ t\in[k,l-1]} \{A|A \rightarrow BC \in P, \\ & B \in X_{i:s,k:t},\ C \in X_{s+1:j,t+1:l}\} \\ & \cup \{A|A \rightarrow BC \in P,\ B \in X_{i:j,0:0}, C \in X_{0:0,k:l}\} \\ & \cup \{A|A \rightarrow BC \in P,\ B \in X_{0:0,k:l}, C \in X_{i:j,0:0}\}. \end{aligned}$$

If $i = j = 0$ or $k = l = 0$, we define the sets $X_{0:0,k:l}$ and $X_{i:j,0:0}$ as

$$\begin{aligned} X_{0:0,k:l} = & \bigcup_{t\in[k,l-1]} \{A|A \rightarrow BC \in P, \\ & B \in X_{0:0,k:t},\ C \in X_{0:0,t+1:l}\}, \\ X_{i:j,0:0} = & \bigcup_{s\in[i,j-1]} \{A|A \rightarrow BC \in P, \\ & B \in X_{i:s,0:0}, C \in X_{s+1:j,0:0}\}. \end{aligned}$$

Clearly, string $[w/w]$ is generated by a WK context-free grammar G if and only if $S \in X_{1:n,1:n}$, where S is the starting nonterminal in G.

The algorithm computes all the sets constructed above according to the lengths of the double-stranded substrings, i.e.

$$1 \leq (j - i + 1) + (l - k + 1) \leq 2n.$$

For instance, if $n = 2$ (the total length is 4), we construct the sets correspondingly to the sum of the lengths of upper and lower strands, i.e.

(1) The length is **1**: $1 + 0 = 0 + 1$

$$X_{1:1,0:0}, X_{2:2,0:0}, X_{0:0,1:1}, X_{0:0,2:2}.$$

(2) The length is **2**: $2 + 0 = 1 + 1 = 0 + 2$

$$X_{1:2,0:0}, X_{1:1,1:1}, X_{1:1,2:2}, X_{2:2,1:1}, X_{2:2,2:2}, X_{0:0,1:2}.$$

(3) The length is **3**: $2 + 1 = 1 + 2$

$$X_{1:2,1:1}, X_{1:2,2:2}, X_{1:1,1:2}, X_{2:2,1:2}.$$

(4) The length is **4**: $2 + 2$

$$X_{1:2,1:2}.$$

The next important question is what is the total number of $X_{i:j,k:l}$, which determines the time complexity of the algorithm.

LEMMA 6.1. *The number of the sets $X_{i:j,k:l}$ for all $0 \leq i, j \leq n$, is $O(n^4)$.*

Proof. Let $x = [x_{11}x_{12}\ldots x_{1n}/x_{21}x_{22}\ldots x_{2n}]$ be a complete double-stranded string. The number of the sets $X_{i:j,k:l}$ for all $0 \leq i, j \leq n$ can be computed by counting the combinations of all possible substrings of the upper strand and all possible substrings of the lower strand.

Since the upper and lower strands are of length n, there are $1 + 2 + \cdots (n - 1) + n + 1)$ substrings for each strand. This is because, i and j start from 0 and we obtain the substrings representing the upper strand of the set

$$\begin{aligned} & X_{i:j,k:l} \\ & = \{x_{0:0,k:l}, x_{1:1,k:l}, x_{1:2,k:l}, \ldots, x_{1:n,k:l}, \\ & x_{2:2,k:l}, x_{2:3,k:l}, \ldots, x_{2:n,k:l}, \ldots, x_{n:n,k:l}\}. \end{aligned}$$

There is no substring like $x_{2:1,k:l}$, $x_{3:2,k:l}$ and so on, because the position of the symbols represented by i and j must be sequential. The equation can be simplified to

$$\begin{aligned} & 1 + 2 + \cdots + (n - 1) + n + 1 \\ & = \frac{(n + 1)n}{2} + 1 \approx \frac{n^2}{2}. \end{aligned}$$

Thus, the number of all possible combinations of both upper and lower substrings is $\frac{n^2}{2}\frac{n^2}{2} = O(n^4)$. □

LEMMA 6.2. *The construction of each set $X_{i:j,k:l}$, $0 \leq i, j \leq n$, $i + 1 \geq 1$, requires at most $(n - 1)^2 + 2$ decompositions.*

Proof. Consider string

$$x = [x_{11}x_{12}\ldots x_{1n}/x_{21}x_{22}\ldots x_{2n}].$$

For $1 \leq i \leq j \leq n$ and $1 \leq k \leq l \leq n$, we compute the number of the decompositions of a substring $x_{i:j,k:l} = \langle x_{1i}\ldots x_{1j}/x_{2k}\ldots x_{2l}\rangle$.

(1) First, let us consider the special cases of the decomposition:

$$\begin{aligned} & \langle x_{1i}\ldots x_{1j}/x_{2k}\ldots x_{2l}\rangle \\ & = \langle x_{1i}\ldots x_{1j}/\lambda\rangle \cdot \langle \lambda/x_{2k}\ldots x_{2l}\rangle \\ & = \langle \lambda/x_{2k}\ldots x_{2l}\rangle \cdot \langle x_{1i}\ldots x_{1j}/\lambda\rangle \end{aligned}$$

where the decomposed substrings involve only one of the strands, i.e. we have two decompositions.

(2) Each decomposed substring contains at least one symbol from each strand:

$$\begin{aligned} & \langle x_{1i}\ldots x_{1j}/x_{2k}\ldots x_{2l}\rangle \\ & = \langle x_{1i}\ldots x_{1p}/x_{2k}\ldots x_{2q}\rangle \\ & \cdot \langle x_{1p+1}\ldots x_{1j}/x_{2q+1}\ldots x_{2l}\rangle \end{aligned}$$

where $1 \leq i \leq p \leq j - 1$ and $1 \leq k \leq q \leq l - 1$. Then, we have $(j - i)(l - k)$ decompositions.

Since $1 \le i \le j \le n$ and $1 \le k \le l \le n$, there are at most $(n-1)^2 + 2$ decompositions of the substring $x_{i:j,k:l}$. □

The algorithm demands that we calculate until the final set $X_{1:n,1:n}$, which can only be obtained after calculating all the sets $X_{i:j,k:l}$. Lemma 6.1 shows the number of iteration needed, i.e. the number of the sets $X_{i:j,k:l}$. Lemma 6.2 calculates the substrings contained in each set $X_{i:j,k:l}$.

From the two lemmas above, we obtain

THEOREM 6.1. *The time complexity of the WK-CYK algorithm is* $O(n^6)$.

Proof. From Lemma 6.2, the calculation of the substrings for each set $X_{i:j,k:l}$ requires at most $(n-1)^2 + 2$ iterations, which is $O(n^2)$. Then, from Lemma 6.1, there are $O(n^4)$ sets of $X_{i:j,k:l}$ to be calculated. Therefore, the total time complexity is $O(n^4)O(n^2) = O(n^6)$. □

6.1. The Algorithm: WK-CYK Algorithm

The algorithm, which is named as WK-CYK Algorithm, is divided into two sub-algorithms. Algorithm 1 works as the main procedure, where all necessary subsets are listed. The subsets are then computed by the function *Compute set* described by Algorithm 2.

In the algorithms, we write

$$X_{i:s,k:t}X_{g:j,h:l} = \{A \mid A \to BC,\ B \in X_{i:s,k:t},\ C \in X_{g:j,h:l}\},$$

where $0 \le i \le s \le g \le j \le n$ and $0 \le k \le t \le h \le l \le n$.

The inputs of Algorithm 1 are a double-stranded string $[w/w] = [x_{11}x_{12}\ldots x_{1n}/x_{21}x_{22}\ldots x_{2n}]$ which can be transformed to the terminal normal form $\langle x_{11}/\lambda\rangle\langle x_{12}/\lambda\rangle\ldots\langle x_{1n}/\lambda\rangle\langle\lambda/x_{21}\rangle\langle\lambda/x_{22}\rangle\ldots\langle\lambda/x_{2n}\rangle$.

We explain briefly some of the steps in Algorithm 1 as follows:

- lines 8–10: calculate the base sets, $X_{i:i,0:0}$ and $X_{0:0,i:i}$,
- line 12: y= total length of the substring,
- line 13: β= the length of the lower substring,
- line 14: α= the length of the upper substring,
- line 15: calculate only lower substring when there is no upper substring, using Algorithm 2,
- line 20: calculate only upper substring when there is no lower substring, using Algorithm 2,
- line 25: calculate both upper and lower substring, using Algorithm 2,
- line 32: finally, if $S \in X_{1:n,1:n}$, then it means that the string can be generated by G, and if $S \notin X_{1:n,1:n}$, then the string cannot be generated by G.

Algorithm 1 Sets construction

1: **procedure** SETS CONSTRUCTION
2: Input:
3: : string $[w/w] = [x_{11}x_{12}\ldots x_{1n}/x_{21}x_{22}\ldots x_{2n}]$
4: $= \langle x_{11}/\lambda\rangle\langle x_{12}/\lambda\rangle\ldots\langle x_{1n}/\lambda\rangle$
5: $\cdot\ \langle\lambda/x_{21}\rangle\langle\lambda/x_{22}\rangle\ldots\langle\lambda/x_{2n}\rangle$,
6: : WK context-free grammar G.
7:
8: **for** $1 \le i \le n$ **do**
9: $X_{i:i,0:0} = \{A\colon A \to \langle x_{1i}/\lambda\rangle\}$
10: $X_{0:0,i:i} = \{A\colon A \to \langle\lambda/x_{2i}\rangle\}$
11:
12: **for** $2 \le y \le 2n$ **do**
13: **for** $0 \le \beta \le n$**do**
14: $\alpha = y - \beta$
15: **if** $\alpha = 0$ **then**
16: $i = j = 0$
17: **for** $1 \le k \le n - y + 1$ **do**
18: $l = k + y - 1$
19: Compute set $X_{i:j,k:l}$
20: **else if** $\beta = 0$ **then**
21: $k = l = 0$
22: **for** $1 \le i \le n - y + 1$ **do**
23: $j = i + y - 1$
24: Compute set $X_{i:j,k:l}$
25: **else**
26: **for** $1 \le i \le (n - \alpha + 1)$ **do**
27: **for** $1 \le k \le (n - \beta + 1)$ **do**
28: $j = i + \alpha - 1$
29: $l = k + \beta - 1$
30: Compute set $X_{i:j,k:l}$
31:
32: **if** $S \in X_{1:n,1:n}$ **then**
33: $w \in L(G)$
34: **else**
35: $w \notin L(G)$.

Similarly, the inputs of Algorithm 2 are the same double-stranded string $[w/w]$ transformed to the terminal normal form, and the parameters i, j, k, l from Algorithm 1.

Algorithm 2 acts as a function to compute the set X_i: j, k: l. We explain briefly some of the steps in Algorithm 2 as follows:

- line 10: calculate only lower substring when there is no upper substring,
- line 15: calculate only upper substring when there is no lower substring,
- line 20: calculate both upper and lower substring.

Algorithm 2 Compute set

function COMPUTE SET
Input:
: string $[w/w] = [x_{11}x_{12}\ldots x_{1n}/x_{21}x_{22}\ldots x_{2n}]$
$= \langle x_{11}/\lambda\rangle\langle x_{12}/\lambda\rangle\ldots\langle x_{1n}/\lambda\rangle$
$\cdot\ \langle\lambda/x_{21}\rangle\langle\lambda/x_{22}\rangle\ldots\langle\lambda/x_{2n}\rangle$,
: WK context-free grammar G.
: parameters i, j, k, l from Algorithm 1.
Output: set $X_{i:j,k:l}$.

  if $i = j = 0$ **then**
    $X_{0:0,k:l} = \bigcup_{t\in[k,l-1]} X_{0:0,k:t}X_{0:0,t+1:l}\}$
  else if $k = l = 0$ **then**
    $X_{i:j,0:0} = \bigcup_{s\in[i,j-1]} X_{i:s,0:0}X_{s+1:j,0:0}\}$
  else

$$\begin{aligned}X_{i:j,k:l} = &\{X_{i:j,0:0}X_{0:0,k:l} \cup X_{0:0,k:l}X_{i:j,0:0}\} \cup \\ &\bigcup_{s\in[i,j-1],t\in[k,l-1]} \{X_{i:s,k:t}X_{s+1:j,t+1:l}\} \cup \\ &\bigcup_{s\in[i,j-1]} \{X_{i:s,k:l}X_{s+1:j,0:0} \cup X_{i:s,0:0}X_{s+1:j,k:l}\} \cup \\ &\bigcup_{t\in[k,l-1]} \{X_{i:j,k:t}X_{0:0,t+1:l} \cup X_{0:0,k:t}X_{i:j,t+1:l}\}\end{aligned}$$

6.2. Example

This section illustrates an example using WK-CYK algorithm.

EXAMPLE 4. Let G be the WK context-free grammar in Chomsky normal form, obtained in Example 3. Is $\{ab\} \in L(G)$?

Here, the target string $[w/w]$ is $[ab/ab]$. Based on

$$\begin{aligned}[w/w] &= [x_{11}x_{12}/x_{21}x_{22}] \\ &= \langle x_{11}/\lambda\rangle\langle x_{12}/\lambda\rangle\ldots\langle x_{1n}/\lambda\rangle \\ &\quad\cdot\ \langle\lambda/x_{21}\rangle\langle\lambda/x_{22}\rangle\ldots\langle\lambda/x_{2n}\rangle,\end{aligned}$$

we get $[ab/ab] = \langle a/\lambda\rangle\langle b/\lambda\rangle\langle\lambda/a\rangle\langle\lambda/b\rangle$.

From Algorithm 1, first, we list the base sets $X_{i:i,0:0}$ and $X_{0:0,i:i}$ for $1 \le i \le n$. Then, from Algorithms 1 and 2, we list the sets $X_{i:j,k:l}$, $0 \le i, j \le n$, $0 \le k, l \le n$, which corresponds to the relations of the symbols in $[w/w]$.

(1) The length is **1**: $1 + 0 = 0 + 1$:

$$X_{1:1,0:0},\ X_{2:2,0:0},\ X_{0:0,1:1},\ X_{0:0,2:2}.$$

(2) The length is **2**: $2 + 0 = 1 + 1 = 0 + 2$:

$$X_{1:2,0:0},\ X_{1:1,1:1},\ X_{1:1,2:2},\ X_{2:2,1:1},\ X_{2:2,2:2},\ X_{0:0,1:2}.$$

(3) The length is **3**: $2 + 1 = 1 + 2$:

$$X_{1:2,1:1},\ X_{1:2,2:2},\ X_{1:1,1:2},\ X_{2:2,1:2}.$$

(4) The length is **4**: $2 + 2$:

$$X_{1:2,1:2}.$$

The above sets are computed in Algorithm 2. The computation is as follows:

(1) Length **1**:

$$X_{1:1,0:0} = \{T_a^u\},\ X_{2:2,0:0} = \{T_b^u\},$$

$$X_{0:0,1:1} = \{T_a^d\},\ X_{0:0,2:2} = \{T_b^d, B\}.$$

(2) Length **2**:

$$\begin{aligned}X_{1:2,0:0} &= X_{1:1,0:0}X_{2:2,0:0} = \{T_a^u\}\{T_b^u\} = \varnothing, \\ X_{1:1,1:1} &= X_{1:1,0:0}X_{0:0,1:1} = \{T_a^u\}\{T_a^d\} = \varnothing \\ &\quad\cup X_{0:0,1:1}X_{1:1,0:0} = \{T_a^d\}\{T_a^u\} = \varnothing, \\ X_{1:1,2:2} &= X_{1:1,0:0}X_{0:0,2:2} = \{T_a^u\}\{T_b^d, B\} = \varnothing \\ &\quad\cup X_{0:0,2:2}X_{1:1,0:0} = \{T_b^d, B\}\{T_a^u\} = \varnothing, \\ X_{2:2,1:1} &= X_{2:2,0:0}X_{0:0,1:1} = \{T_b^u\}\{T_a^d\} = \{A\} \\ &\quad\cup X_{0:0,1:1}X_{2:2,0:0} = \{T_a^d\}\{T_b^u\} = \varnothing, \\ X_{2:2,2:2} &= X_{2:2,0:0}X_{0:0,2:2} = \{T_b^u\}\{T_b^d, B\} = \{Y_3, Y_8\} \\ &\quad\cup X_{0:0,2:2}X_{2:2,0:0} = \{T_b^d, B\}\{T_b^u\} = \varnothing, \\ X_{0:0,1:2} &= X_{0:0,1:1}X_{0:0,2:2} = \{T_a^d\}\{T_b^d, B\} = \{Y_5\}.\end{aligned}$$

(3) Length **3**:

$$\begin{aligned}X_{1:2,1:1} &= X_{1:2,0:0}X_{0:0,1:1} = \varnothing\{T_a^d\} = \varnothing \\ &\quad\cup X_{0:0,1:1}X_{1:2,0:0} = \{T_a^d\}\varnothing = \varnothing \\ &\quad\cup X_{1:1,0:0}X_{2:2,1:1} = \{T_a^u\}\{A\} = \{S,\ B\}, \\ &\quad\cup X_{1:1,1:1}X_{2:2,0:0} = \varnothing\{T_b^u\} = \varnothing, \\ X_{1:2,2:2} &= X_{1:2,0:0}X_{0:0,2:2} = \varnothing\{T_b^d, B\} = \varnothing \\ &\quad\cup X_{0:0,2:2}X_{1:2,0:0} = \{T_b^d, B\}\varnothing = \varnothing \\ &\quad\cup X_{1:1,0:0}X_{2:2,2:2} = \{T_a^u\}\{Y_3, Y_8\} = \varnothing \\ &\quad\cup X_{1:1,2:2}X_{2:2,0:0} = \varnothing\{T_b^u\} = \varnothing, \\ X_{1:1,1:2} &= X_{1:1,0:0}X_{0:0,1:2} = \{T_a^u\}\{Y_5\} = \varnothing \\ &\quad\cup X_{0:0,1:2}X_{1:1,0:0} = \{Y_5\}\{T_a^u\} = \varnothing \\ &\quad\cup X_{1:1,1:1}X_{0:0,2:2} = \varnothing\{T_b^d, B\} = \varnothing \\ &\quad\cup X_{0:0,1:1}X_{1:1,2:2} = \{T_a^d\}\varnothing = \varnothing, \\ X_{2:2,1:2} &= X_{2:2,0:0}X_{0:0,1:2} = \{T_b^u\}\{Y_5\} = \{A\} \\ &\quad\cup X_{0:0,1:2}X_{2:2,0:0} = \{Y_5\}\{T_b^u\} = \varnothing \\ &\quad\cup X_{2:2,1:1}X_{0:0,2:2} = \{A\}\{T_b^d, B\} = \varnothing \\ &\quad\cup X_{0:0,1:1}X_{2:2,2:2} = \{T_a^d\}\{Y_3, Y_8\} = \varnothing.\end{aligned}$$

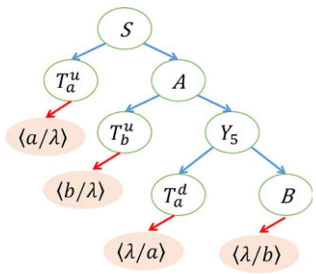

FIGURE 3. Derivation tree for Example 4 resulting from the proposed modified CYK algorithm.

(4) Finally, length **4**:

$$\begin{aligned} X_{1:2,1:2} &= X_{1:2,0:0}X_{0:0,1:2} = \varnothing\{Y_5\} = \varnothing \\ &\cup X_{0:0,1:2}X_{1:2,0:0} = \{Y_5\}\varnothing = \varnothing \\ &\cup X_{1:1,1:1}X_{2:2,2:2} = \varnothing\{Y_3, Y_8\} = \varnothing \\ &\cup X_{1:1,0:0}X_{2:2,1:2} = \{T_a^u\}\{A\} = \{S, B\} \\ &\cup X_{1:1,1:2}X_{2:2,0:0} = \varnothing\{T_b^u\} = \varnothing \\ &\cup X_{1:2,1:1}X_{0:0,2:2} = \{S\}\{T_b^d, B\} = \varnothing \\ &\cup X_{0:0,1:1}X_{1:2,2:2} = \{T_a^d\}\varnothing = \varnothing. \end{aligned}$$

As $S \in X_{1:2,1:2}$, therefore, $w \in L(G)$.

With this algorithm, we are able to solve the membership problem for WK context-free grammars. The derivation tree is shown in Figure 3.

7. CONCLUSION

In this paper, we investigated the simplification processes of WK context-free grammars and introduced a normal form based on Chomsky normal form. We showed that

- similar to (Chomsky) context-free grammars, WK context-free grammars also possess the leftmost and rightmost derivation;
- the processes of substitution, removing λ-production and unit production in WK context-free grammars are similar with (Chomsky) context-free grammars;
- as WK context-free grammars generates double-stranded strings, the normal form differs from Chomsky normal form in the types of terminal symbols generated (Chomsky normal form has one type $A \to a$, while WK context-free's has two types);
- in CYK algorithm for WK context-free grammars, the relationship (the mutual position, neighborhood, and ρ-relation) between two nonterminals i.e. length 2, not only in the same strand but between the upper strand and lower strand, plays an important role in finding the relationship for the next length, thus solving the membership problem.

The following problems related to the topic remain open:

- How to determine if a WK context-free grammar will always generate complete double-stranded strings?
- Is the finiteness, emptiness and equivalence problems of WK context-free grammars decidable?
- What are other normal forms for WK context-free grammars?
- How the proposed algorithms can be improved? For example, can we improve WK-CYK algorithm by calculating the Cartesian products of the sets of nonterminals?

The answers to these questions will lead to more discovery on parsing using double-stranded strings, more possibilities in natural language processing, and DNA-based computing theories in general.

FUNDING

This research was supported by Ministry of Education Malaysia and International Islamic University Malaysia through Fundamental Research Grant Scheme [FRGS13-066-0307].

REFERENCES

[1] Freund, R., Paun, G., Rozernberg, G. and Salomaa, A. (1999) Watson–Crick finite automata. *DNA Based Comput. Three*, **48**, 297.

[2] Păun, G., Rozenberg, G. and Salomaa, A. (1998) *DNA Computing. New Computing Paradigms*. Springer-Verlag.

[3] Tamrin, M.I.M., Turaev, S. and Sembok, T.M.T. (2014) Weighted Watson–Crick Automata. *Proc. 21st Natl. Symp. Mathematical Sciences (SKSM21): Germination of Mathematical Sciences Education and Research towards Global Sustainability*, Penang, Malaysia, pp. 302–306. AIP Publishing.

[4] Czeizler, E. and Czeizler, E. (2006) Parallel communicating Watson–Crick automata systems. *Acta Cybern.*, **17**, 685–700.

[5] Czeizler, E. and Czeizler, E. (2006) A short survey on Watson–Crick automata. *Bull. EATCS*, **88**, 104–119.

[6] Okawa, S. and Hirose, S. (2006) The relations among Watson–Crick automata and their relations with context-free languages. *IEICE Trans. Inf. Syst.*, **E89**, 2591–2599.

[7] Subramanian, K., Venkat, I. and Mahalingam, K. (2011) Context-Free Systems with a Complementarity Relation. *Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pp. 194–198. IEEE.

[8] Subramanian, K., Hemalatha, S. and Venkat, I. (2012) On Watson–Crick Automata. *CCSEIT'12 Proc. Second Int. Conf.*

Computer Science, Science, Engineering and Information Technology, Coimbatore, India, pp. 151–156.

[9] Mohamad Zulkifli, N., Turaev, S., Mohd Tamrin, M. and Messikh, A. (2015) Watson–Crick Linear Grammars. *Lecture Notes in Electrical Engineering (DaEng 2015)*, Bali, Indonesia.

[10] Mohamad Zulkufli, N., Turaev, S., Mohd Tamrin, M. and Messikh, A. (2015) Closure Properties of Watson–Crick Grammars. *Proc. 2nd Innovation and Analytics Conf. Exhibition (IACE)*, Kedah, Malaysia.

[11] Mohamad Zulkufli, N., Turaev, S., Mohd Tamrin, M., Messikh, A. and Alshaikhli, I.F.T. (2015) Computational Properties of Watson–Crick Context-Free Grammars. *4th Int. Conf. Advanced Computer Science Applications and Technologies (ACSAT)*, Kuala Lumpur, Malaysia, December, pp. 186–191.

[12] Linz, P. (2006) *An Introduction to Formal Languages and Automata* (4th edn). Jones and Bartlett Publishers, Inc.

[13] Rozenberg, G. and Salomaa, A. (1997) *Handbook of Formal Languages*, **Vols. 1–3**. Springer-Verlag.

[14] Mohamad Zulkufli, N., Turaev, S., Mohd Tamrin, M. and Messikh, A. (2016) Generative power and closure properties of Watson–Crick grammars. *Appl. Comput. Intell. Soft Comput.*, **2016**, 12. http://dx.doi.org/10.1155/2016/9481971.

[15] Sudkamp, T. (1998) *Languages and Machines: An Introduction to the Theory of Computer Science* (2nd edn). Addison-Weasley.

[16] Cocke, J. and Schwartz, J.T. (1970) *Programming Languages and Their Compilers*. Courant Institute of Mathematical Sciences.

[17] Younger, D.H. (1967) Recognition and parsing of context-free languages in time n3. *Inf. Control*, **10**, 372–375.

[18] Kasami, T. (1965) An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages. Technical Report Technical Report AFCRL-65-758. Air Force Cambridge Research Laboratory, Bedford, MA.