# Essential Context Free Expression (part 1)

Charles Qiuen Yu

December 25, 2020

# Essential Context Free Expression (part 1)

Charles Qiuen Yu[0000−0002−0347−0724]

Pleasanton, CA 94566, USA
`charlesyuq@gmail.com`
`https://scholar.google.com/citations?hl=en&user=pcgAq3QAAAAJ`

**Abstract.** We introduce a system of string pattern specification, notation $ECFX$ for Essential Context Free Expression, as an extension of $CFX$, notation for Context Free Grammar.
The approach of $ECFX$ is seen in three methodological principles: using $CFX$ rules for syntax kernel, allowing arbitrary means of semantic condition for extra control, subject only to a complexity limit, applying the proper complexity of $CFX$ as uniform constraint to all semantic conditions.
The rule format of $ECFX$ is $X \to x[\mathbf{c}]$ where $X \to x$ is a usual $CFX$ rule, and $\mathbf{c}$ is an optional semantic condition on the strings that would match $x$. Let $CFTIME$ stands for the time complexity of $CFX$ for fixed pattern. A pattern $X$ is in $ECFX$ if $X$ is equivalent to a finite set of $ECFX$ rules where all semantic conditions involved are in $CFTIME$.
For characteristics of $ECFX$:
$ECFX$ imposes $CFX$ structure on all its member patterns;
$ECFX$ is *complexity complete* in the sense that for any pattern $X$, $X$ is in $ECFX$ if and only if the fixed pattern decision problem for $X$ is in $CFTIME$;
$ECFX$ as language class is a full trio as it is closed on all full trio required operations [8];
$ECFX$ is in $O(n^3)$ for fixed-pattern complexity;
$ECFX$ is closed on the following operations: a. constrained concatenation as a generalization of back referencing with a restriction, b. constrained iteration for an extension of Kleene star with arbitrary number of iterations synchronized and crossed, and c. permutation for casting any $ECFX$ pattern as a permutation pattern of base patterns.

**Keywords:** String pattern match · Pattern specification · Stringology · Context free grammar · Back referencing · Semantic condition · Semi syntactic · Cumulative time

## 1 Introduction

By a string pattern we mean a logical expression that determines a set of strings. In this sense, any formal grammar represents a string pattern, as it determines a set of strings, also referred to as the language of the grammar. Conversely, any set of strings determines a string pattern, even though the same pattern may be represented by different logical expressions. Practically, given two expressions of the same string set, the one that is most convenient or readiest for systematic computing may serve as pattern expression, whereas the one that is less so (or perhaps most convenient or readiest for enumerating sample members) may serve as set expression.

Since we feel idiosyncratically that *expression* connotes better than *grammar* does to emphasize that a pattern is an abstraction of a string set independent of any concrete application areas, we develop our ideas primarily in terms of expressions, in spite of the fact that we do use terms of expression and grammar interchangeably when it helps.

Based on the notion of string pattern as specified, the general instance of decision problem for string pattern match, say $SPM$ for short, is in format $I = (X, u)$, where $X$ is a pattern, $u$ is a string, and the question is whether $u$ is in $X$. By contrast, an important variant of $SPM$ is $SSPM$, short for substring pattern match, $SSPM$ and $SPM$ have the same instance format $I = (X, u)$, but the question for $SSPM$ instance is which positioned substrings of $u$ if any are in $X$. $SSPM$ thus specified is a search problem; but it can also serve as a decision problem in the sense in which the question for any instance is whether any substring of $u$ is in $X$.

$SPM$ and $SSPM$ have the same order of complexity, with $SSPM$ treated either as a decision problem or as a search one. We will focus on direct issues of $SPM$ only in this paper.

In terms of expression, the classical systems for string pattern specification include the following:

1. $RX$ for (the class of) Regular Expressions or Regular Grammars
2. $CFX$ for Context Free Grammars
3. $CSX$ for Context Sensitive Grammars
4. $REX$ for Recursively Enumerable Grammars
5. $REWBR$ for the extension of $RX$ with back referencing

In view of the above, we introduce and explore a generalization of $CFX$, notation $ECFX$ for Essential Context Free Expression. The approach of $ECFX$ can be seen in its three methodological principles:

1. (syntax) maintaining context free syntax as kernel structure to organize all components of a pattern
2. (semantics) allowing optional semantic conditions for finer logic controls
3. (complexity) imposing a universal complexity upper bound over all semantic conditions

To present $ECFX$ from a broader perspective, we start with a more general notion: $UCFX$ for Unrestricted (Essential) Context Free Expression.

The format of a $UCFX$ rule is $X \rightarrow x[\mathbf{c}]$ where $X \rightarrow x$ is a rule of $CFX$, with $X$ as nonterminal, $x$ as a string of terminals and nonterminals, and $\mathbf{c}$ represents an optional semantic condition. A pattern $X$ is in $UCFX$ if $X$ is equivalent to a finite set of $UCFX$ rules with a designated start nonterminal say $S_X = X$ (allowing a bit of terminology abuse). $X$ matches or say produces string $u$ by $x$ if and only if $x$ produces $u$ in the standard sense of $CFX$ and condition $\mathbf{c}$ holds on $u$, say $\mathbf{c}(u)$ evaluates true.

To get $ECFX$ from $UCFX$, let $CFTIME$ stand for the proper time complexity of $CFX$ for fixed pattern; also let $ECFTIME$ stand for the max time complexity such that for any $X$ in $UCFX$, if all semantic conditions of $X$ are in $ECFTIME$ then $X$ is in $CFTIME$.

Though defined differently, $CFTIME$ and $ECFTIME$ will be shown to be equivalent. Now $ECFX$ can be defined such that for any pattern $X$ in $UCFX$, $X$ is in $ECFX$ if and only if all semantic conditions of $X$ are in $ECFTIME$. We show that $CFTIME$ and $ECFTIME$ are the same. In a separate paper we present an algorithm for $ECFX$ in Earley scheme [5] and prove that $ECFX$ is in $O(n^3)$.

For expressiveness of $ECFX$, we show that $ECFX$ is *complexity complete* in the sense that for any pattern $X$, if $X$ is in $CFTIME$, then $X$ is in $ECFX$. By contrast, $CFX$ is not complexity complete.

In part to substantiate the superiority of $ECFX$ to $CFX$ for expressiveness and prowess, we show that $ECFX$ is closed under all the following pattern operations:

1. basic Boolean operations including conjunction and negation
2. $ECFX$ substitution as an extension of $CFX$ substitution
3. inverse finite homomorphism as an extension of $CFX$ inverse homomorphism
4. constrained concatenation as an extension of back referencing with a restriction of sequentiality
5. constrained iteration for a generalization of Kleene star iteration
6. permutation for casting any $ECFX$ pattern as a permutation pattern of base patterns.

As to complexity of $ECFX$, to be clear first, we mean it for fixed pattern only in this paper unless otherwise indicated. This is mainly because semantic conditions are opaque in size in general. For pure syntactic systems, the decision problems of $SPM$ and $SSPM$ both have two versions: uniform and fixed pattern; for $ECFX$ however, it has only one meaningful version: fixed pattern.

There are four classes in Chomsky hierarchy, all of pure syntax. $REX$ and $CSX$ as the top two are complexity complete; $CFX$ and $RX$ as the bottom two are complexity-incomplete. $ECFX$ is complexity complete but not purely syntactic. This leads to certain **open question**s as to wether there is a system for pattern specification properly between $CSX$ and $CFX$ that is both complexity complete and purely syntactic; more generally, wether there is any such system properly below $CSX$, and if so which are they; whether there is a complexity complete system which is properly below $ECFX$.

In our analysis, there is need in explicitly defining what may be called *cumulative $CFTIME$ requirement.* Briefly, for any semantic condition $\mathbf{c}$, $\mathbf{c}$ is in $\mathbf{c}CFTIME$ if for any string $u$, whether there is $v$ such that $v = v_L u v_R$ for some $v_L, v_R$ matches $\mathbf{c}$ (or say $u$ infix matches $\mathbf{c}$) can be determined in $CFTIME$ at size $|u|$. We conjecture that $cCFTIME = CFTIME$.

## 1.1 Justification

Comparing $ECFX$ with $CSX$, the essence of $ECFX$ is still in *context freeness* or say *context independence.* It is so in the sense that, contrary to $CSX$ patterns, for any pattern $X$ in $ECFX$, any component pattern $Y$ of $X$, and any positioned substring $v$ of any string $u$, whether or not $Y$ and $v$ match depends only on $X$ and $v$, independent of which left or right contexts $v$ has within $u$. Note that this nature of context independence applies to any $CSX$ pattern, but not to any component pattern of a $CSX$ pattern in general. Our study so far seems to suggest strongly that many $CSX$ patterns are actually context free in nature, in a sense. Interestingly, $UCFX$ and $ECFX$ share the same essence, even though $UCFX$ is higher than $ECFX$ in complexity, as shown later in the randomness example. So it is a meta open question as to how much potential gain there is in exploring $ECFX$.

**Motivation and competing approaches** There have long been competing goals of string pattern computing, four of them being prominent, what we may call a quartet contention. That is, it is desirable to have a system that has the following simultaneously:

elegance of theoretical foundation,
extensiveness/expressiveness of pattern specification,
efficiency and convenience of pattern design and implementation,
and efficiency of pattern processing.

There have also been competing approaches towards systems of string pattern specification. Those for our concerns here may be aptly labelled *pure syntactic*, *semi syntactic*, and *non-syntactic.* Roughly, for any class $\mathcal{X}$ of pattern expressions with its own base syntax, if all possible components and aspects of $\mathcal{X}$ can be defined by means of its base syntax, then $\mathcal{X}$ is *pure syntactic*, otherwise *semi syntactic.* If syntax issues and restrictions are mostly ignored, then it is non-syntactic.

Our motivation is to pursue the semi syntactic to strike a good balance of the quartet contention. With hindsight, the guideline for reaching out $ECFX$ is that in order to achieve an attractive tradeoff between competing goals and approaches, it is better to factorize them out first to certain extent; and then recombine some of them as a clear choice.

**Practical advantages** There are big advantages of $ECFX$ on all three practically oriented goals in the competing goal quartet contention, or so we claim. For extensiveness/expressiveness of pattern specification, we mention particularly that many results in stringology commonly viewed as non-syntactic can be aptly represented and integrated in $ECFX$. Related to this, many sequence-oriented computing problems including those regarding super/sub sequencing, sequence alignment, and approximate pattern match can be cast as $ECFX$ pattern match problems.

For efficiency and convenience of pattern design and implementation, we claim that there are huge variety of patterns which are very difficult or intellectually challenging to design but can become easy and trivial with $ECFX$. Our collection of string pattern operations in our subsequent paper helps show this. In particular, $ECFX$ encourages pattern library building. We imagine that $ECFX$ may play big role in building next generation high quality pattern libraries.

For efficiency of pattern processing, it is fair to say that $ECFX$ as a pattern specification system does not auto lead to faster algorithms. But we think that the framework of $ECFX$ offers unique help in better organizing existing algorithms for pattern match, and in composing new algorithms out of existing ones.

For special application areas, we mention bio computing, especially bio sequence analysis and alignment, and text computing.

**Theoretical importance and elegance** $ECFX$ has the following characteristics: $ECFX$ is having all its member patterns based on $CFX$ structure, semi syntactic, complexity complete as the least superclass of $CFX$, qualified as full trio and closed on many other important operations.

## 1.2 Related work

**Chomsky Hierarchy** Chomsky Formal Language Hierarchy consists of four classes: from type-0 to type-3, or $REX, CSX, CFX$, and $RX$, in our jargon. [4] All the four classes are pure syntactic but only $REX$ and $CSX$ are complexity complete in the sense that any string pattern that has complexity in that of $CSX$ or $REX$ can be expressed by means of $CSX$ or $REX$. By contrast, $ECFX$ is semi syntactic and complexity complete. To our knowledge and understanding, $ECFX$ is the first pattern class characterized as such. By the way, a huge variety of results in stringology, which may be meaningfully viewed as non-syntactic, may be representable as semi syntactic within $ECFX$.

**Mildly context sensitive grammars** Let $MCSX$ stands for (the class of) mildly context sensitive grammars. $MCSX$ was introduced by Aravind Joshi in 1985 [**?**], for the sake of covering all polynomial extensions of $CFX$. For major representatives of $MCSX$, see [17] The variety of representatives of $MCSX$ highlights a dichotomy between pure syntactic and semi syntactic. Perhaps a majority of proposals in $MCSX$ are pure syntactic, or so as we are impressed. A few of semi syntactic include $RCX$ for Range Concatenation Grammar [2] and $GCFX$ for Generalized context-free grammar [15], [16]. We think so because $RCX$ and $GCFX$ allow use of unlimited sets of predicates or functions which are undefinable in their base syntaxes respectively.

A special one of the pure syntactic category is $BCFX$, for Boolean Extension of $CFX$, due to Pierre Boullier 2000 and Alexander Okhotin 2001 [3], [12] [13]. To our knowledge, this is the only one in $MCSX$ (though unmentioned in [17]) shown to be in $n$-cube and likely in $CFTIME$. We wonder whether there are other pure syntactic extensions of $CFX$ which are in $n$-cube and likely in $CFTIME$.

***REWBR*** Back Referencing as an extension of $RX$ is proposed as part of programming language SNOBOL in about 1964 [6]. To our attention there, a distinction of SNOBOL is to have (string) patterns treated as "first class data types whose values can be manipulated in all ways".

While $REWBR$ has been widely respected, there have been issues over the completeness of its definition. Some researchers, D. D. Freydenberger for one, pointed out that there are defects in its standard definition, as some aspects of it are left "under specified" [7] (2013). For complexity status, since $REWBR$ is pure syntactic, its decision problem has two versions: uniform and fixed pattern. Alfred V. Aho proved in 1990 that the uniform decision problem for $REWBR$ is $NP$ complete [1]. We will show in our paper (part 2) that the exact type of back referencing in $REWBR$ may be more properly called identity referencing; a wild generalization of identity referencing, named constrained concatenation, is supported in $ECFX$ with a natural restriction. called sequentiality; and $NP$ completeness of $REWBR$ is rooted in violation of sequentiality by virtual disjunction.

***CFX* inner hierarchy** Let $ES$ denotes the Earley algorithmic scheme. $ES$ has a nice auto-switch property to the effect that, for fixed pattern, $ES$ is in $O(n^3)$ for all $CFX$, in $O(n^2)$ for all unambiguous $CFX$, and in $O(n)$ for all deterministic $CFX$ [5]. On the other hand, there have been a series of asymptotic upper bounds on $CFTIME$, the best so far is near $O(n^{2.373})$, approaching $O(n^{2\frac{1}{3}})$, thanks to Leslie GValiant 1975 for initiating the approach [14], and to François Le Gall 2014 for a latest [10]. These results prompt us to consider a complexity incomplete hierarch in $CFX$, and in turn to consider a corresponding complexity complete hierarch in $ECFX$.

## 2 *UCFX* and *ECFX*

In this section, we first define $UCFX$ as a superclass of $CFX$, with the notion of semantic condition being used but unspecified. Next we expand on the roles and meanings of semantic condition in detail. Then we define the notion of match (or production) for $UCFX$ without applying complexity constraints. Finally we define $ECFX$ as a subclass of $UCFX$ with a uniform complexity constraint.

## 2.1 *UCFX*

**Definition 21** *A $UCFX$ is a system say $X = \langle \mathcal{T}, \mathcal{N}, \mathcal{R}, \mathcal{C}, S \rangle$ where*

*$\mathcal{T}$ is a finite set of symbols called terminal characters or simply terminals;*
*$\mathcal{N}$ is a finite set of symbols called nonterminal characters or nonterminals;*
*$S = S_X$ is a special nonterminal in $\mathcal{N}$ designated as the starting nonterminal of $X$;*
*$\mathcal{C}$ is a finite set of predicate symbols, each representing a semantic condition; $\mathcal{T}, \mathcal{N}, \mathcal{C}$ are mutually disjoint; $\mathcal{U} = \mathcal{T} \cup \mathcal{N}$ is referred to as* expression alphabet*;*
*$\mathcal{R}$ is a finite set of (production) rules in the format of $r : N \to x[\mathbf{c}]$ where*
- *$r$ (optional) is a symbolic token for easy reference, unique across $\mathcal{R}$;*
- *$N$ is in $\mathcal{N}$, referred to as the* rule head *of $r$;*
- *$x$ is a string out of $\mathcal{U}$, referred to as the* rule body *of $r$;*
- *$kr : N \to x$ is referred to as the* kernel rule *of $r$, notation $kr = KR(r)$; accordingly, notation $X_K = KR(X) = KR(X.\mathcal{R})$, called the kernel version of $X$, denotes the set of all kernel rules of $X$, with the same start nonterminal as that of $X$;*
- *$\mathbf{c}$ is the* semantic condition *of $r$, possibly empty.*

Below we make certain explanatory notes, all with reference to $X$ and its components as of the definition.

For notational convenience, we will equate $X$ with its starting symbol $S = S_X$. Thus saying that $x$ matches $X$ is equivalent to saying that $x$ matches $S$.

For simplicity, we assume that all elements in $\mathcal{T}, \mathcal{N}, \mathcal{C}$ are actually used in $\mathcal{R}$; so a full listing of all rules in $\mathcal{R}$ is equivalent to $X$.

The necessary non empty components of $X$ are $\mathcal{T}, \mathcal{N}, \mathcal{R}, S$. $\mathcal{C}$ is optional. If it is empty, then each rule of $X$ is a kernel rule, and $X$ is a $CFX$.

From the above, given any pattern $X$ in $UCFX$, $Y = KR(X)$ as set of all kernel rules of $X$ represents a pattern in $CFX$. In a special case, if $X$ is in $CFX$, then $X = KR(X)$.

Finally, we assume that all nonterminals are reachable from $S$ and terminable. From this assumption, for any nonterminal $Y$ in $X$, a pattern as represented by $Y$ can be induced from $X$, notation $Y = CP(Y, X)$, and will be referred to as a *component pattern* of $X$.

## 2.2 Semantic conditions

As to what may qualify as a semantic condition for $UCFX$, the general answer is that any constraint may qualify so long as it represents a computable constraint on a set of strings that can be matched by a $CFX$ kernel rule. More specifically, a semantic condition $\mathbf{c}$ can be viewed as a predicate with its unary argument on a set of strings. Given rule $r : N \to x[\mathbf{c}]$, the domain of $\mathbf{c}$ is the set of strings that match $x$.

Because of this, if the kernel of $r$ is a terminal rule, that is, $x$ is a terminal string, then $\mathbf{c}$ should be empty, meaning that it always evaluates to true, as a terminal rule can only match a single string.

It is possible that certain patterns can be achieved either by pure $CFX$ rules or by proper $UCFX$ rules. For example, pattern $X = a^i b a^i$ is representable in $CFX$, but it can also be represented by a rule like $X \to a^i b a^j[\mathbf{c}]$ where $\mathbf{c}$ requires that $i = j$.

**Examples** For possible uses of semantic condition, we give a few examples below: all with reference to generic rule $r : N \to x[\mathbf{c}]$.

1. (**conjunction**) $x$ can be any $CFX$ rule body, $\mathbf{c}$ requires that any string that matches $x$ match $y$ where $y$ is another string pattern, possibly in $CFX$.
   This example shows that logical conjunction can be implemented by a semantic condition.
2. (**negation**) $x = Y$ is a match-all nonterminal, $\mathbf{c}$ requires that any string that matches $x$ *not* match nonterminal $\mathcal{A}$ with any rule of $\mathcal{A}$. Thus $x[\mathbf{c}]$ is equivalent to the negation of $\mathcal{A}$.
   This example shows how logical negation can be implemented by a semantic condition.
3. (**back referencing**) $x = Yx_0Y[\mathbf{c}]$, where $x_0$ is an arbitrary component pattern, $Y$ is a nonterminal having multiple matches and $\mathbf{c}$ requires that any string $u$ that matches $x$ be representable as $u_1u_2u_3$ such that $u_1, u_3$ match $Y$, $u_2$ match $x_0$, and $u_1 = u_2$ (for identity).
   This example shows how back referencing as of $REWBR$ may be implemented by a semantic condition; and it is more aptly called *identity* referencing for two reasons: 1. nonidentity referencing can be easily supported; 2. the phrase 'back' lost its logical basis.
4. (**constrained concatenation**) $x = Y_1X_1Y_2X_2Y_3[\mathbf{c}]$, where $X_1, X_2, Y_1, Y_2, Y_3$ are all arbitrary component patterns such that each matches multiple strings, and $\mathbf{c}$ requires that for any string $u$ that matches $x$, $u$ be representable as $u_1v_1u_2v_2u_3$ such that $u_1, u_2, u_3$ match $Y_1, Y_2, Y_3$ respectively in the order, $v_1, v_2$ match $X_1, X_2$ respectively in the order, $u_3$ be a common super sequence of $u_1, u_2$ with an additional length requirement. $v_2$ be the reverse of $v_2$.
   This example helps show how identity referencing (above), may be generalized. Largely, if, in question, the main pattern consists of multiple sequential (so to speak) component patterns, and the semantic condition requires that for any target string $u$ to match the main pattern, $u$ consist of substrings which, while matching those sequential component patterns respectively, satisfy certain additional constraints (multiple in general) in terms of those substrings as referenced.
5. (**constrained individual iteration**) $x = a^i[\mathbf{c}]$, where $\mathbf{c}$ requires that for integer $n$ $i = f(n)$ for a number function $f$ with $TIME(f(n))$ being in $O(poly \log n)$.

This example shows how ingenuity demanding it might be to implement certain seemingly simple functionalities without using semantic conditions, and how individual free iteration may be generalized.

6. (**constrained iteration**) $x = x_1^{i_1} x_2^{i_2} x_3^{i_1} x_4^{f_1(i_2)} x_5^{i_1} x_6^{f_2(i_2)}[\mathbf{c}]$, where $\mathbf{c}$ requires that $x_1, x_3, x_5$ be iterated by the same number $i_1$, $x_2, x_4, x_6$ be iterated by numbers $i_2$, $f_1(i_2)$, and $f_1(i_2)$ respectively where $f_1, f_2$ are two number functions in $O(poly \log n)$.
   This example shows how multi iteration crossing, nesting, synchronizing and interdepending may be implemented with semantic conditions.
7. ($MIX$ **counting**) $x = Y[\mathbf{c}]$, where $Y$ is a match-all nonterminal and $\mathbf{c}$ requires that the total numbers of occurrences of characters $a, b$, and $c$ respectively be equal.
   This pattern is known as $MIX$ in a context of demonstrating the power of $RCG$. It was unclear but suspected that $MIX$ is outside of indexed grammar and Linear Context-Free Rewriting Languages [9], p.10, p.162. So this example helps show how huge differences in complexity may result in the implementation of certain patterns between with and without allowance of using semantic conditions.
8. (**resource bounded randomness**) $x = Y[\mathbf{c}]$ is a match-all nonterminal and $\mathbf{c}$ requires that any string that matches $Y$ be $R$ random where $R$ is a computational resource bound. (See [11] for resource bounded complexity.)
   This example helps show several things: first, complexity pertinent constraints are allowed to serve as semantic conditions; second, the complexity of $UCFX$ can be above any recursive bound.

In view of the above, we emphasize that there are abundant scenarios in which the complexities of semantic conditions involved in the above examples except the last one are very low, say in pseudo square time or even lower.

**Meta syntax stipulations** The integration of semantic condition in $UCFX$ incur certain stipulations, or say meta syntactic rules. These rules are mainly for promoting simplicity, and reducing ambiguity and misunderstanding, all without loss of generality. We specify those stipulation rules below, some with explanations:

1. each kernel rule has at most one semantic condition. (if there are more, they can be disjuncted into one)
2. if a nonterminal has multiple kernel rules, each one may have its own semantic condition; and all semantic conditions associated with the same nonterminal may or may not be different.
3. two different kernel rules of the same or distinct nonterminals may have the same or different semantic conditions. in other words, the same semantic condition may be imposed on multiple kernel rules, of the same or different nonterminals.
4. it is allowed but not to be abused that the same semantic condition may assume different names, as associated with different kernel rules; by default,

different semantic condition names mean different semantic conditions, with exceptions allowed.

Note that if any semantic condition is ever used in an expression, its symbolic notation will be treated as an ad hoc syntactic element for the entire pattern expression and the system, as if it is a built-in function integrated within a programming language and gets used in a program.

**Specification and implementation issues** With reference to any rule $r : N \to x[\mathbf{c}]$ and a string $u$ to match rule $r$, the ultimate input of semantic condition $\mathbf{c}$ is the very string $u$ that matches $x$ as rule body. But depending on concrete situations, the salient values of the input, so to speak, may be just one or more derivable attribute values of $u$, such as length of $u$, iteration numbers of component iterations of $x$, a prefix, suffix, or infix of $u$ with a specific qualification, one or more positioned substrings of $u$, etc..

In case $x = x_1 \dots x_k$ for some $k > 1$ is a concatenation of certain indicated component patterns, if string $u$ matches $x$, then $u = u_1 \dots u_k$ is a concatenation of certain indicated substrings in due order, so it is required that $u_i$ match $x_i$ for all $1 \leq i \leq k$. In this scenario, some of the salient attribute values of $u$ for $\mathbf{c}$ can be *viewed as* being derived directly from a subset of these indicated substrings of $u$ rather than from $u$ directly.

Although the set of semantic conditions for each $UCFX$ pattern is finite, the variety of potential semantic conditions under any nontrivial complexity bound is infinite. In order to have a descent set of semantic conditions (or their prototypes) to implement for practical purposes, it is desirable and beneficial to prepare growing sets or say libraries of a. instrumental functions for derivable attribute values, b. internal data types and structures to hold intermediate derivable values and support intermediate computing tasks, and c. index mechanisms for performance reasons. It is reasonable to anticipate that such a preparation process will be on going indefinitely. So a distinction between the endeavor of $CFX$ computing and that of $UCFX$ computing is that the latter has to maintain and extend a library for semantic condition computing on needs.

## 2.3 *UCFX* match and production sequence

From now on, we will equate the notions of production and match, and conduct discussion mainly in terms of match. The definition of $UCFX$ match is below:

**Definition 22** *Given $X = \langle \mathcal{T}, \mathcal{N}, \mathcal{C}, \mathcal{R}, S \rangle$ in UCFX, any nonterminal $N$ in $X$, any rule $r : N \to x[\mathbf{c}]$ of $X$, and any string $u$,*

1. *$u$ matches $N$ by rule $r$ if $x$ is terminal and $x = u$, or the two conditions below both hold*
   **syntactic** *for some $k > 0$, $x = x_1 \dots x_k$, $u = u_1 \dots u_k$, and $u_i$ matches $x_i$ for all $1 \leq i \leq k$.*
   **semantic** *$\mathbf{c}$ is empty, or $\mathbf{c}(u) = 1$ (say $u$ matches $\mathbf{c}$ or $\mathbf{c}(u)$ holds).*

*2. $u$ matches $N$ if $u$ matches $N$ by at least one rule of $N$, notation $u \in L(N)$; if $u$ matches $N$ and $N = S = S_X$, $u$ is said in the language of $X$, or simply $u$ is in (pattern) $X$, notation $u \in L(X)$.*

The above definition is for a natural extension of $CFX$ match to $UCFX$ match. The notion of production sequence for $UCFX$ can also obtain.

Largely, for any $X$ in $UCFX$ and any string $u$, there is a production sequence $PS$ of $u$ from $X$ if and only $PS$ meets the following conditions:

1. Syntactic: $PS$ is $PS : y_0, y_1, \ldots, y_k$ for some $k \geq 0$ such that (start) $y_0 = S_X = X$, (end) $y_k = u$, (production) for each $0 \leq i < k$, there are $y_L, y_R$ and a kernel rule say $kr_i : N \to x$ of $X$ such that $y_i = y_L N y_R$ (the position of $N$ in $y_i$ is referred to as the production position of $y_i$), $y_{i+1} = y_L x y_R$.
2. Semantic: From $PS$, there is a unique kernel rule sequence $KS : kr_0, \ldots, kr_{k-1}$ such that $kr_i$ is used to produce $y_{i+1}$ from $y_i$; the nonterminal at the production position $p$ of $y_i$ maps to a unique positioned substring say $v = v_{i,p}$ of $u$; and if $kr_i$ has nonempty semantic condition $\mathbf{c}$, then $\mathbf{c}(v) = 1$.

So much for this (in view of space).

## 2.4 *ECFX* and its inner hierarchy

We have introduced the ideas of $CFTIME$ and $ECFTIME$, unambiguous $CFX$, and deterministic $CFX$ based on $CFX$ syntax and complexity. We can now use them to define $ECFX$ and its inner hierarch.

**Definition 23** *For any pattern $X$ in $UCFX$,*

*$X$ is in $ECFX$ if and only if all semantic conditions of $X$ are in $CFTIME$;*
*$X$ is in unambiguous $ECFX$ or $ECFX[n^2]$ if and only if all semantic conditions of $X$ are in the minimum of $CFTIME$ and $O(n^2)$;*
*$X$ is in deterministic $ECFX$ or $ECFX[n]$ if and only if all semantic conditions of $X$ are in $O(n)$.*

From the above, we get an $ECFX$ hierarchy

$$ECFX[n] \subseteq ECFX[n^2] \subseteq ECFX \subset UCFX$$

As to why the first two relations are non-proper, note that $CFTIME$ is not proven to be above $O(n^2)$ or even above $O(n)$. Pseudo versions of $ECFX[n]$, $ECFX[n^2]$ defined by adding a log or polylog factor to the principle complexity functions as used can also be defined and may be of interest. But we stop here.

For the relation between $CFTIME$ and $ECFTIME$, we have

**Lemma 21** *$CFTIME = ECFTIME$.*

*Proof.* (ideas only) All $CFX$ patterns can be used as semantic conditions, so $CFTIME \leq ECFTIME$. But if $CFTIME < ECFTIME$, $ECFX$ would not be in $CFTIME$. □

For complexity completeness of $ECFX$, the formal claim is below:

**Lemma 22** *$ECFX$ is $CFTIME$ complete.*

*Proof.* (ideas only) It is almost by definition. For any pattern $X$ in $CFTIME$, if $X$ is not already in $X$, $X$ can be defined in $ECFX$ by using a match-all rule with a semantic condition equivalent to $X$. □

Of course we can have similar results regarding the proper time complexities of $CFX[n^2], ECFX[n^2], CFX[n]$ and $ECFX[n]$, and their pseudo versions.

### 2.5 Cumulative match and cumulative time

For patterns in $CFX$ and hence in $ECFX$, it is interesting and may be important to expand on the cumulative (or say online) nature of $CFTIME$. Largely, if $u$ is a substring of $v$, which is in $X$ of $ECFX$, then $u$ is expected to be recognized as such in time $CFTIME$ at size $|u|$.rather than $|v|$ We formalize this idea below:

**Definition 24** *For any pattern $X$ and string $u$, $u$ is said to* infix match or cumulatively match *$X$, $u$ $C$-match $X$ for short, if there are $u_L, u_R$ (either or both being possibly empty) such that $v = u_L u u_R$ matches $X$.*

*For any complexity function $f$ and any string pattern $X$, $X$ is said to be cumulatively $f$ computable, notation $X \in CumO(f)$, if for any string $u$, whether $u$ $C$-matches $X$ is computable in $f(|u|)$. In particular, $X$ is said to be cumulatively $CFTIME$ computable, or say in* **c***$CFTIME$ for short, if for any string $u$, whether $u$ $C$-matches $X$ can be computed in $CFTIME$ at $|u|$.*

The notion of $C$-match subsumes prefix match and suffix match as may be properly interpreted. Informally, **c**$CFTIME$ behaves like a kind of online processing complexity, not necessarily linear online, but $CFTIME$ online. Note that **c**$CFTIME$ may be stringer than $CFTIME$ as a complexity constraint, as some string patterns in $CFTIME$ might not be in **c**$CFTIME$; but not vice versa.

For motivation of this concept, we add that it is very useful and relatively easy to prove results of the format: if $X$, etc., are in **c**$CFTIME$ (rather than $CFTIME$), then so is $Y$. To see roles of **c**$CFTIME$, we first introduce the following:

**Definition 25** *A string pattern $X$ is said to be* prefix/suffix/infix-saturated *if for any string $u$, if $u$ is in $X$, then any prefix/suffix/infix $v$ of $u$ is in $X$. For any string pattern $X$, the* prefix/suffix/infix-*saturated pattern of $X$, notation $Y = PREFIX(X)/SUFFIX(X)/INFIX(X)$, is such that for any $u$, $u$ matches $Y$ if and only if $u$ is a prefix/suffix/infix of some $v$ that matches $X$.*

With the above definition, each of $PREFIX, SUFFIX, INFIX$ is a pattern operation, namely, a function from patterns to patterns. That is, among others, for example, $PREFIX(X)$ is unique for any legitimate pattern $X$. The following are obvious:

**Lemma 23** *For any pattern $X$, in the sense of language subsumption,*

$X \subseteq PREFIX(X) = PREFIX(PREFIX(X))$;
$X \subseteq SUFFIX(X) = SUFFIX(SUFFIX(X))$;
$X \subseteq INFIX(X) = SUFFIX(PREFIX(X)) = PREFIX(SUFFIX(X))$.

Finally, we have the following regarding $CFX$ and $\mathbf{c}CFTIME$:

**Lemma 24** *For any pattern $X$ in $CFX$, $X$ is in $\mathbf{c}CFTIME$.*

*Proof.* By definition, we need only show that if $X$ is in $CFX$, then $Y = INFIX(X)$ is in $CFX$. From [8] p. 282, a full trio is closed on $PREFIX$, and hence $SUFFIX$ and $INFIX$; and $CFX$ is a full trio, so $X$ in $CFX$ implies $Y = INFIX(X)$ in $CFX$. That means $X$ is in $\mathbf{c}CFTIME$. $\square$

For $ECFX$, we will show later elsewhere that $ECFX$ is also a full trio, and hence $ECFX$ is in $\mathbf{c}CFTIME$. That result would equate $CFTIME$ and $\mathbf{c}CFTIME$. We expect that result helps our further exploration of $ECFX$.

## 3 Summary and next plan

We introduced many new concepts and briefly explored their relations between themselves and with existing ones. We reviewed related work for a better historical background of our main ideas. We raised some open questions along the way, including one concerning understanding of context freeness. We suggested evaluation perspectives and made analysis to justify our innovation efforts. We also made certain technical results near the end of our paper.

For next plan, we try to present our work on the following points: $ECFX$ is closed on many important and powerful operations, including those that ensures the trio status of $ECFX$; $ECFX$ is in $O(n^3)$.

## References

1. Alfred, V.: Algorithms for finding patterns in strings. Algorithms and Complexity **1**, 255 (2014)
2. Boullier, P.: A generalization of mildly context-sensitive formalisms. In: Proceedings of the fourth international workshop on Tree Adjoining Grammars and Related Frameworks (TAG+ 4). pp. 17–20 (1998)
3. Boullier, P.: A cubic time extension of context-free grammars. Grammars **3**(2-3), 111–131 (2000)
4. Chomsky, N.: Three models for the description of language. IRE Transactions on information theory **2**(3), 113–124 (1956)
5. Earley, J.: An efficient context-free parsing algorithm. Commun. ACM **13**(2), 94–102 (Feb 1970). https://doi.org/10.1145/362007.362035, `http://doi.acm.org/10.1145/362007.362035`
6. Farber, D.J., Griswold, R.E., Polonsky, I.P.: Snobol, a string manipulation language. Journal of the ACM (JACM) **11**(1), 21–30 (1964)
7. Freydenberger, D.D.: Extended regular expressions: Succinctness and decidability. Theory of Computing Systems **53**(2), 159–193 (2013)
8. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, vol. 2. Addison-wesley Reading, MA (1979)
9. Kallmeyer, L.: Parsing beyond context-free grammars. Springer Science & Business Media (2010)
10. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th international symposium on symbolic and algebraic computation. pp. 296–303. ACM (2014)
11. Li, M., Vitányi, P., et al.: An introduction to Kolmogorov complexity and its applications, vol. 3. Springer (2008)
12. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics **6**(4), 519–535 (2001)
13. Okhotin, A.: Boolean grammars. Information and Computation **194**(1), 19–48 (2004)
14. Valiant, L.G.: General context-free recognition in less than cubic time. Journal of computer and system sciences **10**(2), 308–315 (1975)
15. Weir, D.J.: Characterizing mildly context-sensitive grammar formalisms (1988)
16. Wikipedia contributors: Generalized context-free grammar — Wikipedia, the free encyclopedia (2019), `https://en.wikipedia.org/w/index.php?title=Generalized_context-free_grammar&oldid=918417045`, [Online; accessed 18-December-2020]
17. Wikipedia contributors: Mildly context-sensitive grammar formalism — Wikipedia, the free encyclopedia (2019), `https://en.wikipedia.org/w/index.php?title=Mildly_context-sensitive_grammar_formalism&oldid=912302365`, [Online; accessed 12-August-2020]