Decision problem of substrings in Context Free Languages.

Mauricio Osorio, Juan Antonio Navarro

Abstract

A context free grammar (CFG) is a set of symbols and productions used to define a context free language. We present an algorithm which, given a CFG and a string α, decides whether a string of the form $\beta\alpha\gamma$ belongs to the language or not. There is a wide variety of applications where deciding if a string belongs to the defined language has relevance but, due to "visualization" limitations, it is not always possible to explicitly have complete strings of data. Our proposed solution involves a set of equations able to compute the sets of non-terminal symbols that can produce every substring. A proof of the correctness of those equations is detailed in the paper. We designed this algorithm with dynamic programming methods, and it can solve the problem in polynomial time.

Keywords: Contex free grammar, parsing

1 Introduction

Given a context free grammar (CFG) we present an algorithm to decide, for a given string α, whether a string of the form $\beta\alpha\gamma$ belongs to the language generated by this grammar. In other words we want to find out if α belongs to the set of *infix terms* in the language.

Our algorithm is a generalization of the well known Cocke-Kasami-Younger [1], which only determines belong ness for exact strings. It is designed following the *dynamic programming* technique and can solve the problem, as will be shown later, on polynomial time. Its worst-case order can be expressed as $n^3p + n^2p^2$, where n is the length of the string α and p the size of the grammar (amount of productions). If we consider the size of the grammar to be a fixed constant then the complexity of the algorithm is $O(n^3)$, just as the best available algorithms for the exact strings problem.

There are known several applications of CFG to model real-life situations, and where deciding if a string belongs to the defined language has relevance. Interesting examples can be found in the analysis of DNA sequences [2], treating grammars as contracts [3] and in shape analysis [4], just to mention some of them.

A common issue in such applications is that, due to "visualization" limitations, it is not always possible to explicitly have complete strings of data. The ideas discussed on this paper introduce the ability to study grammars under these circumstances, where the target string is only partially known. According to [5] this problem shows "interesting connections with the general theory of partial evaluation of programs, which deals with the specialization of programs by propagation of known properties of their input."

This problem has been studied, using a different approach, by Bernard Lang [6]. He claims to have a cubic order algorithm, relative to the length of the input string just as ours, but he does not specify the influence of the size of the grammar. Also his article [6] contains no proof for the correctness of the algorithm, nor its complexity. Therefore we think our work is not only more complete, but offers a more convenient way to solve the problem.

There is also a work from Martin Ruckert [7] who offers ideas for generating substring parsers, but he limited his research into the so called Bounded Right Context Grammars. BRC Grammars are a restricted set of Grammars which is a slight generalization of LR parseable grammars. Our work on this paper is intended to deal with much more general Grammars.

2 Notation

For simplicity, we will assume that the grammar is given in the Chomsky Normal Form (CNF). Let $G = (V, T, P, S)$ be that grammar, where V is the set of non-terminal symbols, T the set of terminal symbols, P the productions and $S \in V$ the start symbol. Remember that CNF assumes that every production is either on the form $A \to BC$ or $A \to a$, for symbols $A, B, C \in V$ and $a \in T$.

For a string $\alpha = X_1 X_2 \dots X_n$ $(X_i \in V \cup T)$ the number n is the length of the string and is denoted by $|\alpha|$. By definition ϵ is the empty string and $|\epsilon| = 0$. Also, if $1 \leq j \leq k \leq |\alpha|$, it is defined α_i^j as the substring $X_i X_{i+1} \dots X_j$ from α. If $i > j$ then $\alpha_i^j \overset{\text{def}}{=} \epsilon$. The limits i, j can be omitted so that $\alpha^j = \alpha_1^j$ and $\alpha_i = \alpha_i^{|\alpha|}$.

For $A \in V$ and a string α, the notation $A \to \alpha$ denotes that the pair (A, α) is in P. If the string β can be derived from α with a finite sequence (possibly empty) of productions then we write $\alpha \Rightarrow \beta$.

Given $\mathcal{A}, \mathcal{B}$ subsets of $(V \cup T)^*$ and $\mathbf{A}, \mathbf{B}, \mathbf{C}$ subsets of V we define

$$\begin{array}{rcl} \mathcal{A}\mathcal{B} & \overset{\text{def}}{=} & \{\alpha\beta \mid \alpha \in \mathcal{A}, \beta \in \mathcal{B}\}. \\ \mathbf{A} \to \mathbf{BC} & \text{iff} & A \to BC \text{ for some } A \in \mathbf{A},\ B \in \mathbf{B} \text{ and } C \in \mathbf{C}. \\ \mathcal{A} \Rightarrow \mathcal{B} & \text{iff} & \alpha \Rightarrow \beta \text{ for some } \alpha \in \mathcal{A} \text{ and } \beta \in \mathcal{B}. \end{array}$$

As an abuse of the notation above we may write for a string α and a set of strings $\mathcal{B}$ their concatenation as $\alpha\mathcal{B}$ which actually stands for $\{\alpha\}\mathcal{B}$.

3 Definitions

Definition. The *generator set* of a string α, denoted by $[\alpha]$, is the subset from V defined as

$$[\alpha] \overset{\text{def}}{=} \{X \in V \mid X \Rightarrow \alpha\}.$$

Also, if $\mathcal{A}$ is a set of strings we extend the notation to

$$[\mathcal{A}] = \bigcup_{\alpha \in \mathcal{A}} [\alpha].$$

Notes. From the definition above note that both affirmations $X \Rightarrow \alpha$ and $X \in [\alpha]$ are equivalent. Some particular generator sets can be easily calculated, the ones used more often in this paper are resumed in the following lemma.

Lemma 1 *Let a be a terminal, A a non-terminal and $\mathbf{A}$ a set of non-terminals. Their generator sets are:*

$$[a] = \{X \in V \mid X \rightarrow a\} \qquad [A] = \{A\} \qquad [\mathbf{A}] = \mathbf{A}.$$

The first two are true because of the language in CNF, remember it is not possible to derive with more than zero productions one single non-terminal. The third one is an immediate consequence of the second.

Definition. Given a set of non-terminal symbols $\mathbf{A}$ the *prefix set* of $\mathbf{A}$ is the function defined from V to V as

$$\mathsf{pref}(\mathbf{A}) \overset{\text{def}}{=} \mathbf{A} \cup [\mathbf{A}V].$$

We also define the n-th iteration of the *prefix set* as

$$\mathsf{pref}^0(\mathbf{A}) \overset{\text{def}}{=} \mathbf{A},$$

$$\mathsf{pref}^n(\mathbf{A}) \overset{\text{def}}{=} \mathsf{pref}(\mathsf{pref}^{n-1}(\mathbf{A})).$$

Notes. It can be easily verified that the function is monotone (i.e. $n < m$ implies $\mathsf{pref}^n(\mathbf{A}) \subset \mathsf{pref}^m(\mathbf{A})$). Then, because V is finite, there exists a number n called the *iteration limit* such that $\mathsf{pref}^n(\mathbf{A}) = \mathsf{pref}^{n+k}(\mathbf{A})$ for all numbers k.

Definition. Given a set of non-terminal symbols $\mathbf{A}$, let n be the iteration limit of $\mathsf{pref}(\mathbf{A})$. Then the *first set* of $\mathbf{A}$ is the function defined as

$$\mathsf{first}(\mathbf{A}) \overset{\text{def}}{=} \mathsf{pref}^n(\mathbf{A}).$$

Definition. Analogously defined are the *suffix* and *infix* sets of $\mathbf{A}$ as the functions

$$\begin{aligned}\mathsf{suff}(\mathbf{A}) &\overset{\text{def}}{=} \mathbf{A} \cup [V\mathbf{A}],\\ \mathsf{inf}(\mathbf{A}) &\overset{\text{def}}{=} \mathbf{A} \cup [V\mathbf{A}] \cup [\mathbf{A}V]\end{aligned}$$

respectively.

And, letting n be the iteration limit for $\mathsf{suff}(\mathbf{A})$ and m the iteration limit for $\mathsf{inf}(\mathbf{A})$ the *last* and *middle* sets are defined to be

$$\begin{aligned}\mathsf{last}(\mathbf{A}) &\overset{\text{def}}{=} \mathsf{suff}^n(\mathbf{A}),\\ \mathsf{mid}(\mathbf{A}) &\overset{\text{def}}{=} \mathsf{inf}^m(\mathbf{A}).\end{aligned}$$

Finally there is a simple *replacing* lemma which will be used often in the proof of the algorithm. It summarizes some operations that can be applied to sets of non-terminals preserving the set contentions.

Lemma 2 *Let $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ be sets of non-terminal symbols such that $\mathbf{A} \subset \mathbf{B}$ and $\mathbf{C} \subset \mathbf{D}$. Then the following properties hold.*

1. $\mathbf{AC} \subset \mathbf{BD}$.
2. $[\mathbf{A}] \subset [\mathbf{B}]$.
3. $\mathsf{first}(\mathbf{A}) \subset \mathsf{first}(\mathbf{B})$
4. $\mathsf{last}(\mathbf{A}) \subset \mathsf{last}(\mathbf{B})$

4 Algorithm

Our solution to the problem consists on a set of equations able to compute, given a string of terminals α (with $|\alpha| > 1$), the generator sets for all strings containing α as a prefix, suffix, or infix.

$$[\alpha T^*] = \bigcup_{i=1}^{|\alpha|-1} \mathsf{first}([[\alpha^i][\alpha_{i+1}T^*]]) \tag{1}$$

$$[T^*\alpha] = \bigcup_{i=1}^{|\alpha|-1} \mathsf{last}([[T^*\alpha^i][\alpha_{i+1}]]) \tag{2}$$

$$[T^*\alpha T^*] = \bigcup_{i=0}^{|\alpha|} \mathsf{mid}([[T^*\alpha^i][\alpha_{i+1}T^*]]) \tag{3}$$

The intuitive idea behind those equations is to split the string α into substrings, calculate their prefix and suffix sets and finally join this information together.

For strings of length 1, we also need the following two formulas

$$[\alpha T^*] = \mathsf{first}([\alpha]), \qquad [T^*\alpha] = \mathsf{last}([\alpha]). \tag{4}$$

Equation (3) works for strings with length 1 as well.

This set of equations will be proved shortly, first an example of their use is given.

5 Example

In this section it will be shown an explicit example of calculating generator sets. Lets consider the grammar

$$E \to E\texttt{+}E \mid E\texttt{*}E \mid \texttt{(}E\texttt{)} \mid \texttt{x}$$

which defines all valid arithmetical expressions involving sums, products and parenthesis with the variable `x`.

We choose this grammar because of its natural matter, it will allow us to easily construct positive and negative examples. The proposed equations will be applied to determine if the string $\alpha =$ "`x ) + x * x`" can be a prefix, suffix or infix in the language.

First lets turn this grammar into its associated CNF so we have

$$E \to ET \mid PR \mid \texttt{x} \qquad T \to ME \mid CE \qquad R \to EQ$$

$$M \to \texttt{+} \qquad C \to \texttt{*} \qquad P \to \texttt{(} \qquad Q \to \texttt{)}.$$

Using the CKY Algorithm we can calculate generator sets for each substring of α. The result of applying this algorithm is an array M whose ij entry corresponds to the generator set of the substring α_i^j.

x	)	+	x	*	x	
E						x
R	Q					)
$\emptyset$	$\emptyset$	M				+
$\emptyset$	$\emptyset$	T	E			x
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	C		*
$\emptyset$	$\emptyset$	T	E	T	E	x

The table shows that no symbol can exactly derive the string α. As we could expect the string α does not belongs to the language, because particularly E can not derive α.

Now we will apply equations (4) and (2) to see if α belongs to the suffixes of the language. First we have to calculate

$$[T^*\alpha^1] = [T^*\texttt{x}] = \mathsf{last}([\texttt{x}]).$$

The set $[\mathtt{x}]$ is equal to $\{E\}$, then we calculate the last of $\{E\}$ iterating the definition of suff.

$$\begin{aligned}\mathsf{suff}(\{E\}) &= \{E\} \cup [VE] = \{E\} \cup \{T\} = \{E,T\}.\\ \mathsf{suff}(\{E,T\}) &= \{E,T\} \cup [V\{E,T\}] = \{E,T\} \cup \{E,T\} = \{E,T\}.\end{aligned}$$

Because of the small language we are dealing here we end very soon, just at the second iteration.

Then according to equation (2) we can proceed

$$[T^*\alpha^2] = \mathsf{last}([[T^*\alpha^1][\alpha_2^2]]) = \mathsf{last}([[T^*\mathtt{x}][\mathtt{)}]]),$$

using the information collected in the matrix M and the previous step we have

$$\mathsf{last}([[T^*\mathtt{x}][\mathtt{)}]]) = \mathsf{last}([\{E,T\}\{Q\}]) = \mathsf{last}(\{R\}) = \{E,T,R\}.$$

The algorithm continues calculating $[T^*\alpha^3]$ as

$$\begin{aligned}[T^*\mathtt{x)+}] &= \mathsf{last}([[T^*\mathtt{x}][\mathtt{)+}]]) \cup \mathsf{last}([[T^*\mathtt{x)}][\mathtt{+}]])\\ &= \mathsf{last}([\{E,T\}\emptyset]) \cup \mathsf{last}([\{E,T,R\}\{M\}])\\ &= \mathsf{last}(\emptyset) \cup \mathsf{last}(\emptyset) = \emptyset\end{aligned}$$

So it turns out that $[T^*\mathtt{x)+}] = \emptyset$, which is expected since no string in the language can end with a single `x` character.

The result of repeating the algorithm for all substrings of α is summarized on the table below.

$[T^*\mathtt{x}]$	$[T^*\mathtt{x)}]$	$[T^*\mathtt{x)+}]$	$[T^*\mathtt{x)+x}]$	$[T^*\mathtt{x)+x*}]$	$[T^*\mathtt{x)+x*x}]$
$\{E,T\}$	$\{E,T,R\}$	$\emptyset$	$\{E,T\}$	$\emptyset$	$\{E,T\}$

This way we are able to conclude, after the proof of the equations involved, that E can derive the strings containing "`x ) + x * x`" as a suffix. Which is of course true, since prepending a "`(`" or a "`( x +`" to α we a get strings of the language.

Applying this same procedure, but scanning the string in the opposite order, we can calculate generator sets of prefixes.

$[\mathtt{x)+x*x}T^*]$	$[\mathtt{)+x*x}T^*]$	$[\mathtt{+x*x}T^*]$	$[\mathtt{x*x}T^*]$	$[\mathtt{*x}T^*]$	$[\mathtt{x}T^*]$
$\emptyset$	$\emptyset$	$\{T\}$	$\{E,R\}$	$\{T\}$	$\{E,R\}$

As expected the result shows that no string in the language can begin with "`x ) + x * x`" . Finally mixing the information for suffixes and prefixes it can be calculated the generator set for infixes $[T^*xT^*]$ as

$$\mathsf{mid}([[T^*][\mathtt{x)+x*x}T^*]]) \cup \mathsf{mid}([[T^*\mathtt{x}][\mathtt{)+x*x}T^*]]) \cup \cdots \cup \mathsf{mid}([[T^*\mathtt{x)+x*x}][T^*]]),$$

which evaluates to $\{E,T,R\}$. This is also clear since all suffix strings are also infix ones.

6 Pseudocode

The algorithm described on the previous section is now presented in a more formal way. The main code includes the calculation of the matrix M using the CKY algorithm and our proposed generalization for suffixes, prefixes, and infixes. Here α is the input string of size n, M is an $n \times n$ matrix, S and P are vectors of size n.

```
/* CKY Algorithm */
for $i \leftarrow 1$ to $n$ do $M_{i,i} \leftarrow [\alpha_i^i]$

for $l \leftarrow 2$ to $n$ do
    for $i \leftarrow 1$ to $n - l + 1$ do {
        $M_{i,i+l-1} \leftarrow \emptyset$;
        for $j \leftarrow i$ to $i + l - 2$ do
            $M_{i,i+l-1} \leftarrow M_{i,i+l-1} \cup [M_{i,j} M_{j+1,i+l-1}]$;
    }

/* Generalization */
$P_1 \leftarrow \mathsf{last}(M_{1,1})$;
$S_1 \leftarrow \mathsf{first}(M_{n,n})$;
for $i \leftarrow 2$ to $n$ do {
    $P_i \leftarrow \emptyset$;
    $S_i \leftarrow \emptyset$;
    for $j \leftarrow 1$ to $i - 1$ do {
        $P_i \leftarrow P_i \cup \mathsf{last}([P_j M_{j+1,i}])$;
        $S_i \leftarrow S_i \cup \mathsf{first}([M_{n-i-1,n-j} S_j])$;
    }
}

$I \leftarrow \mathsf{mid}([P_n V]) \cup \mathsf{mid}([V S_n])$;
for $i \leftarrow 1$ to $n - 1$ do
    $I \leftarrow I \cup \mathsf{mid}([P_i S_{n-i}])$;
```

We also need to show how to calculate generator sets, last, first and mid functions used above. A simple function to calculate, given $\mathbf{X}, \mathbf{Y} \subset V$, the generator set $[\mathbf{XY}]$ is described below. Notice that, if p is the number of productions of G, its worst case order is only p.

```
$R \leftarrow \emptyset$;
for $A \rightarrow BC$ a production of $G$ do
    if $B \in \mathbf{X}$ and $C \in \mathbf{Y}$
        then $R \leftarrow R \cup \{A\}$;
return $R$;
```

The code to calculate the first, last and mid functions is very similar. Let n be the length of the string x, then the order of this algorithm is np in its worst case when only one single symbol is added to the result on each iteration. An implementation of first is shown where $\mathbf{X}$ is its input set.

```
$P \leftarrow \emptyset$;
$R \leftarrow \mathbf{X}$;
while $(R \neq P)$ do {
    $P \leftarrow R$;
    $R \leftarrow R \cup [RV]$;
}
return $R$;
```

As can be easily determined on the main code the order of the CKY part of the algorithm is n^3p, while the order of the appended new code is n^2p^2. A remarkable fact is that the problem is completely solved in polynomial time. Also, if the grammar is thought to be fixed, then p is a constant and the complexity of the algorithm is $\mathrm{O}(n^3)$. We have to remember anyway that this is only an upper bound of the algorithm complexity in its worst case, working on average cases its efficiency might be improved.

7 Proof

Now we will prove the correctness of the equations involved. First, for the pair of equations (4) a more general fact is proven. Note that lemmas 1 and 2 are used constantly between set contentions.

Proposition 1 *Given a set of symbols* $\mathbf{X} \in (V \cup T)$,

$$[\mathbf{X}T^*] = \mathsf{first}([\mathbf{X}]), \qquad [T^*\mathbf{X}] = \mathsf{last}([\mathbf{X}]).$$

Proof. To prove $[\mathbf{X}T^*] \subset \mathsf{first}([\mathbf{X}])$. Let A be an arbitrary element in $[\mathbf{X}T^*]$, then $A \Rightarrow \mathbf{X}t$ for some string of terminals t. By induction over $|t|$ it will be proved that also $A \in \mathsf{first}([\mathbf{X}])$. For $|t| = 0$ we have $t = \epsilon$, then $A \Rightarrow \mathbf{X}$ and $A \in [\mathbf{X}] \subset \mathsf{first}([\mathbf{X}])$.

Lets suppose the affirmation is true for all $m < |t|$ then we prove for $|t|$. Because $A \Rightarrow \mathbf{X}t$ there exist $B, C \in V$ such that $A \rightarrow BC$, $B \Rightarrow \mathbf{X}t^i$ and $C \Rightarrow t_{i+1}$ for some i $(0 \leq i < |t|)$. Using the induction hypothesis $B \in \mathsf{first}([\mathbf{X}])$, because $A \rightarrow BC$ and $BC \subset \mathsf{first}([\mathbf{X}])V$ (lemma 2), then $A \in \mathsf{pref}(\mathsf{first}([\mathbf{X}])) = \mathsf{first}([\mathbf{X}])$.

To prove $[\mathbf{X}T^*] \supset \mathsf{first}([\mathbf{X}])$. Now let A be an arbitrary element in $\mathsf{first}([\mathbf{X}])$, so that $A \in \mathsf{pref}^n([\mathbf{X}])$ for some number n. Again we prove by induction over n that if $A \in \mathsf{pref}^n([\mathbf{X}])$ then $A \in [\mathbf{X}T^*]$. If $n = 0$ then $A \in \mathsf{pref}^0([\mathbf{X}]) = [\mathbf{X}] = [\mathbf{X}\epsilon] \subset [\mathbf{X}T^*]$, the last contention using lemma 2.

Lets suppose the proposition is valid for $n-1$ then we prove for n. If $A \in \mathsf{pref}^n([\mathbf{X}]) = \mathsf{pref}(\mathsf{pref}^{n-1}([\mathbf{X}]))$ then, because of the definition of pref, $A \in \mathsf{pref}^{n-1}([\mathbf{X}])$ or $A \in [\mathsf{pref}^{n-1}([\mathbf{X}])V]$. In the first case the proof is finished because of induction. Otherwise there exist some B and C such that $B \in \mathsf{pref}^{n-1}([\mathbf{X}])$, $C \in V$ and $A \to BC$. Using the inductive hypothesis $B \in [\mathbf{X}T^*]$, so $B \Rightarrow \mathbf{X}t$ for some $t \in T^*$. Then $A \Rightarrow \mathbf{X}tC \Rightarrow \mathbf{X}tu$ for some string u of non-terminals, and because $tu \in T^*$ we get $A \in [\mathbf{X}T^*]$.

The proof for $[T^*\mathbf{X}] = \mathsf{last}([\mathbf{X}])$ is entirely analogous. $\diamond$

Now we are able to prove equation (1), the proof for (2) is also analogous.

Proposition 2 *Given a string $x \in T^+$ with $|x| > 1$,*

$$[xT^*] = \bigcup_{i=1}^{|x|-1} \mathsf{first}([[x^i][x_{i+1}T^*]]).$$

Proof. To simplify notation let $P(x)$ be the right hand side of the expression above. To prove $[xT^*] \supset P(x)$. Let $A \in P(x)$, $A \in \mathsf{first}([[x^i][x_{i+1}T^*]])$ for some i $(1 \leq i < |x|)$. Using the result of proposition 1 $A \in [[[x^i][x_{i+1}T^*]]T^*]$, then $A \Rightarrow x^i x_{i+1}tu = xtu$ for some strings $t, u \in T^*$. But tu is also in T^* so $A \in [xT^*]$.

To prove $[xT^*] \subset P(x)$. On the other hand let $A \in [xT^*]$, then $A \Rightarrow xt$ for some sentence t. It will be proved by induction over $|t|$ that $A \in P(x)$. For $|t| = 0$ we have $t = \epsilon$, so $A \Rightarrow x$ and there exist $B, C \in V$ such that $A \to BC$, $B \Rightarrow x^j$ and $C \Rightarrow x_{j+1}$ for some j $(1 \leq j < |x|)$. Then, using again lemma 2, $A \in [BC] \subset [[x^j][x_{j+1}\epsilon]] \subset [[x^j][x_{j+1}T^*]] \subset \mathsf{first}([[x^j][x_{j+1}T^*]]) \subset P(x)$.

Lets suppose the proposition is true for all $m < |t|$, we prove for $|t|$. There exist $B, C \in V$ such that $A \to BC \Rightarrow xt$. There are two distinct cases shown separately.

1. $B \Rightarrow x^i, C \Rightarrow x_{i+1}t \qquad (1 \leq i < |x|)$.
   Then $A \in [BC] \subset [[x^i][x_{i+1}t]] \subset [[x^i][x_{i+1}T^*]] \subset P(x)$.
2. $B \Rightarrow xt^k, C \Rightarrow t_{k+1} \qquad (0 \leq k < |t|)$.
   By our inductive hypothesis $B \in P(x)$, then $B \in \mathsf{first}([[x^i][x_{i+1}T^*]])$ for some value of i $(1 \leq i < |x|)$. So we have $A \in [Bt_{k+1}] \subset [BT^*] = \mathsf{first}([B]) \subset \mathsf{first}(\mathsf{first}([[x^i][x_{i+1}T^*]])) = \mathsf{first}([[x^i][x_{i+1}T^*]]) \subset P(x)$.

This finishes the proof. $\diamond$

Finally we prove equation number (3). The proof is very similar to the preceding one, but with slight differences as shown below.

Proposition 3 *Given a string $x \in T^*$ we have,*

$$[T^*xT^*] = \bigcup_{i=0}^{|x|} \mathsf{mid}([[T^*x^i][x_{i+1}T^*]]).$$

Proof. Let $M(x)$ be the right hand side of the expression above. To prove $[T^*xT^*] \supset M(x)$. Let A be in $M(x)$ so $A \in \mathsf{mid}([[T^*x^i][x_{i+1}T^*]])$ for some i $(0 \leq i \leq |x|)$, so $A \in [T^*[[T^*x^i][x_{i+1}T^*]]T^*]$. Then there are strings $r, s, t, u \in T^*$ such that $A \Rightarrow rsx^ix_{i+1}tu = rsxtu$, and finally $A \in [T^*xT^*]$.

To prove $[T^*xT^*] \subset M(x)$. Suppose $A \in [T^*xT^*]$, then there are $t, u \in T^*$ such that $A \Rightarrow txu$. It is proven by induction over $|t| + |u|$ that $A \in M(x)$. If $|t| + |u| = 0$ then $t = u = \epsilon$ and $A \in [[\epsilon x][\epsilon]] \subset [[T^*x][T^*]] \subset M(x)$.

Supposing the affirmation is true for $m < |t| + |u|$, we prove for this case. Because $A \Rightarrow txu$ and $|txu| \geq 2$ there are $B, C \in V$ such that $A \to BC$. There are three cases shown separately.

1. $B \Rightarrow tx^i, C \Rightarrow x_{i+1}u \qquad (1 \leq i < |x|)$
   Then $A \in [BC] \subset [[tx^i][x_{i+1}u]] \subset [[T^*x^i][x_{i+1}T^*]] \subset M(x)$.

2. $B \Rightarrow t^k, C \Rightarrow t_{k+1}xu \qquad (0 < k \leq |t|)$
   Using the inductive hypothesis $C \in \mathsf{mid}([[T^*x^i][x_{i+1}T^*]])$ for some value of i $(1 \leq i < |x|)$. So $A \in [VC] \subset \mathsf{mid}(C) \subset \mathsf{mid}([[T^*x^i][x_{i+1}T^*]]) \subset M(x)$.

3. $B \Rightarrow txu^k, C \Rightarrow u_{k+1} \qquad (0 \leq k < |u|)$
   Analogous to the preceding.

The proof is concluded. $\diamond$

8 Conclusions and Results

We already have a working implementation of the algorithm in C++[1], which has solved several test cases correctly and quite fast. With simple grammars, like the one used as example on this paper, solutions are computed instantly.

By generating productions at random to increase the size of the grammar we were able to test our algorithm on more adverse circumstances. It took, in average, 0.32 seconds to solve the problem for a string of 75 characters length and 50 productions in the grammar; 3.64 seconds with 500 productions and about 44.37 seconds using 5'000 productions. Both negative and positive examples were used in the test.

Since B. Lang [6] offers no results of his proposed algorithm, nor an implementation, we are not able to properly compare their efficiency. Both algorithms have cubic complexity respect to the length of the string but, as we showed in this paper, the size of the grammar might be an important factor to consider. More tests should be done in order to shown which algorithm is more appropriate under different circumstances.

Our algorithm is still limited in some way since it assumes consecutive characters of the string are exactly known. We strongly believe that our approach

[1]Available at http://www.udlap.mx/ ma108907/substrings

to the problem can be generalized a little bit more to deal with less precise and broken strings, this may be shown in a future work.

This research is sponsored by the Mexican National Council of Science and Technology, CONACyT (project 35804-A).

References

[1] D.C. Kozen. *Automata and Computability.* Springer, 1997.

[2] K. Park and T.L. Huntsberger. Inference of context free grammars for syntactic analysis of dna sequences. In *Proceedings of the 1990 AAAI Spring Symposium on Artificial Intelligence and Molecular Biology*, pages 110–114, 1990.

[3] M. de Jonge and J. Visser. Grammars as contracts. *To appear 2000*, 2000.

[4] S. Loncaric. A survey of shape analysis techniques. In *Pattern Recognition*, volume 31, pages 983–1001, 1998.

[5] B. Lang. Towards a uniform formal framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*, pages 153–171. Kluwer Academic Publishers, 1991.

[6] B. Lang. Parsing incomplete sentences. In *Proc. 12th Int. Conf. on Computational Linguistics*, volume 1, pages 365–371, 1988.

[7] Martin Ruckert. Generating efficient substring parsers for brc grammars. Technical Report 98-105, State University of New York at New Paltz, 1998.