A Polynomial Time Extension of Parallel Multiple Context-Free Grammar

Peter Ljunglöf

Department of Computing Science,
Göteborg University and Chalmers University of Technology,
SE-412 96 Göteborg, SWEDEN,
peb@cs.chalmers.se

Abstract. It is already known that *parallel multiple context-free grammar* (PMCFG) [1] is an instance of the equivalent formalisms *simple literal movement grammar* (sLMG) [2,3] and *range concatenation grammar* (RCG) [4,5]. In this paper we show that by adding the single operation of intersection, borrowed from *conjunctive grammar* [6], PMCFG becomes equivalent to sLMG and RCG. As a corollary we get that PMCFG with intersection describe exactly the class of languages recognizable in polynomial time.

The layout of this paper is as follows. The first section contains definitions of the basic grammar formalisms we are interested in. The second section introduces the intersection operation for PMCFG. The third section contains the main result of the paper – that PMCFG extended with the intersection operation is equivalent to simple LMG and RCG. The fourth and last section is a small discussion of the results.

1 GCFG, PMCFG, sLMG and RCG

1.1 Generalized Context-Free Grammar

Generalized context-free grammar (GCFG) was introduced by Pollard in the 80's as a way of formally describing *head grammar* [7]. There are several definitions of GCFG in the literature; Seki et al [1] use a definition similar to Pollard's original, while others [8,9,10] more cleanly separates between abstract and concrete syntax. However, the latter definitions use the term GCFG for only the abstract part of the grammar, and the term *context-free rewriting system* for the abstract grammar together with the concrete interpretation function. While Pollard imposed no restriction on the concrete linearization type, other definitions restrict them to be tuples of strings. Here we use the definition from [11], which is close to the original definition.

Definition 1 (GCFG, abstract part). *The abstract grammar of a GCFG is a tuple* $(\mathcal{C}, S, \mathcal{F}, \mathcal{R})$*, where* $\mathcal{C}$ *and* $\mathcal{F}$ *are finite sets of categories and function*

symbols respectively, $S \in \mathcal{C}$ is the starting category, and $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{F} \times \mathcal{C}^*$ is a finite set of context-free syntax rules. For each function symbol $f \in \mathcal{F}$ there is an associated context-free syntax rule:

$$A \to f[B_1, \ldots, B_\delta]$$

The *arity* of the rule is δ, and in general we write δ_f for the arity of the rule f. The tree rewriting relation $t : A$ is defined as $f(t_1, \ldots, t_\delta) : A$ whenever $t_1 : B_1, \ldots, t_\delta : B_\delta$. We say that a tree t is *valid* (for a given category A) if $t : A$.

Example 1. The abstract grammar of a simple fragment of English might look like the following,

$$\begin{aligned}
\mathsf{S} &\to s_p[\mathsf{NP}, \mathsf{VP}] \\
\mathsf{S} &\to s_t[\mathsf{NP}, \mathsf{VP}] \\
\mathsf{VP} &\to vp[\mathsf{V}, \mathsf{NP}] \\
\mathsf{NP} &\to np[\mathsf{D}, \mathsf{N}] \\
\mathsf{D} &\to some[] \\
\mathsf{D} &\to most[] \\
\mathsf{N} &\to cat[] \\
\mathsf{NP} &\to fish[] \\
\mathsf{V} &\to eat[] \\
\mathsf{V} &\to catch[]
\end{aligned}$$

The idea is that the grammar should be able to handle both normal word order ('*most cats eat fish*'), and topicalized sentences ('*it is fish that most cats eat*').

Definition 2 (GCFG, concrete part). *To each category A is associated a* linearization type *A°, which is not further specified. To each function symbol f is associated a partial* linearization function *f°, taking as many arguments as the abstract syntax rule specifies:*

$$f^\circ \in B_1^\circ \times \cdots \times B_\delta^\circ \to A^\circ$$

The linearization $[\![\cdot]\!]$ of syntax trees is defined as,

$$[\![f(t_1, \ldots, t_\delta)]\!] = f^\circ([\![t_1]\!], \ldots, [\![t_\delta]\!])$$

if the application is defined. Note that the definition imposes no restrictions on the linearization types or the linearization functions; this is left to the actual grammar formalism. For our purposes it is enough to view a linearization type as the set of all its possible linearization values.

To be able to define the *language* of a grammar as a set of strings, we demand that the linearization type of the starting category is $S^\circ = \Sigma^*$. The language of a grammar G then becomes:

$$\mathcal{L}(G) = \{\, [\![t]\!] \mid t : S \,\}$$

1.2 Parallel Multiple Context-Free Grammar

Parallel multiple context-free grammar (PMCFG) [1,12] were introduced in the late 80's by Kasami, Seki et al. as a very expressive formalism, incorporating *linear context-free rewriting systems* and other mildly context-sensitive formalisms, but still with a polynomial parsing algorithm.

Definition 3 (PMCFG). *PMCFG is an instance of GCFG, with the following restrictions on linearizations:*

- *Linearization types are restricted to tuples of strings. In other words, each PMCFG grammar defines a* linearization arity $d(C)$ *for each category* C*. The linearization types can then be defined as* $C^\circ = (\Sigma^*)^{d(C)}$*.*
- *The only allowed operations in linearization functions are tuple projections and string concatenations. In other words, each PMCFG linearization function is of the form,*

$$f^\circ\left(\langle x_{1,1}, \ldots, x_{1,d_1}\rangle, \ldots, \langle x_{\delta,1}, \ldots, x_{\delta,d_\delta}\rangle\right) = \langle \alpha_1, \ldots, \alpha_d\rangle$$

  where each α_i *is a sequence of variables* $x_{j,k}$ *and constant strings.*

Example 2. The concrete syntax of the example English grammar might look like follows:

$$\begin{aligned}
s_p^\circ(x, \langle y_1, y_2\rangle) &= x\ y_1\ y_2\\
s_t^\circ(x, \langle y_1, y_2\rangle) &= \text{`}it\ \ is\text{'}\ y_2\ \text{`}that\text{'}\ x\ y_1\\
vp^\circ(x, y) &= \langle x, y\rangle\\
np^\circ(x, y) &= x\ y\\
most^\circ &= \text{`}most\text{'}\\
cat^\circ &= \text{`}cats\text{'}\\
fish^\circ &= \text{`}fish\text{'}\\
eat^\circ &= \text{`}eat\text{'}\\
catch^\circ &= \text{`}catch\text{'}
\end{aligned}$$

Note that verb phrases have to consist of two discontiuous phrases, for the topicalization to function.[1]

1.3 Subclasses of PMCFG

A PMCFG where each variable $x_{i,j}$ occurs in its linearization is called *nonerasing*. If no variable $x_{j,k}$ occurs twice in a linearization the grammar is called a *linear* MCFG (LMCFG or just MCFG). A nonerasing and linear grammar (i.e. if each variable occurs exactly once in its linearization), is called a *linear context-free rewriting system* (LCFRS). The following lemma states that LMCFG and LCFRS are equivalent formalisms [1]:

[1] If the example seems strange, there are other languages (such as German or Swedish) where discontinuous verb phrases are more natural.

Lemma 1. *Any PMCFG grammar can be converted into an equivalent nonerasing grammar. Furthermore, linearity is preserved by the conversion.*

1.4 Literal Movement Grammar and Range Concatenation Grammar

Literal movement grammar (LMG; [2,3]), and its relative *range concatenation grammar* (RCG; [4,5]), are grammar formalisms based on *predicates* over string tuples. A grammar is a collection of *clauses* for predicates, very similar to Horn clauses and the programming language Prolog. We here define the general formalism of LMG, and then two equivalent subclasses, RCG and *simple* LMG (sLMG). We assume given a finite set Σ of terminal tokens, and an infinite supply of logical variables $x_1, x_2, \ldots \in \mathsf{Var}$.

Definition 4 (predicate). *A* predicate *is a term $A(\alpha_1, \ldots, \alpha_n)$, where each $\alpha_i \in (\Sigma \cup \mathsf{Var})^*$ is a concatenative sequence of terminals and logical variables.*

Definition 5 (clause). *A* clause *is of the form $\phi \vdash \psi_1, \ldots, \psi_m$ where each of $\phi, \psi_1, \ldots, \psi_m$ are predicates. A clause can be* instantiated *by substituting a string for each variable in the clause.*

A literal movement grammar consists a finite number of clauses together with a designated start predicate. To define the language of a LMG grammar G, we define a rewriting relation $\Rightarrow_G$ on sequences of instantiated predicates,

$$\Gamma_1, \phi, \Gamma_2 \Rightarrow_G \Gamma_1, \psi_1, \ldots, \psi_m, \Gamma_2$$

whenever $\phi \vdash \psi_1, \ldots, \psi_m$ is an instantiation of a clause in G. The language of a grammar is then,

$$\mathcal{L}(G) = \{\, w \in \Sigma^* \mid S(w) \Rightarrow_G^* \epsilon \,\}$$

where S is the start predicate in G.

Example 3. The example grammar looks like follows in LMG format:

$$\begin{aligned}
\mathsf{S}(x\ y_1\ y_2) &\vdash \mathsf{NP}(x),\ \mathsf{VP}(y_1, y_2) \\
\mathsf{S}(\text{`}it\ \ is\text{'}\ y_2\ \text{`}that\text{'}\ x\ y_1) &\vdash \mathsf{NP}(x),\ \mathsf{VP}(y_1, y_2) \\
\mathsf{VP}(x, y) &\vdash \mathsf{V}(x),\ \mathsf{NP}(y) \\
\mathsf{NP}(x\ y) &\vdash \mathsf{D}(x),\ \mathsf{N}(y) \\
\mathsf{D}(\text{`}most\text{'}) &\vdash \epsilon \\
\mathsf{N}(\text{`}cats\text{'}) &\vdash \epsilon \\
\mathsf{NP}(\text{`}fish\text{'}) &\vdash \epsilon \\
\mathsf{V}(\text{`}eat\text{'}) &\vdash \epsilon \\
\mathsf{V}(\text{`}catch\text{'}) &\vdash \epsilon
\end{aligned}$$

A possible instantiation of the second clause is:

$$\mathsf{S}(\text{`}it\ \ is\ \ fish\ \ that\ \ most\ \ cats\ \ eat\text{'}) \vdash \mathsf{NP}(\text{`}most\ \ cats\text{'}), \mathsf{VP}(\text{`}eat\text{'}, \text{`}fish\text{'})$$

LMG is a very general, Turing-complete, grammar formalism. To get a recognizable subclass of LMG, one can consider two possibilities; to restrict the definition of clause instantiation, or to put syntactic restrictions on the form of the predicates.

Definition 6 (RCG). *A* range concatenation grammar *(RCG) is an LMG with a restricted form of clause instantiation. A clause can only be instantiated by substrings of the given input string; i.e. if* $\phi \vdash \psi_1, \ldots, \psi_m$ *is an instantiation of a clause, then all arguments to* $\phi, \psi_1, \ldots, \psi_m$ *are substrings of the input.*

By only allowing instantiations by substrings of the input we assure that all strings in a RCG can be replaced by pairs of input positions, called *ranges*.[2] This has the effect that RCG parsing is polynomial in the length of the input string.

Example 4. If the input string is '*b a c h*', then for the following clauses,

$$A(bac) \vdash B(b),\ C(c)$$
$$A(bach) \vdash B(b),\ C(ch)$$
$$A(back) \vdash B(b),\ C(ck)$$

the first two are RCG instantiations of the clause $A(x\ a\ z) \vdash B(x),\ C(z)$; but not the third.

Definition 7 (sLMG). *A* simple *LMG (sLMG) is an LMG where each clause obeys the following three syntactic restrictions:*

- ***Non-combinatorial (NC)****: The arguments of each* ψ_i *are variables.*
- ***Bottom-up nonerasing (BNE)****: All variables in each* ψ_i *also occur in* ϕ*.*
- ***Bottom-up linear (BL)****: No variable occurs more than once in* ϕ*.*

Strictly speaking, bottom-up linearity is not a necessary condition, as the following lemma states:

Lemma 2. *Any LMG clause can be converted to an equivalent bottom-up linear (BL) clause. Furthermore, the conversion preserves NC and BNE.*

Proof (taken from [2,3]). Assume that the clause in question is $\phi \vdash \psi_1, \ldots, \psi_m$, and that there is a variable x occurring twice in ϕ. Replace one occurrence by a new variable x', and add a call to the bottom-up linear predicate $\mathsf{Eq}(x, x')$, with the following definition:

$$\mathsf{Eq}(\epsilon, \epsilon) \vdash \epsilon$$
$$\mathsf{Eq}(s\ x, s\ y) \vdash \mathsf{Eq}(x, y) \qquad \text{(for each } s \in \Sigma)$$

The new clause $\phi \vdash \psi_1, \ldots, \psi_m, \mathsf{Eq}(x, x')$ is equivalent to the original, since the predicate call $\mathsf{Eq}(x, x')$ says that x and x' are equal strings.

The conversion preserves NC, since the predicate $\mathsf{Eq}(x, x')$ is non-combinatorial. Furthermore, is preserves BNE, since the only variable that is introduced on the left-hand side (x') is also introduced on the right-hand side. □

[2] Boullier [4,5] defines RCG directly on ranges, but our definition is equivalent.

Both formalisms sLMG and RCG are equivalent, since they describe exactly the class of languages recognizable in polynomial time [2,3,4,5,13]. Note that sLMG/RCG are closed under intersection; if S_1 and S_2 are the start predicates of G_1 and G_2, then $S(x) \vdash S_1(x)$, $S_2(x)$ defines the intersection of the languages $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$.

1.5 PMCFG is an Instance of sLMG/RCG

Assume given the following PMCFG rule:

$$\begin{aligned} & A \to f[B_1, \ldots, B_\delta] \\ f^\circ(x_{1,1}, \ldots, x_{1,n_1}\,; & \\ \ldots\,; & \\ x_{\delta,1}, \ldots, x_{\delta,n_\delta}) &= \alpha_1, \ldots, \alpha_n \end{aligned}$$

By lemma 1, we can assume that the linearization is nonerasing. Furthermore, it is straightforward to convert a nonerasing PMCFG grammar into an equivalent sLMG grammar, as shown in [2,3]. Each rule above is converted to the clause:

$$\begin{aligned} A(\alpha_1, \ldots, \alpha_n) \vdash\ & B_1(x_{1,1}, \ldots, x_{1,n_1}), \\ & \ldots, \\ & B_\delta(x_{\delta,1}, \ldots, x_{\delta,n_\delta}) \end{aligned}$$

Note that this clause is NC (since each of the $x_{i,j}$ is a variable) and BNE (since f° is nonerasing), and therefore the clause is sLMG.

2 The Intersection Operation

There is an extension of context-free grammar called *conjunctive grammar* [6], where the right-hand sides of rules are extended with a new intersection operator. A conjunctive context-free rule is written:

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_n$$

where $\alpha_i \in (N \cup \Sigma)^*$. The informal interpretation is that A can be rewritten to $w \in \Sigma^*$ iff all α_i can be rewritten to w. This operation can be directly transformed to PMCFG linearizations.

Definition 8 (intersection). *The intersection operation is a partial linearization operation with the definition; $\phi_1 \,\&\, \phi_2$ is calculated to ϕ_1 iff $\phi_1 = \phi_2$.*

This definition can be made formal by lifting the linearization types to sets of linearization values; where the unit set denotes the existence of a linearization and the empty set denotes an undefined linearization. String concatenation, tuple forming and tuple projection are straightforwardly lifted to this domain. The definition of the intersection operation then simply becomes set intersection.

We call PMCFG extended with the intersection operation *conjunctive* PMCFG. The following laws for intersections of linearizations are simple consequences of the formal definition:

$$\phi \,\&\, \phi = \phi$$
$$\alpha\ (\beta_1 \,\&\, \beta_2)\ \gamma = (\alpha\ \beta_1\ \gamma) \,\&\, (\alpha\ \beta_2\ \gamma)$$

The second law says that we can push out an intersection to a row, which is used in the equivalence proof later.

Example 5. In our running example, we have introduced discontinuous verb phrases to handle topicalization. Groenink [2,3] suggests to handle verb phrase coordination by using conjunction on the verb component of the verb phrase. In PMCFG format, this looks like follows:

$$\mathsf{VP} \to \mathit{coord}[\mathsf{VP},\ \mathsf{VP}]$$
$$\mathit{coord}^\circ(\langle x_1, x_2\rangle\ ,\ \langle y_1, y_2\rangle) = \langle x_1\ \text{`}\mathit{and}\text{'}\ y_1,\ x_2 \,\&\, y_2\rangle$$

By combining two verb phrases with the same object, we can form a coordinated verb phrase:

$$\mathit{coord}^\circ(\langle \text{`}\mathit{catch}\text{'}, \text{`}\mathit{fish}\text{'}\rangle\ ,\ \langle \text{`}\mathit{eat}\text{'}, \text{`}\mathit{fish}\text{'}\rangle) = \langle \text{`}\mathit{catch\ \ and\ \ eat}\text{'},\ \text{`}\mathit{fish}\text{'}\rangle$$

which in turn can be used to form sentences like '*many cats catch and eat fish*', or the topicalized version '*it is fish that many cats catch and eat*'.

2.1 A Strict Extension of PMCFG

Theorem 1. *The class of languages recognized by conjunctive PMCFG grammars is closed under intersection.*

Proof. Let G_1 and G_2 be two grammars (with no common categories or function symbols) recognizing the languages $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ respectively. Let G contain all rules from G_1 and G_2 plus the following single rule for the new starting category S:

$$S \to f[S_1,\ S_2]$$
$$f^\circ(x,\ y) = x \,\&\, y$$

It is trivial to see that G recognizes all and only those strings that are recognized by both G_1 and G_2. □

Corollary 1. *The intersection operation is a strict extension of PMCFG.*

The corollary follows from the fact that PMCFG is not closed under intersection [1], a property it shares with context-free grammars.

2.2 Language-Theoretic Implications

Closedness under intersection has some less desirable properties, which conjunctive PMCFG inherits from *conjunctive grammar* [6]:

- The following decision problems are undecidable: emptiness, finiteness, regularity, context-freeness, inclusion and equivalence. This is because these decision problems are undecidable for finite intersections of context-free grammars, see e.g. [14].
- Conjunctive PMCFG is not closed under homomorphism. This follows from the fact that any recursively enumerable language $\mathcal{L}$ can be described by $h(\mathcal{L}_1 \cap \mathcal{L}_2)$, for some homomorphism h and context-free languages $\mathcal{L}_1$, $\mathcal{L}_2$, see e.g. [15].

2.3 Usefulness of Intersection

Conjunctive PMCFG is not only closed under intersection, but the closure is also *modular*, i.e. it preserves the structure of the underlying grammar conjuncts. This makes it useful for modular grammar engineering, as already noted in [4,5]. Intersection might also be useful for modeling secondary/tertiary structures of biological sequences, as has been investigated in [16]. For purely linguistic phenomena, [2,3] contains a suggestion of how to use intersection to describe verb coordination, as shown in example 5.

3 Conjunctive PMCFG Describes the Polynomial Languages

In this section we show that conjunctive PMCFG, or to be more exact, nonerasing conjunctive PMCFG, is equivalent to sLMG and RCG. The following theorem is a direct consequence of lemmas 3, 4 and 5 below:

Theorem 2. *Nonerasing conjunctive PMCFG is equivalent to sLMG and RCG.*

Since it is already known that sLMG and RCG exactly describe the class of languages recognizable in polynomial time, we get the same result for nonerasing PMCFG extended with intersection.

Corollary 2. *The class of languages recognizable by nonerasing conjunctive PMCFG is exactly the class of languages recognizable in polynomial time.*

3.1 Conjunctive PMCFG is an instance of sLMG/RCG

Lemma 3. *Any nonerasing conjunctive PMCFG can be converted to an equivalent sLMG.*

Proof. Since intersections can be pushed out, we can assume that the PMCFG rules are of the form,

$$\begin{aligned} & A \to f[B_1, \ldots, B_\delta] \\ f^\circ(\langle x_{1,1}, \ldots, x_{1,n_1} \rangle, & \\ \ldots, & \\ \langle x_{\delta,1}, \ldots, x_{\delta,n_\delta} \rangle) = \langle & \alpha_{1,1} \,\&\, \ldots \,\&\, \alpha_{1,c_1}, \\ & \ldots, \\ & \alpha_{n,1} \,\&\, \ldots \,\&\, \alpha_{n,c_n} \rangle \end{aligned}$$

where each $\alpha_{i,j}$ is a sequence of strings and variables, as above. Translate this to the sLMG clause,

$$\begin{aligned} \hat{A}(\alpha_{1,1} \,\&\, \ldots \,\&\, \alpha_{1,c_1} \,; & \\ \ldots \,; & \\ \alpha_{n,1} \,\&\, \ldots \,\&\, \alpha_{n,c_n}) \vdash & B_1(x_{1,1}, \ldots, x_{1,n_1}), \\ & \ldots, \\ & B_\delta(x_{\delta,1}, \ldots, x_{\delta,n_\delta}) \end{aligned}$$

where the left-hand side is just syntactic sugar for a predicate with arity $c_1 + \cdots + c_n$. The clause is NC (since each of the $x_{i,j}$ is a variable) and BNE (since f° is nonerasing), and therefore it is sLMG. Finally, add coercion clauses for $\hat{A}(\ldots)$, implementing the intersections:

$$A(x_1, \ldots, x_n) \vdash \hat{A}(x_1 \,\&\, \ldots \,\&\, x_1 \,; \ldots ; \, x_n \,\&\, \ldots \,\&\, x_n)$$

The resulting sLMG grammar is equivalent to the original PMCFG grammar. □

Example 6. The following is the result of translating the PMCFG rule for verb coordination in example 5, into sLMG/RCG:

$$\begin{gathered} \widehat{\mathsf{VP}}(x_1 \text{ `}\textit{and}\text{'} \, y_1 \,;\; x_2 \,\&\, y_2) \vdash \mathsf{VP}(x_1, x_2),\ \mathsf{VP}(y_1, y_2) \\ \mathsf{VP}(x, y) \vdash \widehat{\mathsf{VP}}(x \,;\, y \,\&\, y) \end{gathered}$$

After simplifying away $\widehat{\mathsf{VP}}$, we get the same clause as in Groenink's original example [2,3]:

$$\mathsf{VP}(x \text{ `}\textit{and}\text{'} \, y,\ z) \vdash \mathsf{VP}(x, z),\ \mathsf{VP}(y, z)$$

3.2 sLMG/RCG is an Instance of Conjunctive PMCFG

We say that a clause $\phi \vdash \psi_1, \ldots, \psi_m$ is *top-down nonerasing* (TNE) if all variables in ϕ also occur in some ψ_i.

Lemma 4. *Any LMG clause can be converted to an equivalent top-down nonerasing (TNE) clause. Furthermore, the conversion preserves NC and BNE.*

Proof. Assume that the clause in question is $\phi \vdash \psi_1, \ldots, \psi_m$, and that there is a variable x in ϕ not occurring in any of $\psi_1, \ldots, \psi_m$. Add a call to the top-down nonerasing predicate $\mathsf{Str}(x)$, with the following definition:

$$\begin{aligned} \mathsf{Str}(\epsilon) &\vdash \epsilon \\ \mathsf{Str}(s\ x) &\vdash \mathsf{Str}(x) \qquad (\text{for each } s \in \Sigma) \end{aligned}$$

The new clause $\phi \vdash \psi_1, \ldots, \psi_m, \mathsf{Str}(x)$ is equivalent to the original, since the predicate $\mathsf{Str}(x)$ only says that x is a string.

The conversion preserves NC, since the predicate $\mathsf{Str}(x)$ is non-combinatorial. Furthermore, it preserves BNE, since no variable is introduced. □

Lemma 5. *Any top-down nonerasing sLMG can be converted to an equivalent nonerasing conjunctive PMCFG.*

Proof. A sLMG clause is of the following form:

$$\begin{aligned} A(\alpha_1, \ldots, \alpha_n) \vdash\ & B_1(x_{1,1}, \ldots, x_{1,n_1}), \\ & \ldots, \\ & B_\delta(x_{\delta,1}, \ldots, x_{\delta,n_\delta}) \end{aligned}$$

If the variables $x_{i,j}$ all are distinct, it is equivalent to the PMCFG rule:

$$\begin{aligned} & A \to f[B_1, \ldots, B_\delta] \\ f^\circ(&\langle x_{1,1}, \ldots, x_{1,n_1} \rangle, \\ & \ldots, \\ & \langle x_{\delta,1}, \ldots, x_{\delta,n_\delta} \rangle) = \langle \alpha_1, \ldots, \alpha_n \rangle \end{aligned}$$

However, in sLMG, the variables in the right-hand side of a clause need not be distinct. Assume therefore that $x_{i',j'} = x_{i,j} = x$. Now, introduce a new variable x' to replace x as $x_{i',j'}$; and replace each occurrence of x in the right-hand side with the conjunction $(x \,\&\, x')$.

The resulting rule is a syntactically correct conjunctive PMCFG rule, and equivalent to the given sLMG clause. Furthermore, it is nonerasing since the original clause is TNE. □

Example 7. The resulting clause from the previous example,

$$\mathsf{VP}(x\ \text{`}and\text{'}\ y,\ z) \vdash \mathsf{VP}(x, z),\ \mathsf{VP}(y, z)$$

is converted to the following conjunctive PMCFG rule:

$$\begin{aligned} & \mathsf{VP} \to f[\mathsf{VP}, \mathsf{VP}] \\ f^\circ(&\langle x, z \rangle, \langle y, z' \rangle) = \langle x\ \text{`}and\text{'}\ y, z \,\&\, z' \rangle \end{aligned}$$

4 Discussion

The results in this and earlier papers [1,2,3,4,5,11,17] give some insights into the nature of the class of polynomial time recognizable languages. We can now try to describe what kind of constructions are necessary (and sufficient) to be able to describe any polynomial language, apart from ordinary string concatenation.

Multiple constituents Parse time complexity is directly related to the maximal number of discontinuous constitutents in a grammar [2,3,4,5,11,17]. Therefore there should be some way of coding discontinuous constitutents in a grammar, be it with string tuples as in the formalisms discussed in this paper, or with some kind of ingenious coding.

Reduplication The exponentially growing language a^{2^n} is polynomially recognizable, and we see no other (simple) way of describing that language but to use string duplication – there can only be a finite number of multiple constituents in a grammar, and an intersection cannot be used to duplicate strings.

Intersection As noted by Boullier [4,5], the intersection of two polynomially parsable languages is also polynomially parsable – simply recognize for each language in turn. And since formalisms with multiple constituents and reduplication (e.g. PMCFG) are not closed under intersection, we have to introduce intersection as an explicit operation.

Suppose we design a new grammar formalism having some construction which cannot be decomposed into these constructions. Then our conjecture is that the formalism cannot be parsable in polynomial time.

One such construction which we have already come across is *erasing* PMCFG grammars – the possibility to erase linearization information of parts of the syntax tree, as discussed in section 1.3. Without the intersection operation, any erasing grammar can be transformed into an equivalent nonerasing grammar. But for conjunctive PMCFG, it is not clear whether erasing syntax rules can be transformed away. Either it is possible, in which case any conjunctive PMCFG is polynomially parsable; or it is not possible, in which case conjunctive PMCFG is not polynomial in general.

References

1. Seki, H., Matsumara, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science **88** (1991) 191–229
2. Groenink, A.: Mild context-sensitivity and tuple-based generalizations of context-free grammar. Linguistics and Philosophy **20** (1997) 607–636
3. Groenink, A.: Surface without Structure — Word order and tractability issues in natural language analysis. PhD thesis, Utrecht University (1997)
4. Boullier, P.: A cubic-time extension of context-free grammars. Grammars **3** (2000) 111–131
5. Boullier, P.: Range concatenation grammars. In: 6th International Workshop on Parsing Technologies, Trento, Italy (2000) 53–64

6. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics **6** (2001) 519–535
7. Pollard, C.: Generalised Phrase Structure Grammars, Head Grammars and Natural Language. PhD thesis, Stanford University (1984)
8. Weir, D.: Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania, Philadelphia, PA (1988)
9. Becker, T.: HyTAG: A New Type of Tree Adjoining Grammars. PhD thesis, Universität des Saarlandes (1994)
10. Chiang, D.: Constraints on strong generative power. In: 39th Meeting of the Association for Computational Linguistics. (2001) 124–131
11. Ljunglöf, P.: Expressivity and Complexity of the Grammatical Framework. PhD thesis, Göteborg University (2004)
12. Kasami, T., Seki, H., Fujii, M.: Generalized context-free grammars and multiple context-free grammars. IEICE Transactions **J71-D-I** (1988) 758–765
13. Bertsch, E., Nederhof, M.J.: On the complexity of some extensions of RCG parsing. In: 7th International Workshop on Parsing Technologies. (2001) 66–77
14. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
15. Ginsburg, S.: Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland/Elsevier (1975)
16. Chiang, D.: Evaluating Grammar Formalisms for Applications to Natural Language Processing and Biological Sequence Analysis. PhD thesis, University of Pennsylvania (2004)
17. Satta, G.: Recognition of linear context-free rewriting systems. In: 30th Meeting of the Association for Computational Linguistics, Newark, Delaware (1992) 89–95