MORPHISMS OF CONTEXT-FREE GRAMMARS

TIBOR BEKE

1. INTRODUCTION

Let me begin with a little known comment by Noam Chomsky (see [GE82] p.15 or [GE04] p.42), made in response to a question on the significance of automata theory for linguistics and mathematics:

> This seems to me what one would expect from applied mathematics, to see if you can find systems that capture some of the properties of the complex system that you are working with, and to ask whether those systems have any intrinsic mathematical interest, and whether they are worth studying in abstraction. And that has happened exactly at one level, the level of context-free grammar. At any other level it has not happened. The systems that capture other properties of language, for example, of transformational grammar, hold no interest for mathematics. But I do not think that is a necessary truth. It could turn out that there would be richer and more appropriate mathematical ideas that would capture other, maybe deeper properties of language than context free grammars do. In that case you have another branch of applied mathematics which might have linguistic consequences. That would be exciting.

Let me hasten to say that I do *not* wish to argue with Chomsky's assessment. It would be hard to do, at any rate, since he leaves room for both possibilities: that there is no linguistic theory beyond context-free grammars that is of interest to mathematics; or perhaps there *is*. But I was particularly struck by the sentence *The systems that capture other properties of language, for example, of transformational grammar, hold no interest for mathematics.* From the 1970's on, transformational grammar has been responsible, directly on indirectly, for much research on generalizations of automata that, instead of transforming strings to strings, transform trees to trees. Rational transducers, for example, gave rise to a variety of tree transducers (deterministic, non-deterministic, top-down, bottom-up), in no small part motivated by the desire to find a compact mathematical formalism underlying transformational grammar. In fact, transformations of parse trees, called translations in the computer science literature, are central to the contemporary theory of compilers. There has been a subtle change of perspective, though. Transformational grammar, motivated by examples such as the English passive, seeks to understand operations on tree-like structures within *one* given language. Compilers translate from source code to object code: from one (typically context-free) language to another.

One can appeal to an algebraic analogy at this point. If context-free languages are like algebras, then context-free grammars are like presentations of algebras via generators and relations. One can

Date: August 3, 2017.

map one set of generators into another in a way that preserves relations; such a mapping induces a homomorphism of algebras. So there ought to be such a thing as mapping one context-free grammar into another in a 'structure-preserving' way, and this should induce a homomorphism between languages.

The goal of this note is to give *one* possible definition of morphism of context-free grammars. This notion will organize context-free grammars into a category [CWM] in such a way that the effects of morphisms on parse trees — these are, more or less, the 'translations' of computer science — become functorial. The appearance of these category-theoretic concepts is somewhat auxiliary, however, to the main enterprise, which is to understand what it means to map one grammar into another 'in a grammatical way'.

We will be guided by four examples of grammatical operations. Keeping in mind Chomsky's dictum, each of them arises naturally within some body of formalized mathematics — algebra or logic. Going through the motivating examples, the reader is invited to play with the following questions: Which levels of the Chomsky hierarchy do the source and target languages belong to? Which family of transformations (translations? transductions?) does the operation belong to? Each of the motivating examples is given by an explicit formal recipe. Isn't that recipe an outright 'morphism'?

Motivating examples.

(a) In a (non-commutative) ring, the commutator $[x, y]$ of two elements x, y is defined by
$$[x, y] = x \cdot y - y \cdot x$$
Let L_0 be the language of well-formed iterated commutators of elements, and let L_1 be the language of well-parenthesized terms in the function symbols $\cdot$ and $-$. Consider the operation that associates to an expression in L_0 its equivalent in L_1 (prior to expansion and simplification). For example, $[[x, y], z]$ is to be mapped to
$$(((x \cdot y) - (y \cdot x)) \cdot z) - (z \cdot ((y \cdot x) - (x \cdot y))) \ .$$

(b) Consider the language L_0 of (ambiguous) parenthesis-free terms formed from a set of variables with the binary operators $\circledast$ and $\boxtimes$. Let L_1 be the language of terms, with the same operators, in prefix form. Consider the multi-valued mapping that associates to a term in L_0 its prefix forms, under all possible parses. For example, the possible parses of $x \circledast y \boxtimes z$ are (using parentheses, informally)
$$(x \circledast y) \boxtimes z \qquad \text{resp.} \qquad x \circledast (y \boxtimes z)$$
or $\boxtimes \circledast xyz$ resp. $\circledast x \boxtimes yz$ in prefix form.

Can this multi-valued mapping be described without mentioning prefix and infix traversals of binary trees?

(c) Fix a first order signature, and consider the language L of well-formed formulas of first order logic. Let x be a variable, $\mathtt{t}$ a term and ϕ a formula in L. Define the result $\tau_{x \to \mathtt{t}}(\phi)$ of *replacing the free occurrences of* x *in* ϕ *by* $\mathtt{t}$ by the usual set of rules. (These rules will not be recalled here; see e.g. Mendelson [M10] or any careful textbook of logic.) Fix x and t, and consider the map from L to itself sending ϕ to $\tau_{x\to\mathsf{t}}(\phi)$.

(d) Consider again the language L of first order logic. The *negation normal form*, $\textsc{nnf}(\phi)$ of a formula ϕ is defined by the rewrite rules

$$\neg\neg\phi \Rightarrow \phi$$

$$\neg(\phi\wedge\psi) \Rightarrow \neg\phi\vee\neg\psi$$

$$\neg(\phi\vee\psi) \Rightarrow \neg\phi\wedge\neg\psi$$

$$\neg\,\forall x\phi \Rightarrow \exists x\neg\psi$$

$$\neg\,\exists x\phi \Rightarrow \forall x\neg\psi$$

Iterated application of these rules transforms any well-formed formula into a logically equivalent one where the targets of negation symbols (if any) are atomic formulas. Does the operation sending ϕ to $\textsc{nnf}(\phi)$ belong in the same family as any of (a), (b) or (c)?

Notation. We will consider alphabets $\mathcal{A}$ and context-free grammars G with productions written $x \to \mathsf{s}$ where $x \in \mathcal{A}$ and s is a string in $\mathcal{A}^*$. Neither $\mathcal{A}$ nor G is assumed finite. An element x of $\mathcal{A}$ is non-terminal if it occurs on the left-hand side of some production, and is terminal otherwise. N and T will denote the set of non-terminal and terminal symbols, respectively; so $\mathcal{A} = N \sqcup T$. For $\mathsf{u}, \mathsf{v} \in \mathcal{A}^*$, write $\mathsf{u} \Rightarrow \mathsf{v}$ if v is immediately derivable from u; let $\Rightarrow^+$ denote the transitive and $\Rightarrow^*$ the reflexive-transitive closure of the relation $\Rightarrow$.

We will find it convenient to consider each non-terminal as a possible start symbol, and to consider strings both in the full alphabet $\mathcal{A}$ and in the set of terminals T. For $x \in N$, define

$$\hat{L}_G(x) = \{\mathsf{u} \in \mathcal{A}^* \mid x \Rightarrow^* \mathsf{u}\}$$

and

$$L_G(x) = \{\mathsf{u} \in T^* \mid x \Rightarrow^* \mathsf{u}\}$$

Thus, for non-terminal x, $\hat{L}_G(x)$ is the set of sentential forms that can be generated from x (considered as a start symbol), and $L_G(x)$ is the usual language generated from x.

Let us recall the notion of unambiguous grammar in the form that will be most useful to us:

Definition 1.1. The context-free grammar G is unambiguous if for every non-terminal x and $\mathsf{u} \in \mathcal{A}^*$ with $x \Rightarrow^+ \mathsf{u}$ there exists exactly one pair of k-tuples

$$s_1, s_2, s_3, \ldots, s_k;\ \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3, \ldots, \mathsf{u}_k$$

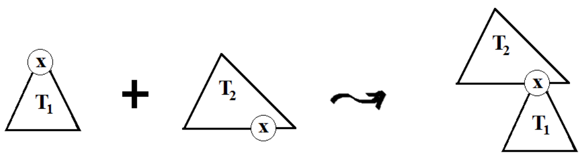
where $s_i \in \mathcal{A}$ and $\mathsf{u}_i \in \mathcal{A}^*$, such that

- $x \to s_1 s_2 s_3 \ldots s_k$ is a production
- $\mathsf{u} = \mathsf{u}_1\mathsf{u}_2\ldots\mathsf{u}_k$, and
- $s_i \Rightarrow^* \mathsf{u}_i$ for each $1 \leqslant i \leqslant k$.

This is equivalent to the requirement that the parse tree of every sentential form $\mathsf{u} \in \hat{L}_G(x)$ be unique; or, equivalently, that there exist a unique leftmost derivation, starting from x, for each $\mathsf{u} \in \hat{L}_G(x)$. If every non-terminal is productive, that is, $L_G(x)$ is non-empty for all non-terminals x, then Def. 1.1 is equivalent to the unambiguity of the $L_G(x)$ in the classical sense. However, Def. 1.1 makes sense even if some or all of the $L_G(x)$ are empty.

Definition 1.2. For $x \in N$, let $\mathrm{tree}_G(x)$ denote the set of parse trees of sentential forms from $\hat{L}_G(x)$, with root x. (One could just as well consider the set of leftmost or rightmost derivations, or other representatives of equivalence classes of derivations, but the formalism of trees is the handiest.) The depth of a tree is the number of nodes on the longest path from root to any leaf, minus 1. Thus, for $T \in \mathrm{tree}_G(x)$, $\mathrm{depth}(T) = 0$ if and only if T consists solely of the root (which is also a leaf) x. Note that $\mathrm{depth}(T) = 1$ if and only if T equals some production $x \to \mathsf{s} \in G$. Let $\mathrm{NT}(T)$ denote the set of leaves of T labeled by non-terminal symbols; for a node t of T, let $\mathrm{label}(t)$ denote the label (i.e. element of the alphabet $\mathcal{A}$) at t.

Let $T_1 \in \mathrm{tree}_G(x)$ and let T_2 be a tree with a leaf t such that $\mathrm{label}(t) = x$. We will skip the definition of the horticultural maneuver of *grafting* T_1 *onto* T_2 *at the location* t. It is the same as the composition of (chains of) productions, as the illustration(s) below will make it clear.



2. Morphisms of grammars

Let G_0 and G_1 be context-free grammars in the alphabets $\mathcal{A}_0$ and $\mathcal{A}_1$, with terminals T_0, T_1 and non-terminals N_0, N_1 respectively.

Definition 2.1. A *morphism from* G_0 *to* G_1 consists of the following data:

- a mapping $\alpha : N_0 \to N_1$
- a mapping β that assigns to each production $x \to \mathsf{s} \in G_0$ an element of $\mathrm{tree}_{G_1}(\alpha(x))$
- for each production $p \in G_0$, a function $\gamma(p, -)$ from $\mathrm{NT}(\beta(p))$ to $\mathrm{NT}(p)$, with the property that for all $t \in \mathrm{NT}(\beta(p))$,

$$\alpha\big(\mathrm{label}(\gamma(p,t))\big) = \mathrm{label}(t) \ .$$

More plainly, α gives the translation of lexical categories. β specifies, for each production $p : x \to \mathsf{s}$ in the source grammar, a parse tree in the target grammar, with root $\alpha(x)$. Productions of the form $x \to \mathsf{s}$ will be translated to trees of the form $\beta(x \to \mathsf{s})$. The re-indexing map $\gamma(p, -)$ associates to the location of each non-terminal symbol r occurring as a leaf in $\beta(x \to \mathsf{s})$ the location of a non-terminal symbol s in s such that α will translate s to r. This permits translation of the input parse tree by either top-down or bottom-up recursion.

Let us make this more concrete by a formalization of our motivating example (a). For the sake of readability, we will depart from the BNF convention of enclosing names of non-terminals in angle brackets; strings typeset in sans serif font, such as var and expr, should be considered as stand-alone symbols. Also, we will drop commas separating elements of a set being listed. Dots '. . . ' indicate a (potentially infinite) set indexed by the natural numbers.

Example 2.2. Consider the source alphabet

$$\begin{aligned} N_0 &= \{ \ \mathsf{var}\ \mathsf{expr}\ \} \\ T_0 &= \{ \ [\ ,\ ]\ x_1\, x_2\, \ldots\, x_i\, \ldots \} \end{aligned}$$

Let the grammar G_0 consist of the productions

$$\begin{aligned} \mathsf{var} &\to x_1 \mid x_2 \mid \ \ldots\ \mid x_i \mid \ \ldots \\ \mathsf{expr} &\to [\mathsf{var}, \mathsf{var}] \\ \mathsf{expr} &\to [\mathsf{var}, \mathsf{expr}] \\ \mathsf{expr} &\to [\mathsf{expr}, \mathsf{var}] \\ \mathsf{expr} &\to [\mathsf{expr}, \mathsf{expr}] \end{aligned}$$

Now consider the target alphabet

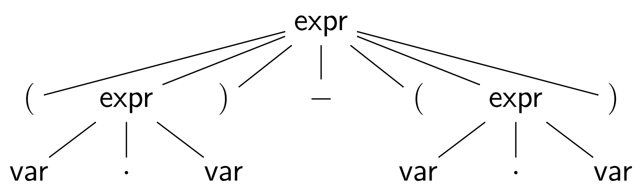
$$\begin{aligned} N_1 &= \{ \ \mathsf{var}\ \mathsf{expr}\ \} \\ T_1 &= \{ \ (\ )\ -\ \cdot\, x_1\, x_2\, \ldots\, x_i\, \ldots \} \end{aligned}$$

Let the grammar G_1 consist of the productions

$$\begin{aligned} \mathsf{var} &\to x_1 \mid x_2 \mid \ \ldots\ \mid x_i \mid \ \ldots \\ \mathsf{expr} &\to \mathsf{var} - \mathsf{var} \mid \mathsf{var} \cdot \mathsf{var} \\ \mathsf{expr} &\to \mathsf{var} - (\mathsf{expr}) \mid \mathsf{var} \cdot (\mathsf{expr}) \\ \mathsf{expr} &\to (\mathsf{expr}) - \mathsf{var} \mid (\mathsf{expr}) \cdot \mathsf{var} \\ \mathsf{expr} &\to (\mathsf{expr}) - (\mathsf{expr}) \mid (\mathsf{expr}) \cdot (\mathsf{expr}) \end{aligned}$$

There is a morphism from G_0 to G_1 with components α, β, γ defined by

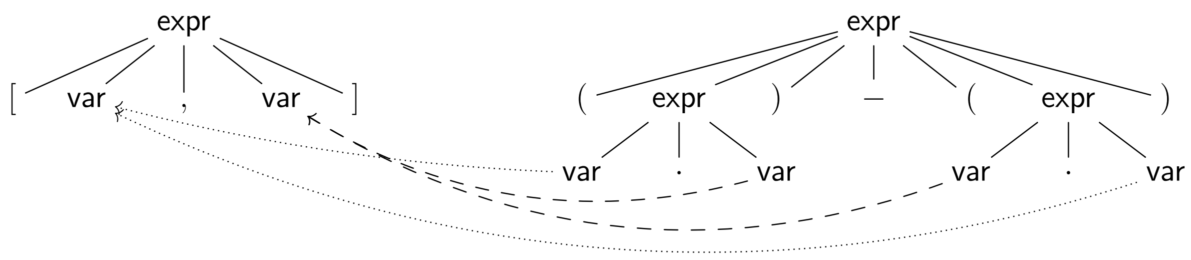
- $\alpha(\ \mathsf{expr}\ ) = \mathsf{expr}$ and $\alpha(\ \mathsf{var}\ ) = \mathsf{var}$
- $\beta(\ \mathsf{var} \to x\ ) = x$ for any variable x; note that $\gamma(\mathsf{var} \to x, -)$ has empty domain
- $\beta(\ \mathsf{expr} \to [\mathsf{var}, \mathsf{var}]\ )$ is



generating the string $(\mathsf{var} \cdot \mathsf{var}) - (\mathsf{var} \cdot \mathsf{var})$. Let us refer to the leaves of the above tree via their location in '$(\mathsf{var} \cdot \mathsf{var}) - (\mathsf{var} \cdot \mathsf{var})$'; so the leaves labeled with non-terminals occur at $\{2, 4, 8, 10\}$. Similarly, let us refer to the leaves in $\text{NT}(\,\mathsf{expr} \to [\mathsf{var}, \mathsf{var}]\,)$ through their location in the string '$[\mathsf{var}, \mathsf{var}]$', i.e. $\{2, 4\}$. Then define

$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{var}], 2) = 2$$
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{var}], 4) = 4$$
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{var}], 8) = 4$$
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{var}], 10) = 2$$

Visually, the re-indexing map $\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{var}], -)$ is indicated by the dotted and broken arrows



Continuing with the next production, define

$$\beta(\,\mathsf{expr} \to [\mathsf{var}, \mathsf{expr}]\,) = (\mathsf{var} \cdot (\mathsf{expr})) - ((\mathsf{expr}) \cdot \mathsf{var})$$

(Since G_1 is unambiguous, we will identify sentential forms with their parse trees.) Using the same coding of locations as above, define

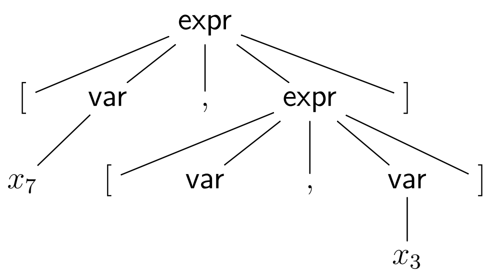
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{expr}], 2) = 2$$
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{expr}], 5) = 4$$
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{expr}], 10) = 4$$
$$\gamma(\mathsf{expr} \to [\mathsf{var}, \mathsf{expr}], 13) = 2$$

The treatment of the other two productions, and re-indexing of non-terminals therein, is analogous.

How does translation from $\hat{L}_{G_0}(\mathsf{expr})$ to $\hat{L}_{G_1}(\mathsf{expr})$ actually work? Consider a sentential form generated by G_0 from expr, say,

$$[x_7, [\mathsf{var}, x_3]]$$

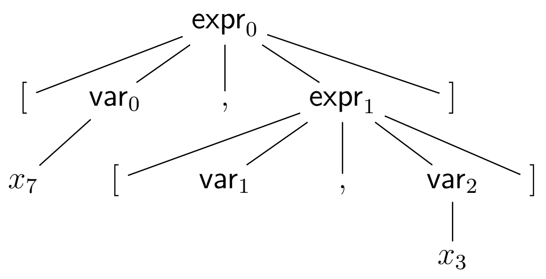
with parse tree



Since G_1 is unambiguous, the process is easiest to describe by bottom-up induction. Starting from the leaves, associate to each non-terminal symbol t in the input tree a string $\tau(t)$ from $\hat{L}_{G_1}(\alpha(x))$:

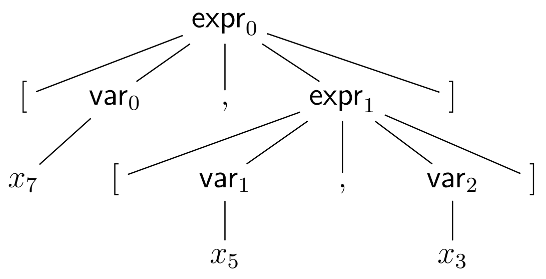
- If t is a leaf, let $\tau(t) = \alpha(t)$.
- If t is var, with descendant $\mathsf{var} \to x$, set $\tau(\mathsf{var}) = x$.
- Suppose t is a node expr with descendants, say, $[\mathsf{var}, \mathsf{var}]$. Let $s_1 = \tau(\mathsf{var})$ for the first occurrence of 'var' in $[\mathsf{var}, \mathsf{var}]$, and $s_2 = \tau(\mathsf{var})$ for the second occurrence. (τ is supposed to be defined on those two symbols by induction.) Then set
$$\tau(\mathsf{expr}) = (s_1 \cdot s_2) - (s_2 \cdot s_1)$$
The idea is analogous for the other productions with source expr.

To see what is going on, let us affix subscripts to the non-terminals of the above parse tree:



Then
$$\tau(\mathsf{var}_1) = \mathsf{var} \qquad \tau(\mathsf{var}_0) = x_7 \qquad \tau(\mathsf{var}_2) = x_3$$
$$\tau(\mathsf{expr}_1) = (\mathsf{var} \cdot x_3) - (x_3 \cdot \mathsf{var})$$
$$\tau(\mathsf{expr}_0) = (x_7 \cdot ((\mathsf{var} \cdot x_3) - (x_3 \cdot \mathsf{var}))) - (((\mathsf{var} \cdot x_3) - (x_3 \cdot \mathsf{var})) \cdot x_7)$$

For the parse tree



a moment's thought confirms that
$$\tau(\mathsf{expr}_0) = (x_7 \cdot ((x_5 \cdot x_3) - (x_3 \cdot x_5))) - (((x_5 \cdot x_3) - (x_3 \cdot x_5)) \cdot x_7)$$
respecting all long-distance dependencies.

Above, G_0 was an unambiguous grammar, hence one could talk of the translation of a string or of a parse tree interchangeably. The next proposition defines the effect of a morphism of grammars in general. We retain the notation of Def. 2.1.

Proposition 2.3. *A morphism of grammars from G_0 to G_1 induces, for each $x \in N_0$, a mapping*
$$\tau : \mathrm{tree}_{G_0}(x) \to \mathrm{tree}_{G_1}(\alpha(x)) \ .$$

Indeed, for $T \in \mathrm{tree}_{G_0}(x)$, define $\tau(T) \in \mathrm{tree}_{G_1}(\alpha(x))$ by induction on the depth of T:

• If $\mathrm{depth}(T) = 0$, then T must be x itself, and $\tau(T)$ is defined to be $\alpha(x)$.
• If $\mathrm{depth}(T) > 0$, let $x \to \mathsf{s} \in G_0$ be the top production in T. Write p for $x \to \mathsf{s}$ for brevity. Note that $\mathrm{NT}(p)$ can be identified with a subset of s, namely, the locations of the non-terminal symbols in s. Since G_0 is context-free, each $s \in \mathrm{NT}(p)$ induces a subtree T_s of T with s as root. For each $t \in \mathrm{NT}(\beta(p))$, graft the tree $\tau(T_{\gamma(p,t)})$ on $\beta(p)$ with t as root. $\tau(T)$ is defined to be the resulting tree.

The definition makes sense: since $\mathrm{depth}(T_s) < \mathrm{depth}(T)$ for any $s \in \mathrm{NT}(p)$, $\tau(T_s)$ is defined by the induction hypothesis. Note that $\tau(T_s)$ belongs to $\mathrm{tree}_{G_1}(\alpha(\mathrm{label}(s))$ by the induction assumption, and $\alpha\big(\mathrm{label}(\gamma(p,t))\big) = \mathrm{label}(t)$ by Def. 2.1. That is, the non-terminal symbol at the root of $\tau(T_{\gamma(p,t)})$ coincides with the non-terminal symbol at the location t. Since G_1 is a context-free grammar, the graft is well-defined, and $\tau(T)$ will belong to $\mathrm{tree}_{G_1}(\alpha(x))$ as desired. $\square$

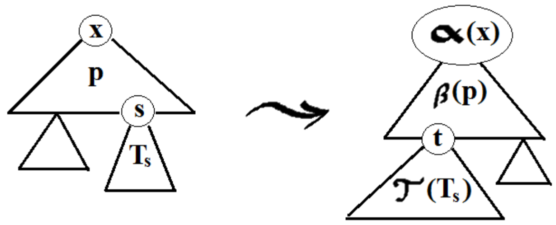


FIG. 1: Computing $\tau(T)$. Above, p is a production (i.e. tree of depth 1), $\beta(p)$ is a tree, s and t are leaves labeled with non-terminal symbols such that $s = \gamma(p,t)$. x and $\alpha(x)$ are the labels of the roots.

Obviously, one can rewrite the above recursive definition into an algorithm to compute $\tau(T)$ by bottom-up induction on T, from leaves toward the root. Note that if $\mathrm{depth}(T) = 1$, that is, T is a production $x \to \mathsf{s}$ in G_0, then $\tau(T)$ ends up being the same as $\beta(T)$.

We will sometimes consider the induced translation τ as a multi-valued mapping

$$\hat{L}_{G_0}(x) \to \hat{L}_{G_1}\big(\alpha(x)\big) \ .$$

Indeed, for each $\mathsf{u} \in \hat{L}_{G_0}(x)$, there is a value for each parse tree T of u, namely, the string in $\hat{L}_{G_1}\big(\alpha(x)\big)$ generated by $\tau(T)$.

Proposition 2.4. *For any $x \in N_0$ and $\mathsf{u} \in L_{G_0}(x)$ with parse tree T, $\tau(T)$ generates a string in $L_{G_1}\big(\alpha(x)\big)$.*

Proof. By induction on the depth of T. $\mathrm{depth}(T) = 0$ is impossible, since x is assumed non-terminal and u is a string of terminals. If $\mathrm{depth}(T) = 1$, then T consists of the single production $x \to \mathsf{u} \in G_0$. The leaves of $\tau(x \to \mathsf{u}) = \beta(x \to \mathsf{u})$ must consist of terminals. Indeed, if there was a leaf labeled with a non-terminal, then $\gamma(x \to \mathsf{u}, -)$ would need to map its location to the location of some non-terminal in u, but u does not contain any non-terminals. So $\tau(T) = \beta(x \to \mathsf{u})$ generates a string in $L_{G_1}\big(\alpha(x)\big)$.

If $\mathrm{depth}(T) > 1$, then $\tau(T)$ is, by the definition, the result of grafting trees of the form $\tau(T_s)$, for subtrees T_s of T, onto those leaves of $\beta(x \to \mathsf{s})$ that contain non-terminals. Using the induction hypothesis, all leaves of $\tau(T_s)$ are labeled with terminal symbols; hence $\tau(T)$ generates an element of $L_{G_1}\big(\alpha(x)\big)$ as well. □

Let us summarize the discussion so far:

Corollary 2.5. *A morphism (α, β, γ) of context-free grammars from G_0 to G_1 induces a function, for each $x \in N_0$, from $\mathrm{tree}_{G_0}(x)$ to $\mathrm{tree}_{G_1}(\alpha(x))$. This induces, in turn, a multi-valued function from $\hat{L}_{G_0}(x)$ to $\hat{L}_{G_1}\big(\alpha(x)\big)$, which restricts to a multi-valued function from $L_{G_0}(x)$ to $L_{G_1}\big(\alpha(x)\big)$. If G_0 is an unambiguous grammar, then the latter two maps are single-valued.*

Example 2.6. Returning to our motivating example (b), consider the source alphabet

$$N_0 = \{\ \mathsf{expr}\ \}$$
$$T_0 = \{\ \circledast\ \boxtimes\ x_1\, x_2\ \ldots\ x_i\ \ldots\}$$

Let the grammar G_0 consist of the productions

$$\mathsf{expr} \to x_1 \mid x_2 \mid \ldots \mid x_i \mid \ldots$$
$$\mathsf{expr} \to \mathsf{expr} \circledast \mathsf{expr}$$
$$\mathsf{expr} \to \mathsf{expr} \boxtimes \mathsf{expr}$$

Now consider the target grammar G_1 with identical alphabet $N_1 = N_0$, $T_1 = T_0$ but productions

$$\mathsf{expr} \to x_1 \mid x_2 \mid \ldots \mid x_i \mid \ldots$$
$$\mathsf{expr} \to \circledast\, \mathsf{expr}\, \mathsf{expr}$$
$$\mathsf{expr} \to \boxtimes\, \mathsf{expr}\, \mathsf{expr}$$

There is a morphism from G_0 to G_1 with components α, β, γ defined by

- $\alpha(\ \mathsf{expr}\ ) = \mathsf{expr}$
- $\beta(\ \mathsf{expr} \to x\ ) = x$ for any variable x; note that $\gamma(\mathsf{expr} \to x, -)$ has empty domain
- $\beta(\ \mathsf{expr} \to \mathsf{expr} \circledast \mathsf{expr}\ ) = \circledast\, \mathsf{expr}\, \mathsf{expr}$ with $\gamma(2) = 1$ and $\gamma(3) = 3$
- $\beta(\ \mathsf{expr} \to \mathsf{expr} \boxtimes \mathsf{expr}\ ) = \boxtimes\, \mathsf{expr}\, \mathsf{expr}$ with $\gamma(2) = 1$ and $\gamma(3) = 3$

(Since G_1 is unambiguous, there is no loss in writing the values of β as strings, as opposed to parse trees. The first argument of γ is suppressed for the sake of readability; numbers refer to locations of non-terminal symbols, as before.) For any $\mathsf{u} \in L_{G_0}(\mathsf{expr})$, the values of $\tau(\mathsf{u})$ will be the prefix forms of the parses of u.

Before moving on to compositions of morphisms and the rest of our motivating examples, let us make a series of remarks.

• The definition of morphism of grammars, as given above, appears out of the blue, and in somewhat austere generality. Admittedly, the definition, like most in the realm of algebra, is 'experimental', and driven by several, not easily formalizable criteria. It should cover enough cases of interest, seemingly not otherwise connected; it should possess good structural properties; and should have a family, or conceptual resemblance to other notions that have proved useful.

As for the instances of morphisms of grammars in mathematical syntax, I am hopeful this article provides quite a few. The desired structure theory is phrased in the language of categories; see below. As for family resemblances, there exist significant overlaps between the formalisms of tree transducers, term rewrite systems and context-free language transformations, discussion of which would take us far afield. Suffice it to say that the notion of morphism of grammars is most similar to (and in fact, properly contains) *synchronous context-free grammars* (SCFG); see e.g. Chapter 23 of [AS10]. SCFG are themselves notational variants of the *syntax-directed translation schemata* of Aho and Ullman [AU72]. The differences are quite significant:

- unlike SCFG, morphisms assume the existence of source and target grammars, their alphabets linked by a map α
- SCFG pair rules with rules; morphisms associate to each rule in the source grammar a parse tree in the target grammar
- in a SCFG, each re-indexing map is a permutation of non-terminal symbols; in a morphism, the re-indexing datum $\gamma(x \to \mathsf{s}, -)$ is a map from the *locations* of non-terminal symbols in $\beta(x \to \mathsf{s})$ to the *locations* of non-terminal symbols in s.

Thus, because of the presence of repeated variables, our motivating example (a) could not be handled by a SCFG. Nonetheless, it is fair to think of morphisms of grammars as syntax-directed translation schemes, boosted to their 'natural level of generality'.

• Recall that our grammars do not contain preferred start symbols; a morphism of grammars induces a multi-valued map

$$\hat{L}_{G_0}(x) \to \hat{L}_{G_1}\big(\alpha(x)\big)$$

for each non-terminal x in the alphabet $\mathcal{A}_0$ of G_0. It may well happen that for some $\mathsf{u} \in \mathcal{A}_0^*$, there exist distinct $x_0, x_1 \in N_0$ such that $\mathsf{u} \in \hat{L}_{G_0}(x_0)$ and $\mathsf{u} \in \hat{L}_{G_0}(x_1)$, and the translation(s) into $\hat{L}_{G_1}$ differ when u is considered as a descendant of x_0 from when it is considered a descendant of x_1.

• The language of iterated commutators, cf. Example 2.2, could be more succinctly defined with the help of a single non-terminal symbol expr and productions

$$\mathsf{expr} \to x_1 \mid x_2 \mid \ \ldots \ \mid x_i \mid \ \ldots$$
$$\mathsf{expr} \to [\mathsf{expr}, \mathsf{expr}]$$

However, as long as one prefers to put parentheses around compound expressions, but not around individual variables in the language of terms with infix operators $-$ and $\cdot$, one needs both of the syntactic categories var and expr in the target language. This, in turn, necessitates that the source language should distinguish variables from compound expressions; hence the more labored grammar G_0 of Example 2.2. This observation highlights that our morphisms are defined between context-free *grammars*, and are sensitive to the choice of grammar, even for unambiguous languages.

• What seems to be conspicuously missing from the definition of morphism is how the terminal symbols get translated. Indeed, the function α that is part of the morphism data goes from non-terminal symbols to non-terminal symbols. Of course, the function β is responsible for the translation of terminals, since terminals occurring in the language can be reached from the source non-terminal via productions. In fact, the reader may enjoy working the following out. Let T_0, T_1 be alphabets. Recall that any map $h : T_0 \to T_1^*$ induces a semigroup homomorphism $h : T_0^* \to T_1^*$. (The reuse of the letter 'h' should cause no confusion.) For a language $L \subseteq T_0^*$, h restricts to a map $h : L \to T_1^*$. Maps of this type are called literal homomorphisms.

Exercise. Let G_0 be a context-free grammar in the alphabet $N_0 \sqcup T_0$ and T_1 another set of terminals. Let $h : T_0 \to T_1^*$ be a map, inducing a literal homomorphism $h : L_{G_0}(x) \to T_1^*$ for each $x \in N_0$. Show that there exists a context-free grammar G_1 in the alphabet $N_0 \sqcup T_1$ and a morphism of grammars $G_0 \to G_1$ whose associated translation $\tau : L_{G_0}(x) \to L_{G_1}(x)$ is single-valued and satisfies $\tau(\mathsf{u}) = h(\mathsf{u})$ for all $\mathsf{u} \in L_{G_0}(x)$, any $x \in N_0$. (Hint: extend h to a semigroup homomorphism $(N_0 \sqcup T_0)^* \to (N_0 \sqcup T_1)^*$ by setting $h(x) = x$ for $x \in N_0$. α is the identity. Now let $\beta(x \to \mathsf{s}) = h(\mathsf{s})$.)

That is, any literal homomorphism can be induced by a morphism of grammars. Similarly, any rational transducer (thought of as a multi-valued mapping from its domain to its range, both being rational languages) can be encoded via a morphism of grammars. The details of this encoding are straightforward, but will be skipped here. It is unlikely that the notion of morphism of grammars will have anything to add to the very fine-tuned theory of rational transducers.

The next proposition is a simultaneous extension of Prop. 2.4 and of the defining property of the re-indexing map γ from the definition of morphism.

Proposition 2.7. *Let (α, β, γ) be a morphism of context-free grammars from G_0 to G_1, $x \in N_0$ and $T \in \mathrm{tree}_{G_0}(x)$. There is a natural map $\gamma(T, -)$ from $\mathrm{NT}(\tau(T))$ to $\mathrm{NT}(T)$ such that for any $t \in \mathrm{NT}(\tau(T))$,*

$$\alpha\big(\mathrm{label}(\gamma(T,t))\big) = \mathrm{label}(t) \ .$$

Proof. By induction on the depth of T. If $\mathrm{depth}(T) = 0$ then T consists of just the root $x \in N_0$, and $\tau(T)$ is the tree containing only the root $\alpha(x) \in N_1$. So $\mathrm{NT}(T) = \{x\}$ and $\mathrm{NT}(\tau(T)) = \{\alpha(x)\}$; $\gamma(T, -)$ is uniquely determined.

If $\mathrm{depth}(T) > 0$, recall how $\tau(T)$ is defined. Let $p \in G_0$ be the top production in T. As before, this induces subtrees T_s of T with roots $s \in \mathrm{NT}(p)$. For each $t \in \mathrm{NT}(\beta(p))$, graft the tree $\tau(T_{\gamma(p,t)})$ on $\beta(p)$ with t as root. $\tau(T)$ is defined to be the resulting tree.

Consider any $t \in \mathrm{NT}(\beta(p))$ and let $s = \gamma(p, t)$. Since $\mathrm{depth}(T_s) < \mathrm{depth}(T)$, by the induction hypothesis there is a map $\gamma(T_s, -)$ from $\mathrm{NT}(\tau(T_s))$ to $\mathrm{NT}(T_s)$, with α as left inverse to the action of $\gamma(T_s, -)$ on labels. When grafting $\tau(T_s)$ to $\beta(p)$, the domain of $\gamma(T_s, -)$ can be shifted with it, to become a subset of $\mathrm{NT}(\tau(T))$.

However, $\mathrm{NT}(\tau(T))$ is the disjoint union of the various $\mathrm{NT}(\tau(T_s))$ grafted to $\beta(p)$, with $s = \gamma(p, t)$, as t ranges over $\mathrm{NT}(\beta(p))$. $\gamma(T, -)$ can thus be defined as the disjoint union of the (appropriately shifted) maps $\gamma(T_s, -)$. □

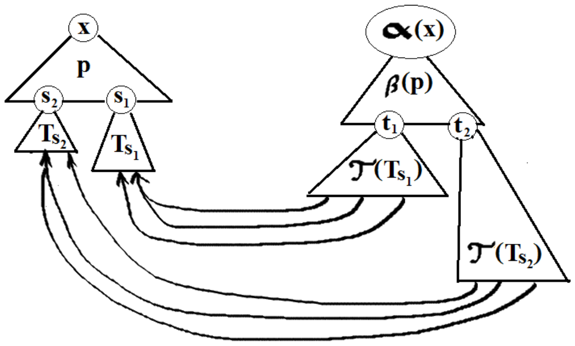


FIG. 2: Defining $\gamma(T,-)$. Above, p is a production (i.e. tree of depth 1), $\beta(p)$ is a tree, s_1, s_2, t_1 and t_2 are leaves labeled with non-terminal symbols such that $s_1 = \gamma(p, t_1)$ and $s_2 = \gamma(p, t_2)$.

Note that if T is a production $x \to \mathsf{s} \in G_0$ then $\gamma(T,-)$, as constructed above, coincides with $\gamma(x \to \mathsf{s}, -)$ that is part of the morphism data; there is thus no conflict of notation.

Observe also that when T is a parse tree of some string u containing only terminal symbols then the leaves of $\tau(T)$ cannot contain non-terminals either (since no map $\gamma(T,-)$ with the properties above could exist); so we indeed have an extension of Prop. 2.4.

Our choice of terminology insinuates that morphisms can be composed, and, with context-free grammars as objects, form a category. We will treat this next.

Definition 2.8. Let G_0, G_1, G_2 be context-free grammars, and let $(\alpha_{01}, \beta_{01}, \gamma_{01})$ be a morphism from G_0 to G_1, and $(\alpha_{12}, \beta_{12}, \gamma_{12})$ a morphism from G_1 to G_2. Define their composite

$$(\alpha_{02}, \beta_{02}, \gamma_{02}) = (\alpha_{01}, \beta_{01}, \gamma_{01}) \star (\alpha_{12}, \beta_{12}, \gamma_{12})$$

a morphism from G_0 to G_2, as follows:

α_{02} is the composite $N_0 \xrightarrow{\alpha_{01}} N_1 \xrightarrow{\alpha_{12}} N_2$.

Let $x \to \mathsf{s}$ (abbreviated as p) be a production in G_0. Set $\beta_{02}(p) = \tau_{12}\big(\beta_{01}(p)\big)$, where τ_{12} is the induced translation from $\mathrm{tree}_{G_1}(y)$ to $\mathrm{tree}_{G_2}\big(\alpha_{12}(y)\big)$, for $y \in N_1$. Note that $\beta_{02}(x \to \mathsf{s})$ is an element of $\mathrm{tree}_{G_2}(\alpha_{12}(\alpha_{01}(x)))$, i.e. of $\mathrm{tree}_{G_2}(\alpha_{02}(x))$, as required.

$\gamma_{02}(p,-)$ is to be a map from $\mathrm{NT}(\beta_{02}(p))$ to $\mathrm{NT}(p)$. It is defined as the composite

$$\mathrm{NT}\big(\tau_{12}(\beta_{01}(p))\big) \xrightarrow{\gamma_{12}(\beta_{01}(p),-)} \mathrm{NT}(\beta_{01}(p)) \xrightarrow{\gamma_{01}(p,-)} \mathrm{NT}(p)$$

More plainly, a production $p \in G_0$ is translated by β_{01} into a parse tree T_1 formed with G_1, which τ_{12} translates into a parse tree T_2 formed with G_2. The re-indexing map $\gamma_{12}(\beta_{01}(p),-)$ goes from leaves of T_2 labeled with non-terminal symbols to leaves of T_1 labeled with non-terminal symbols, followed by the re-indexing map $\gamma_{01}(p,-)$ from leaves of T_1 labeled with non-terminal symbols, to leaves (i.e. letters on the right-hand side) of the production p that are non-terminal symbols.

As a continuation of Example 2.6, it is instructive at this point to construct grammars G_0, G_1, G_2 for prefix resp. postfix resp. fully parenthesized infix terms of binary function symbols $\circledast$ and $\boxtimes$, and morphisms $G_i \to G_j$ ($i, j \in \{0, 1, 2\}$) that form a commutative diagram of isomorphisms. Of course, one expects more: a commutative diagram of (iso)morphisms of grammars should induce a commutative diagram of (bijective) mappings between the associated languages. That is indeed so. To prove it, we need a key structural property of τ. By definition, the translation $\tau(T)$ of a parse tree T can be generated by attaching to the translation of the top production in T the translations of the sub-trees of the top production — appropriately re-indexed. The next lemma states that the same recipe applies if one separates any top segment, not necessarily just the top production, of the input tree. The necessary re-indexing is supplied by Prop. 2.7.

Lemma 2.9. *Let $x \in N_0$ and $T \in \mathrm{tree}_{G_0}(x)$. For each $s \in \mathrm{NT}(T)$, suppose given $U_s \in \mathrm{tree}_{G_0}(\mathrm{label}(s))$. Let $T_U \in \mathrm{tree}_{G_0}(x)$ be the result of grafting each U_s to s as root. Now, for each $t \in \mathrm{NT}(\tau(T))$, graft $\tau(U_{\gamma(T,t)})$ to $\tau(T)$ with t as root. Let $\tau(T)_{\tau(U)} \in \mathrm{tree}_{G_1}(\alpha(x))$ be the resulting tree. Then $\tau(T_U) = \tau(T)_{\tau(U)}$.*

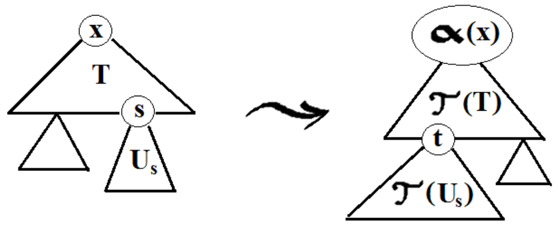


FIG. 3: Modularity of $\tau(T)$. s and t are leaves labeled with non-terminal symbols such that $s = \gamma(p, t)$. x and $\alpha(x)$ are labels of the roots. Note the similarity with Fig. 1.

Proof. By induction on $\mathrm{depth}(T)$. When $\mathrm{depth}(T) = 0$, the lemma is a tautology. When $\mathrm{depth}(T) = 1$, it is the inductive step in the definition of τ (applied to the tree T_U, whose top production is T).

If $\mathrm{depth}(T) > 1$, let $p \in G_0$ be the top production in T. As before, this induces subtrees T_r of T with roots $r \in \mathrm{NT}(p)$. The set of leaves of T with non-terminal labels, $\mathrm{NT}(T)$, is the disjoint union of $\mathrm{NT}(T_r)$ as r ranges over $\mathrm{NT}(p)$. For each $r \in \mathrm{NT}(p)$, let $T_{r,U}$ be the tree that results from grafting U_s to s for each $s \in \mathrm{NT}(T_r)$. $T_{r,U}$ is thus the same as the subtree of T_U with r as root.

$\tau(T_U)$ (by the inductive step in the definition of τ) is the result of grafting $\tau(T_{\gamma(p,v),U})$ to v, for v ranging over $\mathrm{NT}(\beta(p))$. Pick such a $v \in \mathrm{NT}(\beta(p))$ and let $r = \gamma(p, v)$. Since $\mathrm{depth}(T_{r,U}) < \mathrm{depth}(T)$, by the induction hypothesis $\tau(T_{r,U})$ is the same as the result of grafting, for each $t \in \mathrm{NT}(\tau(T_r))$, $\tau(U_{\gamma(T_r,t)})$ to t as root. As v ranges over $\mathrm{NT}(\beta(p))$, this assembles to the same tree as $\tau(T)$ with $\tau(U_{\gamma(T,t)})$ grafted to t for each $t \in \mathrm{NT}(\tau(T))$. But that is the same as $\tau(T)_{\tau(U)}$ by definition, completing the induction step. □

Proposition 2.10. *If G_0, G_1, G_2 are context-free grammars and $(\alpha_{01}, \beta_{01}, \gamma_{01}) : G_0 \to G_1$ resp. $(\alpha_{12}, \beta_{12}, \gamma_{12}) : G_1 \to G_2$ morphisms of grammars, with composite $(\alpha_{02}, \beta_{02}, \gamma_{02}) : G_0 \to G_2$ and associated translation functions τ_{01}, τ_{12} and τ_{02}. Then for all $x \in N_0$ and $T \in \mathrm{tree}_{G_0}(x)$,*

$$\tau_{12}\big(\tau_{01}(T)\big) = \tau_{02}(T) \quad \text{in } \mathrm{tree}_{G_2}(\alpha_{02}(x)) \ .$$

Proof. When $\mathrm{depth}(T) = 0$, this reduces to $\alpha_{12}\big(\alpha_{01}(x)\big) = \alpha_{02}(x)$. When $\mathrm{depth}(T) > 0$, let p be the top production in T, inducing subtrees T_s with roots $s \in \mathrm{NT}(p)$ as before. $\tau_{01}(T)$, by definition, is the result of grafting $\tau_{01}(T_{\gamma_{01}(p,t)})$ to t for each $t \in \mathrm{NT}(\beta_{01}(p))$. τ_{12} of that composite tree, by Lemma 2.9, is the result of grafting $\tau_{12}\big(\tau_{01}(T_{\gamma_{01}(p,t)})\big)$, with $t = \gamma_{12}(\tau_{12}(\beta_{01}(p)), r)$, to $r \in \mathrm{NT}(\tau_{12}(\beta_{01}(p)))$. But that is the same as the translation of T under τ_{02}, by definition of the composite of two morphisms. □

Proposition 2.11. *The composition of morphisms of context-free grammars is associative. That is, if G_i ($i = 0, 1, 2, 3$) are context-free grammars, and $\mu_{i,i+1} = (\alpha_{i,i+1}, \beta_{i,i+1}, \gamma_{i,i+1})$ morphisms from G_i to G_{i+1} (here $i = 0, 1, 2$) then*

$$\mu_{01} \star (\mu_{12} \star \mu_{23}) = (\mu_{01} \star \mu_{12}) \star \mu_{23} \ .$$

Proof. The component α_{03} of $G_0 \to G_3$ is the composite

$$N_0 \xrightarrow{\alpha_{01}} N_1 \xrightarrow{\alpha_{12}} N_2 \xrightarrow{\alpha_{23}} N_3 \ .$$

As regards β_{03} : given $p \in G_0$, $\mu_{01} \star (\mu_{12} \star \mu_{23})$ associates to it $\tau_{13}\big(\beta_{01}(p)\big)$, while $(\mu_{01} \star \mu_{12}) \star \mu_{23}$ sends it to $\tau_{23}\big(\beta_{02}(p)\big)$. But both of those equal $\tau_{23}\big(\tau_{12}(\beta_{01}(p))\big)$, by Prop. 2.10.

Finally, $\gamma_{03}(p, -)$, computed either way, is the composite

$$\mathrm{NT}\big(\tau_{23}\big(\tau_{12}(\beta_{01}(p))\big)\big) \xrightarrow{\gamma_{23}(\tau_{12}(\beta_{01}(p)),-)} \mathrm{NT}\big(\tau_{12}(\beta_{01}(p))\big) \xrightarrow{\gamma_{12}(\beta_{01}(p),-)} \mathrm{NT}(\beta_{01}(p)) \xrightarrow{\gamma_{01}(p,-)} \mathrm{NT}(p)$$

□

Definition 2.12. Let CFG be the category whose objects are context-free grammars, with morphisms defined by Prop. 2.1 and composition defined by Prop. 2.8. The identity morphism on G is given by $(\mathrm{id}_N, \mathrm{id}_G, \mathrm{id}_{\mathrm{NT}(p)})$, i.e. identity maps.

We are now ready to assemble Prop. 2.3, Cor. 2.5, Prop. 2.10 and Prop. 2.11 into the main theorem of this paper. Intuitively, it says that tree is a functor from CFG to the category of sets. However, since we did not include a preferred start symbol in the data for context-free grammars (and much less did we assume that any such symbol would be preserved by morphisms), the target category is slightly more complicated. Let $\mathrm{Mor}(\mathit{Set})$ be the category of maps of sets. An object of $\mathrm{Mor}(\mathit{Set})$ is thus a function $f : X \to Y$ between arbitrary sets; a morphism from $f_1 : X_1 \to Y_1$ to $f_2 : X_2 \to Y_2$ consists of maps $u : X_1 \to X_2$ and $v : Y_1 \to Y_2$ such that

$$\begin{tikzcd} X_1 \arrow[r, "u"] \arrow[d, "f_1"'] & X_2 \arrow[d, "f_2"] \\ Y_1 \arrow[r, "v"] & Y_2 \end{tikzcd}$$

commutes. Morphisms are composed ‘horizontally’. $\mathrm{Mor}(\mathit{Set})$ is an example of a diagram category (see e.g. MacLane [CWM]), but an alternative way to think of it is as the category of sets fibered over a base: $f : X \to Y$ can be thought of as the family of sets $f^{-1}(y)$ with $y \in Y$. Morphisms are then fiberwise maps.

Theorem 2.13. (a) tree *is a functor* CFG $\to \mathrm{Mor}(\mathit{Set})$. *It associates to a context-free grammar G the family of sets $\{\mathrm{tree}_G(x) \mid x \in N\}$. To a morphism of grammars $G_0 \to G_1$ it* *associates the map of families $\alpha : N_0 \to N_1$ and $\tau : \mathrm{tree}_{G_0}(x) \to \mathrm{tree}_{G_1}(\alpha(x))$, where $x \in N_0$.*

(b) *Let* UCFG *be the full subcategory of* CFG *whose objects are the unambiguous context-free grammars. $\hat{L}$ is a functor* UCFG $\to \mathrm{Mor}(Set)$. *It associates to a context-free grammar G the family of sets $\{\hat{L}_G(x) \mid x \in N\}$. To a morphism of grammars $G_0 \to G_1$ it associates the map of families $\alpha : N_0 \to N_1$ and $f : \hat{L}_{G_0}(x) \to \hat{L}_{G_1}(\alpha(x))$ with $x \in N_0$, that sends* $\mathsf{u} \in \hat{L}_{G_0}(x)$ *to the sentential form generated by $\tau(T(\mathsf{u}))$, where $T(\mathsf{u})$ is the (unique) parse of* u.

(c) *L, sending G to the family $\{L_G(x) \mid x \in N\}$, is a subfunctor of $\hat{L}$.*

There exists a well-understood interplay between rational languages, finite state automata, and monoid objects in categories; the canonical reference is Arbib [AA69]. Category-theoretic properties of CFG (for example, the existence of pullbacks, filtered colimits or coproducts) as well as the roles that morphisms, functors, natural transformations etc. may play in formal language theory at higher levels of the Chomsky hierarchy, are much less explored.

3. LOOKING AHEAD

We have only dealt with two of the motivating examples. Neither of the other two can be described by a morphism $G \to G$ where G is any of the usual unambiguous context-free grammars for first order logic, or, I suspect, any context-free grammar for it. It should come as no surprise that there are limitations to the 'word processing power' of morphisms, as defined above. One expects that there exists a hierarchy of mappings between context-free grammars, just as there are hierarchies of languages, complexity classes, and so on. The goal of this final — much more speculative — section is to sketch further levels of this hierarchy. But first, here is one expression of the structural limitations of morphisms.

Proposition 3.1. *Suppose $(\alpha, \beta, \gamma) : G_0 \to G_1$ is a morphism of grammars with the property that for some constant K,*

$$\mathrm{depth}(\beta(p)) \leqslant K$$

for all $p \in G_0$. Then for all $x \in N_0$ and $T \in \mathrm{tree}_{G_0}(x)$,

$$\mathrm{depth}(\tau(T)) \leqslant K \cdot \mathrm{depth}(T)\ .$$

The proof is by induction on $\mathrm{depth}(T)$. Note that such a bound K always exists if G_0 is finite; however, our grammars (and alphabets) were not assumed to be so by default.

Example 3.2. Let L be the language of function terms for an associative binary operation (denoted by juxtaposition), fully parenthesized, with infinitely many variables available. The alphabet is

$$N = \{ \ \mathsf{expr} \ \}$$
$$T = \{ \ (\,)\, x_1\, x_2\, \ldots\, x_i\, \ldots \}$$

with unambiguous grammar

$$\mathsf{expr} \to x_1 \mid x_2 \mid \ \ldots \ \mid x_i \mid \ \ldots$$
$$\mathsf{expr} \to (\mathsf{expr}\,\mathsf{expr})$$

Let $\tau : L \to L$ be the mapping that sends an expression to its leftmost-parenthesized equivalent. For example,

$$((x_5x_3)((x_1x_3)x_2))$$

is to be sent to

$$((((x_5x_3)x_1)x_3)x_2)$$

If there was a morphism of grammars $(\alpha, \beta, \gamma) : G \to G$ inducing τ, it would have to satisfy

$$\beta(\mathsf{expr} \to x_i) = x_i$$

for all $i = 1, 2, \ldots$. Since there is only one other production in the grammar, namely,

$$\mathsf{expr} \ \to \ (\mathsf{expr}\,\mathsf{expr})$$

Prop. 3.1 would apply. However, for any positive integer d, let T be the term in variables $x_1, x_2, \ldots, x_{2^d}$ whose parse tree (ignoring parentheses) is the complete binary tree of depth d; e.g. for $d = 3$:

$$(((x_1x_2)(x_3x_4))((x_5x_6)(x_7x_8)))$$

$\tau(T)$ is a left-branching tree, with depth 2^d.

$$\left\{\frac{\mathrm{depth}(\tau(T))}{\mathrm{depth}(T)} \ \mid \ T \in \mathrm{tree}_G(\mathsf{expr})\right\}$$

is thus unbounded, and the mapping τ cannot correspond to any morphism of grammars.

This argument does not apply to our motivating example (c), replacement of free occurrences of a variable x in the input formula ϕ by some term t, since

$$\mathrm{depth}\left(\tau_{x\to\mathsf{t}}(\phi)\right) \leqslant \mathrm{depth}(\phi) + \mathrm{depth}(\mathsf{t})$$

always. (We have silently fixed an unambiguous context-free grammar G for first order logic.) However, no morphism $G \to G$ induces $\tau_{x\to\mathsf{t}}(\phi)$. The recursive rules

$$\tau_{x\to\mathsf{t}}(\phi \wedge \psi) \ \Rightarrow \ \tau_{x\to\mathsf{t}}(\phi) \wedge \tau_{x\to\mathsf{t}}(\psi)$$
$$\tau_{x\to\mathsf{t}}(\forall y\phi) \ \Rightarrow \ \forall y\tau_{x\to\mathsf{t}}(\phi)$$

showing that replacement descends the parse tree along boolean connectives and quantification with respect to variables other than x, conform perfectly to the combinatorial possibilities of a self-morphism of G. However, one has

$$(*) \qquad \tau_{x\to\mathsf{t}}(\forall x\phi) \ \Rightarrow \ \forall x\phi$$

since all free occurrences of x in ϕ become bound in $\forall x\phi$. $\tau_{x\to\mathsf{t}}(\forall x\phi)$ is thus not a function of $\tau_{x\to\mathsf{t}}(\phi)$, since ϕ cannot in general be reconstructed from $\tau_{x\to\mathsf{t}}(\phi)$. So $\tau_{x\to\mathsf{t}}$ cannot be computed by bottom-up induction, whereas translations induced by morphisms can always be.

Intuitively, a morphism of grammars applies the *same* functional transformation (itself!), iteratively, to subtrees of the input tree, whereas $(*)$ calls on a different transformation (namely, the identity) when the input has the form $\forall x\phi$. Recall that two functions $f, g : \mathbb{N} \to \mathbb{N}$ are defined by simultaneous recursion if $f(0)$ and $g(0)$ are given, and there exist functions F and G such that for $n > 0$,

$$f(n) = F\big(n, f(n-1), g(n-1)\big)$$
$$g(n) = G\big(n, f(n-1), g(n-1)\big) \ .$$

In the presence of a pairing function that codes the ordered pair $\langle f(n), g(n)\rangle$ as a single natural number, simultaneous recursion can be replaced by ordinary recursion. However, for tree transformations, simultaneous recursion on syntax has more expressive power than simple recursion.

Definition 3.3. Let G_0 and G_1 be context-free grammars in the alphabets N_0, T_0, N_1, T_1 as usual, and k a positive integer. A *k-morphism from G_0 to G_1 defined by simultaneous recursion* consists of the following data:

- mappings $\alpha_i : N_0 \to N_1$ for $i = 1, 2, \ldots, k$
- mappings β_i, for $i = 1, 2, \ldots, k$, assigning to each production $x \to \mathsf{s} \in G_0$ a parse tree from $\mathrm{tree}_{G_1}(\alpha_i(x))$
- for each $i = 1, 2, \ldots, k$ and each production $p \in G_0$, a function $\gamma_i(p, -)$ from $\textsc{nt}(\beta_i(p))$ to $\textsc{nt}(p)$ and a function $\delta_i(p, -)$ from $\textsc{nt}(\beta_i(p))$ to $\{1, 2, \ldots, k\}$, with the property that for all $i = 1, 2, \ldots, k$ and all $t \in \textsc{nt}(\beta_i(p))$, writing $j = \delta_i(p, t)$,
$$\alpha_j\big(\,\mathrm{label}(\gamma_i(p, t))\big) = \mathrm{label}(t) \ .$$

A k-morphism is, roughly, a k-tuple of grammatical transformations that are intertwined via the function δ: the i-th transformation can call on the j-th transformation to act on a subtree of the input tree. The maps α_i provide the initial values. There is no circular dependency, since each recursive call applies to a lower-level *subtree* of the input tree. More precisely,

Proposition 3.4. *A k-morphism of grammars from G_0 to G_1 induces, for each $i = 1, 2, \ldots, k$ and $x \in N_0$, a mapping*
$$\tau_i : \mathrm{tree}_{G_0}(x) \to \mathrm{tree}_{G_1}(\alpha_i(x)) \ .$$

Proof. For $T \in \mathrm{tree}_{G_0}(x)$, define the $\tau_i(T) \in \mathrm{tree}_{G_1}(\alpha_i(x))$ simultaneously by induction on the depth of T:

• If $\mathrm{depth}(T) = 0$, then T must be x itself, and $\tau_i(T)$ is defined to be $\alpha_i(x)$.
• If $\mathrm{depth}(T) > 0$, let $x \to \mathsf{s} \in G_0$ be the top production in T. Write p for $x \to \mathsf{s}$ for brevity. As usual, $\textsc{nt}(p)$ can be identified with a subset of s, the locations of the non-terminal symbols in s. Since G_0 is context-free, each $s \in \textsc{nt}(p)$ induces a subtree T_s of T with s as root. For each $i = 1, 2, \ldots, k$ and $t \in \textsc{nt}(\beta_i(p))$, writing $j = \delta_i(p, t)$, graft the tree $\tau_j(T_{\gamma_i(p,t)})$ on $\beta_i(p)$ with t as root. $\tau_i(T)$ is defined to be the resulting tree.

Since $\mathrm{depth}(T_s) < \mathrm{depth}(T)$ for all $s \in \textsc{nt}(p)$, $\tau_j(T_s)$ is defined by the induction hypothesis. Note that $\tau_j(T_s)$ belongs to $\mathrm{tree}_{G_1}(\alpha_j(\mathrm{label}(s))$ by the induction assumption, and $\alpha_j\big(\,\mathrm{label}(\gamma_i(p, t))\big) = \mathrm{label}(t)$ by Def. 3.3. That is, the non-terminal symbol at the root of $\tau_j(T_{\gamma_i(p,t)})$ coincides with the non-terminal symbol at the location t. Since G_1 is a context-free grammar, the graft is well-defined, and $\tau_i(T)$ will belong to $\mathrm{tree}_{G_1}(\alpha_i(x))$ as desired. □

When finding $\tau_i(T)$ by recursion from root to leaves on T, one can restrict to computing $\tau_j(T_s)$ only for those subtrees T_s of T and values $j \in \{1, 2, \ldots, k\}$ that are called for by the indexing function δ. When using bottom-up induction, the entire k-tuple of values $\big(\tau_1(-), \tau_2(-), \ldots, \tau_k(-)\big)$ needs to be computed for all subtrees of T.

Mutatis mutandis, the results of the previous section, from Prop. 2.3 to Prop. 3.1, remain valid for morphisms defined by simultaneous recursion. The composition of a k-morphism from G_0 to G_1 and n-morphism from G_1 to G_2 will be a $k \cdot n$-morphism from G_0 to G_2. Composition is associative, and tree_G becomes a functor from CFG to tuples of functions of sets. The details, while not conceptually complicated, are quite tedious (largely for notational reasons) and will not be needed here.

The reader is invited to define the pair of transformations $(\tau_{x \to \mathtt{t}}, \mathrm{id})$ by simultaneous recursion on the syntax of first order logic. $\tau_{x \to \mathtt{t}}$ calls itself and id, while the identity transformation calls itself only. The fact that the treatment of descendant nodes is inherited from their parent nodes is reminiscent of attribute grammar.

Note that $\phi_{x \to \mathtt{t}}$, replacing all free occurrences of the variable x in the formula ϕ by the term $\mathtt{t}$, is the least complicated of the multitude of operations involving variable replacement and binding. If a free variable in $\mathtt{t}$ is captured by a quantifier in ϕ, then ϕ will no longer imply its instance $\phi_{x \to \mathtt{t}}$; to preserve the intended logical meaning, the dummy variable appearing in the capturing quantifier in ϕ should be renamed first, to a variable not occurring in ϕ or $\mathtt{t}$. However, the function that returns a variable *not* occurring in a given formula does not have a canonical value, and is not easily describable in terms of language operations. A related, and much researched, issue is the formalization of *explicit substitution* in lambda calculi [ES90]: under explicit substitution, the operation $x \to \mathtt{t}$ does not belong to the meta-language, but is part of the language itself. On the other hand, there seem to exist few studies, from the viewpoint of mathematical linguistics, of the syntax of substitutions through de Bruijn indices or Bourbaki's variable-free notation [TL99].

Prop. 3.1 does not apply either to our fourth (and last) motivating example, transforming first order formulas ϕ to negation normal form $\mathrm{NNF}(\phi)$, since $\mathrm{depth}(\mathrm{NNF}(\phi)) \leqslant \mathrm{depth}(\phi)$ always. But NNF cannot be induced by a morphism, or in fact k-morphism. The standard context-free grammars of first order logic contain the production

$$\mathsf{expr} \to \urcorner\, \mathsf{expr}$$

But $\beta(\mathsf{expr} \to \urcorner\, \mathsf{expr})$ cannot contain any terminal symbols; any non-terminal other than 'expr'; or more than one copy of 'expr': each of those possibilities would be inconsistent with the fact that $\mathrm{NNF}(\urcorner\urcorner\phi) = \mathrm{NNF}(\phi)$. So $\beta(\mathsf{expr} \to \urcorner\, \mathsf{expr})$ is forced to be 'expr', which of course is incompatible with the negation normal form of $\urcorner\phi$ for atomic ϕ.

Intuitively, the issue is that the rewrite rules

$$\begin{aligned} \neg\neg\phi &\Rightarrow \phi \\ \neg(\phi \wedge \psi) &\Rightarrow \neg\phi \vee \neg\psi \\ \neg(\phi \vee \psi) &\Rightarrow \neg\phi \wedge \neg\psi \\ \neg\forall x\phi &\Rightarrow \exists x \neg\psi \\ \neg\exists x\phi &\Rightarrow \forall x \neg\psi \end{aligned}$$

may introduce new instances of the negation symbol on their right hand sides. It requires a moment of thought to verify that this set of rules is *noetherian* (starting with any formula, they cannot be applied infinitely often) and many more moments of thought to verify that they are *confluent* (every formula has a unique negation normal form, even if the above rewrite rules are applied to arbitrary *subformulas* first, in any order, until no rule applies anywhere).

Recall that a *term rewrite system* (TRS) is an unordered set of rewrite rules acting on function terms in some fixed signature. A TRS is called convergent if it is both noetherian and confluent [TR98]. All four motivating examples, and Example 3.2 as well, belong to the family of convergent TRS, adapted from the unambiguous grammar of function terms to the general setting of context-free grammars. The fact that the effect of morphisms on parse trees can be computed by both bottom-up and top-down recursion, as well as Lemma 2.9, can be seen as corollaries of confluence.

It is quite challenging, however, to fashion a category out of convergent TRS. To begin with, neither the confluence nor the noetherianness of TRS is, in general, decidable (though, curiously, the confluence of noetherian TRS *is* decidable). Secondly, a famous example due to Toyama shows that the disjoint union of two convergent TRS need not be convergent. Thus the composite of two TRS cannot, in general, be defined as the disjoint union of their underlying rules. There exist, however, sufficient conditions for the modularity of convergence for TRS. Alternatively, one can experiment with ordered (prioritized) rewriting rules.

In a different direction, the notion of morphism of context-free grammars could be broadened to allow for non-determinism: several right-hand sides of the component β. Finally, the focus on parse trees is, to some extent, restrictive: the domain of these transformations could be any set of node-labeled rooted trees closed under taking subtrees.

I hope to elaborate some of these ideas in later publications. In closing, let me return to the quote from Chomsky that opened this article. *Suppose* that the only gift linguistics ever gave mathematics was, indeed, the notion of context-free grammar. Let's play with this present: expand the focus from context-free grammars to maps of context-free grammars (from objects to morphisms) and I think we will agree that linguistics has given mathematics a gift that keeps on giving.

References

[AA69] M. Arbib: Theories of abstract automata. Prentice–Hall, 1969

[AS10] Algorithms and Theory of Computation Handbook. Vol 2: Special Topics and Techniques. Ed. by M. Atallah and M. Blanton. 2nd ed., Chapman & Hall, 2010

[AU72] A. Aho and J. Ullman: The Theory of Parsing, Translation, and Compiling. Vol 1: Parsing. Prentice–Hall, 1972

[CWM] S. MacLane: Categories for the Working Mathematician. 2nd ed., Springer–Verlag, 1998

[ES90] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy: Explicit substitutions. Digital Systems Research Center Technical Report SRC-RR-54, 1990

[GE82] *Noam Chomsky on The Generative Enterprise: A Discussion with Piny Huybregts and Henk van Riemsdijk*, Foris Publications, Dordrecht–Holland 1982

[GE04] *The Generative Enterprise Revisited: Discussions with Riny Huybregts, Henk Van Riemsdijk, Naoki Fukui, and Mihoko Zushi.* With a New Foreword by Noam Chomsky. de Gruyter, 2004

[M10] E. Mendelson: Introduction to Mathematical Logic. 5th edition, Chapman & Hall, 2010

[TL99] A.R.D. Mathias: A term of length 4,523,659,424,929. Preprint, 1999
Available at `www.dpmms.cam.ac.uk/~ardm/inefff.pdf`

[TR98] F. Baader and T. Nipkow: Term Rewriting and All That. Cambridge University Press, 1998

Department of Mathematics, University of Massachusetts, Lowell, One University Avenue, Lowell, MA 01854

E-mail address: `tibor_beke@uml.edu`