$\mathcal{L}_0 = h_e(\mathcal{L}_2 \cap \mathcal{L}_2)$
A Most Wonderful Theorem

Dick Grune
VU University Amsterdam

July 1, 2014

Abstract

Any phrase-structure (Type 0) language can be obtained by intersecting two context-free (Type 2) languages and applying an erasing homomorphism to the result. This surprising and interesting theorem is treated only very formally in the existing literature. This article gives a more informal description.

1 Introduction

(for those who wonder what this is all about...)

In its most abstract form a "language" is a usually infinitely large collection of sentences; the only property of a sentence is whether it is in the language or not. This makes a description of a language infinitely large, which is awkward, and humanity restricted itself rapidly to languages that have finite descriptions. But even finite descriptions do not always define a language; for example, "the language whose sentences are not combinations of other sentences in the language" has a finite description but is not well-defined, and the awkwardness remains.

One solution is to restrict oneself to languages that can be *generated by finite recipes* rather than *described by finite texts* (by "generating a language" we mean making all its sentences, sentence by sentence.) Chomsky proposed a grammatical rewriting system for these recipes, and introduced four levels of increasing simplification on them: Type 0 to Type 3, generating the "phrase-structure", "context-sensitive", "context-free", and "regular" languages, respectively. Although these grammars are all very simple on paper, it turns out that Type 0 and Type 1 grammars are still too difficult to be of much use. Only Type 2 grammars (more conveniently called CF grammars) and Type 3 grammars and their languages are within the grasp of human understanding, and much of computer science is based on them.

It therefore comes as quite a surprise that any Type 0 language $\mathcal{L}_0$ (defined by a simple but inscrutable Type 0 grammar) can be generated using only two CF languages, $\mathcal{L}_{2_P}$ and $\mathcal{L}_{2_I}$ and two bits of simple math. Even better, given the grammar for the inscrutable Type 0 language $\mathcal{L}_0$, the grammars for the simple CF languages $\mathcal{L}_{2_P}$ and $\mathcal{L}_{2_I}$ can be constructed easily, even by hand.

The two bits of simple math are "intersection" and "erasure". The *intersection* of two languages is a new language which contains only those sentences that are present in both languages. The intersection of two languages X and Y is written as $X \cap Y$. An *erasure* (officially an *erasing homomorphism*) on a language is an operation that takes a list of letters and erases them in all sentences in the language. Vwl-rsr n th nglsh lngg wld mk t lk lk ths.[1]

Now we are in a position to read the title of this paper. A Type 0 language ($\mathcal{L}_0$) is equal to the intersection of two CF languages ($\mathcal{L}_{2_P} \cap \mathcal{L}_{2_I}$) to which an erasure h_e has been applied, all for the proper $\mathcal{L}_0$, $\mathcal{L}_{2_P}$, $\mathcal{L}_{2_I}$, and h_e. How the proper $\mathcal{L}_{2_P}$, $\mathcal{L}_{2_I}$, and h_e are constructed for a given $\mathcal{L}_0$, is detailed below.

2 The Construction

The construction (and with it the proof of the theorem) was first published in 1967 by Ginsburg *et al.* [1], pp. 402-405; a detailed and formal treatment can be found in Harrison [2], pp. 307-311.

We recall that a terminal production of a grammar G (i.e. a sentence in the language defined by G) is obtained by starting from the start symbol of G, and then replacing left-hand sides of grammar rules in G by their corresponding right-hand sides, until only terminals symbols remain:

$$S \overset{S\to\beta_i}{\Rightarrow} X_1^+,\ X_1^+ \overset{\alpha_{i1}\to\beta_{i1}}{\Rightarrow} X_2^+,\ X_2^+ \overset{\alpha_{i2}\to\beta_{i2}}{\Rightarrow} X_3^+,\quad \ldots\ X_k^+ \overset{\alpha_{ik}\to t^*_{ik}}{\Rightarrow} t^* \tag{1}$$

Here S is the start symbol; X is a non-terminal or terminal symbol; t is a terminal symbol; and a production step $X^+ \overset{\alpha\to\beta}{\Rightarrow} X^+$ means that the left X^+ contains α and this α is replaced by β in the right X^+, with $\alpha \to \beta$ a rule in G. The sequence (1) describes the complete production process $S\big(\overset{\alpha\to\beta}{\Rightarrow}\big)^+ t^*$.

The basic idea for the construction of the two CF grammars L_P and L_I is to have the sentences in the intersection between them represent complete production processes of G. A production step using the rule $\alpha_i \to \beta_i$ is encoded thus:

$$\langle X_l^* \alpha_i X_r^* \Rightarrow \overleftarrow{X_r^*} \overleftarrow{\beta_i} \overleftarrow{X_l^*} \rangle \tag{2}$$

where the $\Rightarrow$, $\langle$ and $\rangle$ are unique markers; X_l^* (the left environment of α_i) and X_r^* (the right environment of α_i) are arbitrary sequences of terminals and non-terminals; and $\overleftarrow{X^*}$ means the reverse of X^*.

The language L_P consists of all sequences of production steps of the form (2), provided that the first step starts with the start symbol of G. Since it is easy to produce $Sx\overleftarrow{S}$ by nesting in a CF grammar, a form like (2) is also easy to produce: apply nesting twice, once for the X_r^* and once for X_l^*.

However, such a grammar produces sequences like

$$\langle S \overset{S\to\beta}{\Rightarrow} \overleftarrow{X_1^+} \rangle\ \langle X_2^+ \overset{\alpha\to\beta}{\Rightarrow} \overleftarrow{X_3^+} \rangle\ \langle X_4^+ \overset{\alpha\to\beta}{\Rightarrow} \overleftarrow{X_5^+} \rangle\quad \ldots \tag{3}$$

in which each step is unrelated to the previous one. For a sequence of steps to be a good production sequence, the right-hand X^+ of one step must be equal to the

[1] Vowel-erasure on the English language would make it look like this.

left-hand X^+ of the next, as in (1) above. More precisely, $X^*_{l(k+1)}\alpha_{(k+1)}X^*_{r(k+1)}$ must be equal to $\overleftarrow{X^*_{rk}\beta_k X^*_{lk}}$.

This is where the language L_I comes in. It consists of sequences of "connectors"

$$\overleftarrow{X^+}\rangle\ \langle X^+\Rightarrow$$

which connect $\overleftarrow{X^*_{rk}\beta_k X^*_{lk}}$ to $X^*_{l(k+1)}\alpha_{(k+1)}X^*_{r(k+1)}$ in the proper way. These connectors are easy to construct using a CF grammar since their structure is essentially $\overleftarrow{S}\,xSy$.

To make them work the sequence of connectors is positioned so that upon intersection with a sentence from L_P, the markers $\Rightarrow$, $\langle$ and $\rangle$ line up in both sentences:

$$\begin{array}{rll} \ldots\ \langle\ldots\ \overleftarrow{X^*_{rk}\beta_k X^*_{lk}}\rangle & \langle X^*_{l(k+1)}\alpha_{(k+1)}X^*_{r(k+1)}\Rightarrow\ldots\rangle\ \ldots & \text{from } L_P \\ \ldots\ \overleftarrow{X^+}\rangle & \langle X^+\Rightarrow\ldots & \text{from } L_I \end{array}$$

This forces two equalities:

$$\overleftarrow{X^*_{rk}\beta_k X^*_{lk}} = \overleftarrow{X^+}$$
$$X^*_{l(k+1)}\alpha_{(k+1)}X^*_{r(k+1)} = X^+$$

Reversing both sides of the first equality gives

$$\overleftarrow{\overleftarrow{X^*_{rk}\beta_k X^*_{lk}}} = X^+$$

which, combined with the second equality, yields

$$\overleftarrow{\overleftarrow{X^*_{rk}\beta_k X^*_{lk}}} = X^*_{l(k+1)}\alpha_{(k+1)}X^*_{r(k+1)}$$

as required. This makes the intersection of sentences in L_P and sentences in L_I representations of a valid Type 0 production process – almost, since there are three more details to be taken care of: start-up, close-down, and harvesting the result.

A production sequence must start with the start symbol, so the first step in any sentence in L_P must be $\langle S\Rightarrow\overleftarrow{\beta_i}\rangle$ for some rule $S\rightarrow\beta_i$ in G. But the sequence of connectors $\overleftarrow{X^+}\rangle\ \langle X^+\Rightarrow$ in L_I does not start with $\langle S\Rightarrow$. This is remedied by prefixing the connector sequences by $\langle X^+\Rightarrow$, in conformity with the real connectors. After these additions the heads of sentences in L_P and L_I match for correctly starting production sequences in G.

A production sequence is correct only if the X^* after the last $\Rightarrow$ consists of terminal symbols only. L_P cannot enforce this (it is hard to identify the last $\alpha\rightarrow\beta$ and impossible to construct the proper left and right environments for it), but L_I can: terminate all sequences of connectors with the form $t^*\rangle$. Now the tails of sentences in L_P and L_I match for correctly ending production sequences in G.

How to harvest the result is less obvious. Since sentences in the intersection look like this:

$$\langle {X^+}_0\Rightarrow\overleftarrow{{X^+}_1}\rangle\ \langle {X^+}_1\Rightarrow\overleftarrow{{X^+}_2}\rangle\ \langle {X^+}_2\Rightarrow\overleftarrow{{X^+}_3}\rangle\quad\ldots\ \langle {X^+}_k\Rightarrow\overleftarrow{t^*}\rangle$$

it seems easy enough to just erase the X^+s and the markers, and t^* remains, albeit in reverse order. But X may be N *or* T, and spurious terminal symbols would remain from all over the production sequence.

The first step towards the solution is to replace each terminal t in G by a non-terminal t' with no production rule. This creates a new grammar G', with non-terminals $N \cup T'$ where T' is the set of all t's, no terminals, and rules $\alpha' \rightarrow \beta'$. And where we had $X \in N \cup T$ we now have $X' \in N \cup T'$. In short, Z' is Z in which all terminals from T have been replaced by the corresponding ones from T', for any form Z. Sentences in the intersection now read like this:

$$\langle {X^+}'_0 \Rightarrow \overleftarrow{{X^+}'_1} \rangle \ \langle {X^+}'_1 \Rightarrow \overleftarrow{{X^+}'_2} \rangle \ \langle {X^+}'_2 \Rightarrow \overleftarrow{{X^+}'_3} \rangle \quad ... \quad \langle {X^+}'_k \Rightarrow \overleftarrow{t'^*} \rangle$$

In the next step we replace the final $t'^* \rangle$ in L'_I, which was an add-on anyway, by $\overleftarrow{Y'} \rangle Y$, with $Y = t^*$. This exports sequences of symbols in T' to outside the production sequence, reverses them, and converts them to symbols in T. Again this form is easy to produce since its structure is essentially $\overleftarrow{S'} x S$.

In the last step we extend the sentences in L'_P with t^*, to match the t^* supplied by the sentences from L'_I. If we now erase the markers in a sentence in the intersection, the non-terminals in N and the pseudo-terminals in T', everything disappears except the final sequence of terminals, a production of G. This proves the theorem in the title by construction.

There is a nice symmetry here: L'_P makes sure the production sequence starts with a start symbol, but has no grip on the final result; L'_I makes sure the production sequence ends in the correct sequence of terminals, but has no grip on the start of the production process. In between L'_P ensures that the start of a production step is related by a production rule to the result of that step; and L'_I makes sure that the result of one production step is the start of the next production step. Together they ensure that a sentence in the intersection is a correctly starting, correctly proceeding, and correctly ending production sequence in G'.

It can also be pointed out that only L_P depends on the rules in G; L_I can be constructed from the symbols in G only.

3 The Implementation

We will now turn to the precise form of the context-free grammars for L'_P and L'_I for a given Type 0 grammar

$$G \equiv \{N, T, P, S\}$$

where N is the set of non-terminals
T is the set of terminals
P is the set of production rules $\alpha_i \rightarrow \beta_i$
S is the start symbol, $S \in N$

We define G' as:

$$G' \equiv \{N \cup T', \emptyset, P', S\}$$

where N is the set of non-terminals
T' is the set of pseudo-terminals corresponding to T
P' is the set of production rules $\alpha'_i \rightarrow \beta'_i$
S is the start symbol, $S \in N'$

The production language is

$$\langle S \Rightarrow \overleftarrow{\beta'_i}\rangle \quad \left(\langle {X^*}_l \alpha'_i {X^*}_r \Rightarrow \overleftarrow{{X^*}_r}\,\overleftarrow{\beta'_i}\,\overleftarrow{{X^*}_l}\rangle\right)^* \quad V$$

and the intersection language is

$$\langle X^* \Rightarrow \quad \left(\overleftarrow{X^*}\rangle\langle X^* \Rightarrow\right)^* \quad \overleftarrow{V'}\rangle V$$

where X is any element of $N \cup T'$ and V is any sequence of terminal symbols.

These languages are simple and have simple grammars, and it would be nice if we could modify them so they would have some special properties, to strengthen the theorem in the title.[2]

As they stand they have no special properties; they can even be ambiguous. For example, given the rule $AA \to A$ in G', the production step $\langle AAA \Rightarrow AA\rangle$ in L'_P can be parsed as $\langle A(AA(\Rightarrow)A)A\rangle$ and as $\langle (AA(A \Rightarrow A)A)\rangle$.

Marking the end of the substitution in the production step with a special marker $|$ removes the ambiguity but fails to make L'_P deterministic. For example, given the rules $ABC \to X$ and $BC \to X$ in G', and the LR(1) parsing stack $...\langle ...AABC[F]X|A$ where $[F]$ is the already reduced input around the $\Rightarrow$, an LR(1) parser cannot know whether to reduce $ABC[F]X$ or $BC[F]X$ to $[F]$, since both are compatible with the look-ahead A.

To solve this, we need markers $|_i$ which identify the rule that has been used in the substitution. But if we going to use rule-identifying markers anyway, we might as well put them in front of the α_i of the substituting rule. This makes the production language L'_P

$$\langle S \Rightarrow |_i \overleftarrow{\beta'_i}\rangle \quad \left(\langle {X^*}_l |_i \alpha'_i {X^*}_r \Rightarrow \overleftarrow{{X^*}_r}\,\overleftarrow{\beta'_i}\,\overleftarrow{{X^*}_l}\rangle\right)^* \quad V$$

which is LL(1); a stronger result can hardly be obtained.

If we modify the production language L'_P by inserting markers into it we have to insert them in the intersection language L'_I too. Since L'_P makes sure they appear only in the right place, L'_I can insert them in any place: a sentence with a marker in the wrong place will just not appear in the intersection. This makes the intersection language L'_I

$$\langle X^* \Rightarrow \quad \left(\overleftarrow{X^*}\rangle\langle X^*_| \Rightarrow\right)^* \quad \overleftarrow{V'}\rangle V$$

where $X_|$ means an X possibly preceded by one of the markers $|_i$.

There is no way to make this language LL(1): an LL(1) parser cannot decide when to switch over from reading an $\overleftarrow{X^*}$ to reading the final $\overleftarrow{V'}$. But the language is LR(1), since an LR(1) parser can postpone the decision between X and t' until after the $\rangle$.

We can now formulate a sharper form of the theorem in the title:

$$\mathcal{L}_0 = h_e(\mathcal{L}_{LL(1)} \cap \mathcal{L}_{LR(1)})$$

The grammar templates for $\mathcal{L}_{LL(1)} = L_P$ and $\mathcal{L}_{LR(1)} = L_I$ are given in Figures 1 and 2. Both grammar templates start with $S \to HMR$, for Head, Middle and Result. This reflects the fact that the production and intersection languages both have the same fundamental structure. The precise definitions of H, M, and R then follow from the forms discussed above.

[2] Harrison [2] goes to some length to prove that his L_P and L_I are deterministic.

$$
\begin{array}{ll}
S_P \to HMR & \\
H \to \langle H_1 \rangle & \\
H_1 \to |_i S \Rightarrow \overleftarrow{\beta_i'} & \text{for all rules } S \to \beta_i' \text{ in } G' \\
M \to M_1 M & \\
M \to \varepsilon & \\
M_1 \to \langle M_2 \rangle & \\
M_2 \to X_i M_2 X_i & \text{for all } X_i \text{ in } N \cup T' \text{ in } G' \\
M_2 \to M_3 & \\
M_3 \to |_i \alpha_i' M_4 \overleftarrow{\beta_i'} & \text{for all rules } \alpha_i' \to \beta_i' \text{ in } G' \\
M_4 \to X_i M_4 X_i & \text{for all } X_i \text{ in } N \cup T' \text{ in } G' \\
M_4 \to \Rightarrow & \\
R \to R_1 R & \\
R \to \varepsilon & \\
R_1 \to T_i & \text{for all } T_i \text{ in } G'
\end{array}
$$

Figure 1: Grammar template for the production language

$$
\begin{array}{ll}
S_I \to HMR & \\
H \to \langle |H_1 \Rightarrow & \\
H_1 \to H_1 H_2 & \\
H_1 \to \varepsilon & \\
H_2 \to X_i & \text{for all } X_i \text{ in } N \cup T' \text{ in } G' \\
M \to M M_1 & \\
M \to \varepsilon & \\
M_1 \to M_2 \Rightarrow & \\
M_2 \to X_i M_2 | X_i & \text{for all } X_i \text{ in } N \cup T' \text{ in } G' \\
M_2 \to \rangle \langle & \\
R \to T_i' R T_i & \text{for all } T_i', T_i \text{ pairs in } G' \\
R \to \rangle &
\end{array}
$$

Figure 2: Grammar template for the intersection language

4 A Detailed Example

We will use the standard example of a Type 0 (actually Type 1) grammar, a grammar for $a^n b^n c^n$:

G:
0: $S \to abc$
1: $S \to aSQ$
2: $bQc \to bbcc$
3: $cQ \to Qc$

The corresponding grammar with pseudo-terminals A (=a'), B (=b'), and C (=c') is:

G':
0: $S \to ABC$
1: $S \to ASQ$
2: $BQC \to BBCC$
3: $CQ \to QC$

The relevant sets are:

$N = \{S, Q\}$
$T = \{a, b, c\}$
$T' = \{A, B, C\}$
$N \cup T' = \{S, Q, A, B, C\}$

These and the templates from Figures 1 and 2 lead to the LL(1) grammar (in *LLgen* format) in Figure 3 and the LR(1) grammar (in *bison* format) in Figure 4.

It would be nice to show parts of the languages L'_P and L'_I, and find a sentence in the intersection, but the production sequence for a simple example like *aaabbbccc* is 143 tokens long, and there are at least 10^{84} sentences of that length in both L'_P and L'_I. We will therefore just show the sentence in the intersection:

```
<1S => QSA>
<A1SQ => QQSAA>
<AA0SQQ => QQCBAAA>
<AAAB3CQQ => QCQBAAA>
<AAA2BQCQ => QCCBBAAA>
<AAABBC3CQ => CQCBBAAA>
<AAABB3CQC => CCQBBAAA>
<AAAB2BQCC => CCCBBBAAA>
aaabbbccc
```

This sentence is parsed correctly by the parsers in Figures 3 and 4.

5 Evaluation

No real-world languages are defined through Type 0 or Type 1 grammars, and if they were, the construction used here would produce unusable CF languages for them that would not provide any insight. The value of the theorem lies not in the actual construction of the CF languages but in that it shows that is always possible to find two CF grammars and an erasing homomorphism that will do the job; restricting ourselves to these three items will not exclude any Type 0 language.

There is no reason to assume that the two CF languages produced by the above construction are the only ones or even the simplest. In fact, many non-CF languages can be created as the intersection of two much simpler CF languages than suggested by the theorem. The language $a^n b^n c^n$ is a prime example: it can be obtained as the intersection of the languages $a^k b^k c^i$ and $a^i b^k c^k$, both clearly CF (and LL(1)).

At least one author has applied these ideas to real-world problems. In a series of articles (2001-2011), Alexander Okhotin describes how the context conditions in a programming language can be enforced grammatically by intersecting the traditional CF grammar for the language with one or more "context-enforcing languages".

Also, Jerrold Sadock's "Autolexical Syntax" (1991) can be viewed as being based on a series of intersecting CF and regular grammars, although this interpretation requires more research.

Literature

1. S. Ginsburg, S. Greibach and M.A. Harrison, *One-Way Stack Automata*, J. ACM, **14(2)**, pp. 389-418, April 1967.

2. M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, pp. 594, 1978.

```
Start_P: H M R;       /* grammar specification in LLgen format */

H: '<' H_1 '>';
/* for all rules S -> beta_i' in G': H_1: i S => Rev(beta_i'); */
H_1: H_10 | H_11;
H_10: '0' 'S' '=' '>' 'C' 'B' 'A';
H_11: '1' 'S' '=' '>' 'Q' 'S' 'A';


M: M_1 M | ;

M_1: '<' M_2 '>';

/* for all tokens X in N + T' in G': M_2: X M_2 X; */
M_2: M_2S | M_2Q | M_2A | M_2B | M_2C | M_2_;
M_2S: 'S' M_2 'S';
M_2Q: 'Q' M_2 'Q';
M_2A: 'A' M_2 'A';
M_2B: 'B' M_2 'B';
M_2C: 'C' M_2 'C';
M_2_: M_3;

/* for all rules alpha_i' -> beta_i' in G': M_3: i alpha_i' M_4 Rev(beta_i') */
M_3: M_30 | M_31 | M_32 | M_33;
M_30: '0'         'S' M_4 'C' 'B' 'A';
M_31: '1'         'S' M_4 'Q' 'S' 'A';
M_32: '2' 'B' 'Q' 'C' M_4 'C' 'C' 'B' 'B';
M_33: '3'     'C' 'Q' M_4 'C' 'Q';

/* for all tokens X in N + T' in G': M_4: X M_4 X; */
M_4:  M_4S | M_4Q | M_4A | M_4B | M_4C |  M_4_;
M_4S: 'S' M_4 'S';
M_4Q: 'Q' M_4 'Q';
M_4A: 'A' M_4 'A';
M_4B: 'B' M_4 'B';
M_4C: 'C' M_4 'C';
M_4_: '=' '>';

R: R_1 R | ;

/* for all tokens in T in G': R_1: T; */
R_1: R_1a | R_1b | R_1c;
R_1a: 'a';
R_1b: 'b';
R_1c: 'c';
```

Figure 3: LL(1) grammar for the production language for G'

```
Start_I: H M R;       /* grammar specification in bison format */

H: '<' I H_1 '=' '>';
H_1: H_1 H_2;
H_1: ;

/* for all tokens X in N + T' in G': H_2: X; */
H_2: 'S';
H_2: 'Q';
H_2: 'A';
H_2: 'B';
H_2: 'C';

M: M M_1;
M: ;

M_1: M_2 '=' '>';

/* for all tokens X in N + T' in G': M_2: X M_2 I X; */
M_2: 'S' M_2 I 'S';
M_2: 'Q' M_2 I 'Q';
M_2: 'A' M_2 I 'A';
M_2: 'B' M_2 I 'B';
M_2: 'C' M_2 I 'C';
M_2: '>' '<';

/* for all (T', T) token pairs in G': R: T' R T; */
R: 'A' R 'a';
R: 'B' R 'b';
R: 'C' R 'c';
R: '>';

/* for all rule numbers N in G': I: N; */
I: '0';
I: '1';
I: '2';
I: '3';
I: ;
```

Figure 4: LR(1) grammar for the intersection language for G'