# Transforming Grammers and the Effects on Parse Trees

Ernest Kirstein

December 10, 2014

Implementing a parser often requires a grammar with certain properties. [1, 3] There are also special properties that a grammer can have which will make parsing more efficient. [2, 3]

But modifying a grammar can be expensive from a software engineering perspective. Using a non-intuitive form of the grammer will make implementing compilation more difficult since the parse trees produced using the modified grammer will be different than those one would expect from the natural, unmodified form of the grammar.

An especially expensive engineering problem would be modifying an already-implemented compiler to use a newly discovered grammar property. Implementing that change would also require developers to change both the parser and the code generator which uses the output of the parser.

I propose a radical change - a way to avoid that highly coupled design. Let parsers use a special modified grammer and let the compiler use the more natural form of the grammar. Impossible? No. Impractical? Maybe.

## 1 Definitions

**Definition 1.** *A* symbol *is an abstract building block used in formal grammar definitions. Symbols may be either* terminal *or* non-terminal *(an artificial construction). In practical examples, terminal symbols often correspond to tokens or printable characters, while non-terminal symbols correspond to their.*

**Definition 2.** *A* string *is an ordered list of symbols.*

**Definition 1.** *A* context free production rule describes how a symbol in a string can be replaced with another string.

**Definition 1.** *A* context-free gramma

**Definition 1.** *A* parse tree *is an ordered, rooted tree who's nodes correspond to the following of production rules in a context-free grammar.*

**Theorem 1.** *Let $G$ and $G'$ be weakly equivalent grammars. Then for every parse tree, $t$, generated under $G$ there exists a weekly equivalent parse tree $t'$ under $G'$.*

*Proof.* Let $G$ and $G'$ be weakly equivalent grammars. Therefore, the language produced by $G$, $L(G)$, is equal to the language produced by $G'$. For all strings, $s$, in $L(G)$ there exists a tree $t$ □

## References


[1] F. D. Lewis. Recursive descent parsing. `http://www.cs.engr.uky.edu/~lewis/essays/compilers/rec-des.html`, 2002.

[2] Stefano Crespi Reghizzi. *Formal Languages and Compilation.* Springer-Verlag London Limited, Italy, 2009.

[3] William M. Waite and Lynn R. Carter. *An Introduction to Compiler Construction.* HarperCollins College Publishers, New York, NY, 1993.