WPI-CS-TR-08-14 October 2008

Well-behaved parsing of extensible-syntax languages

by

John N. Shutt

Well-behaved parsing of extensible-syntax languages

John N. Shutt
jshutt@cs.wpi.edu
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

October 2008

Abstract

Some programming languages allow the programmer to extend the language syntax; this language feature is called *extensible syntax.* In this paper, we use formal mathematics to illuminate the general question of what kinds of syntax extensions can be supported without introducing various forms of undesirable parsing behavior into the language. We define a *parser* as a function that maps each source string, finite set of CFG rules, and start symbol to a set of syntax trees (usually, a null set or singleton set); and construct well-behavedness criteria for parsers by considering how the behavior of a parser on one set of rules relates to its behavior on other, slightly different sets of rules.

Among the results obtained are that there is a unique largest well-behaved completeness property supporting a universal parser, and that whether an arbitrary grammar satisfies this property is formally undecidable.

Contents

List of Definitions

List of Theorems

1 Introduction

Even though most programming languages have non-context-free syntax —in the sense that the set of all statically valid, i.e. compilable, program texts cannot be freely generated by a CFG (Context-Free Grammar)[1]— it is usual to present for each language a CFG, describing a context-free superset of the language syntax. CFGs appeal to the human sense of simple syntactic structure, by imposing syntax-tree structure on every valid source string.[2] We define a *parser* to be, abstractly, a function that takes as input a start symbol, a set of CFG rules, and a source string, and produces as output a set of syntax trees consistent with the given inputs. The parser imposes non-context-free syntactic constraints of the language (such as requiring identifiers to be declared, or possibly even enforcing a static type discipline) by selectively omitting from its output some trees allowed by the input set of CFG rules. The parser rejects a source string by returning the empty set; if the parser returns more than one tree, the source string is *ambiguous*. (Ambiguous parsing is not usually practiced by compilers, but is handled by some parsing algorithms, such as Earley's algorithm ([Ea70]).)

Usually, the programming language CFG is fixed by the language designer (who typically optimizes it for efficient parsing rather than human readability, as in the case of LR(k) grammars). The parser is then always given the same rule set and start symbol, with only the source string varying between invocations. However, some programming languages allow the programmer to extend the language syntax by means of instructions within the program, a language feature called *extensible syntax*. In this case, the parser will not always receive the same rule set, and it is relevant to ask how its behavior varies as changes are made to the rule set. The current paper proposes mathematical well-behavedness criteria for parsers over classes of rule sets, and characterizes parsers that meet the criteria. The properties considered are purely extensional: their definitions are independent of parsing algorithms, efficiency, and even computability. Principal results proven are:

- There is no parser that is universally well-behaved and language-complete. (Theorem 7.5.)
- There are parsers that are universally well-behaved and association-complete. (Theorem 7.9.)
- There is a unique largest well-behaved completeness property that supports a universal parser, property Z_{WR}. (Theorem 8.12.)

[1]This is a common delineation of what constitutes syntax. However, when Donald Knuth devised Attribute Grammars in the late 1960s, by his own account ([Kn90]), he differed from most of his peers at the time in that he considered ALGOL to be a context-free language, consisting of a context-free set of strings some of which had the semantics 'uncompilable program'.

[2]The conceptual significance of CFGs is discussed in [Shu03, §§1.6–1.7].

- Whether an arbitrary grammar conforms to property Z_{WR} is formally undecidable. (Theorem 8.13.)

§2 clarifies a few basic mathematical notations and terms used in the following treatment. §§3–5 formally develop *syntax trees* and, building on them, *parsers.* §6 considers properties of parsers with respect to particular grammars. §§7–8 consider behavior of parsers with respect to entire classes of grammars. §9 discusses implications of the results obtained.

Note: Non-Chomsky grammars

There are a variety of formalisms (grammar models) for describing program syntax that provide tree structure but also incorporate non-context-free constraints. The most common in general usage is Attribute Grammars; but there are also a wide variety of *adaptive* grammar formalisms that allow the grammar to manipulate its own rule set during parsing ([Shu03]). Attribute grammars are also used widely outside of programming language analysis; and adaptive grammar formalisms, though usually encountered in connection with extensible-syntax programming languages, are also occasionally used elsewhere (a recent example being [OrCrAl07]), and some adaptive grammar formalisms could in principle be applied as widely as attribute grammars.

These formalisms are not meant to be excluded, nor even discouraged, by our choice in this paper to cast our analysis in terms of CFGs. We use CFGs because we are studying the relation of parsing behavior to tree structure, and CFGs are the abstract essence of logical tree structure. The non-context-free aspects of syntax must exist for our analysis to be relevant, because they determine what behavior we are relating to the tree structure. The formal results of this analysis have no direct bearing on any assessment of the merits or demerits of any particular non-Chomsky formalism for use in describing, or mechanically parsing, programs.

2 Mathematical preliminaries

For arbitrary set A, $\mathcal{P}(A)$ denotes the *power set* of A, the set of all subsets of A (including A itself). $\mathcal{P}(A) = \{B \mid B \subseteq A\}$. $\mathcal{P}_\omega(A)$ denotes the set of all finite subsets of A.[3]

For arbitrary sets A and B, $A - B$ denotes the *set difference* of A minus B, the set of all elements of A that are not elements of B. $A - B = \{a \mid (a \in A) \wedge (a \notin B)\}$.

For arbitrary sets A and B, $A \times B$ denotes the *product* of A and B, the set of all pairs whose first element is from A and whose second element is from B. Pairs are denoted $\langle a, b\rangle$ or (a, b). $A \times B = \{\langle a, b\rangle \mid (a \in A) \wedge (b \in B)\}$.

For arbitrary set A, A^* denotes the set of all strings over A, that is, all strings whose elements belong to A. The number of elements in a string is its *length.* Strings

[3]Notation $\mathcal{P}_\omega(A)$ is borrowed from [We92, p. 47].

are understood to have finite length. A^+ denotes the set of all non-empty strings over A, that is, strings with non-zero length. A non-string object (such as an atom or pair) may be coerced to a string of length one. Strings are usually named by Greek letters, qualified by subscripts or primes as needed. Concatenation of strings is denoted by juxtaposition, as $\omega = \alpha\beta\gamma$. The set of elements of a string ω is denoted $\bigcup\omega$; thus, for $\omega = x_1x_2\ldots x_n$, $\bigcup\omega = \{x_k \mid 1 \leq k \leq n\}$.

An entity that must be a set is named by a capital letter, and entities that must belong to it are named by the corresponding lower-case letter, qualified as needed.

Each function has a fixed arity. If its arity is $n \geq 2$, it may be treated interchangeably as a unary function whose input is an n-tuple (delimited by angle brackets, $\langle\rangle$). For arbitrary functions f and g, $f \subseteq g$ means that for all possible inputs u, $g(u)$ is defined iff $f(u)$ is defined, and if they are defined then their results are sets with $f(u) \subseteq g(u)$. (We do not take the extensional view of functions, whereby a function is a set of input/output pairs.)

3 Syntax trees

Definition 3.1 Set U is the universe. The following three subsets of U are each countably infinite:

- Set S of symbols.
- Set $S_T \subset S$ of terminals.
- Set $S_N = S - S_T$ of nonterminals. ■

The following function $\mathcal{T}$ will be used to construct both (CFG) rules and (syntax) trees.

Definition 3.2 $\mathcal{T}\colon \mathcal{P}(U) \to \mathcal{P}(S_N \times U^*)$

$$\text{where} \quad \mathcal{T}(X) = S_N \times X^* . \ ■$$

Accessor functions **lhs** and **rhs** return the left- and right-hand parts of a pair constructed by $\mathcal{T}$, with the minor complication that, in the subsequent analysis, terminal symbols will be considered syntax trees of depth zero; a terminal symbol is its own left-hand side, and has no right-hand side.

Definition 3.3
$\mathbf{lhs}\colon S_T \cup (S_N \times U^*) \to S$

$$\text{where} \quad \mathbf{lhs}(x) = \begin{cases} n & \text{if } x = \langle n, \omega\rangle \\ x & \text{otherwise}. \end{cases} \tag{1}$$

$\mathbf{rhs}\colon S_N \times U^* \to U^*$

$$\text{where} \quad \mathbf{rhs}(\langle n, \omega\rangle) = \omega . \ ■ \tag{2}$$

The following function **mapr** applies a function f to each of the components of the **rhs** of a construct, and returns the concatenated results.

Definition 3.4

$\mathbf{mapr}\colon (U \to U) \times (S_N \times U^*) \to U^*$

$$\text{where} \quad \mathbf{mapr}(f, \langle n, u_1 u_2 u_3 \cdots u_k \rangle) = f(u_1) f(u_2) f(u_3) \cdots f(u_k)\,.$$

It is now easy to construct rules and grammars. (Here the unadorned word *grammar* will always mean *unpointed grammar*, that is, a set of rules with no designated start symbol.)

Definition 3.5

Set $R = \mathcal{T}(S)$ of rules.
Set $G = \mathcal{P}_\omega(R)$ of grammars. ■

Trees are only slightly more involved.

Definition 3.6

$$\begin{aligned} \text{Sets} \qquad & T_0 = S_T \\ \forall k \geq 1,\ & T_k = T_{k-1} \cup \mathcal{T}(T_{k-1}). \end{aligned} \tag{3}$$

$$\text{Set } T = \bigcup_{k \geq 0} T_k \quad \text{of trees.} \tag{4}$$

$$\text{Set } T_N = T - T_0. \ ■ \tag{5}$$

Note that T_k is the set of all trees of depth $\leq k$.

Functions **fringe**, **rules**, and **subtrees** map a tree into, respectively, the sentence that it derives, the set of rules that it uses in doing so, and the set of its proper subtrees.

Definition 3.7

$\mathbf{fringe}\colon T \to S_T^*$

$$\begin{aligned} \text{where} \quad \forall t \in T_0, \quad & \mathbf{fringe}(t) = t \\ \forall t \in T_N, \quad & \mathbf{fringe}(t) = \mathbf{mapr}(\mathbf{fringe}, t)\,. \end{aligned} \tag{6}$$

$\mathbf{rules}\colon T \to G$

$$\begin{aligned} \text{where} \quad \forall t \in T_0, \quad & \mathbf{rules}(t) = \{\} \\ \forall t \in T_N, \quad & \mathbf{rules}(t) = \{\langle \mathbf{lhs}(t), \mathbf{mapr}(\mathbf{lhs}, t) \rangle\} \\ & \qquad \cup \bigcup \bigcup \mathbf{mapr}(\mathbf{rules}, t)\,. \end{aligned} \tag{7}$$

$\mathbf{subtrees}\colon T \to \mathcal{P}(T_N)$

$$\begin{aligned} \text{where} \quad \forall t \in T_0, \quad & \mathbf{subtrees}(t) = \{\} \\ \forall t \in T_N, \quad & \mathbf{subtrees}(t) = (T_N \cap \bigcup \mathbf{rhs}(t)) \\ & \qquad \cup \bigcup \bigcup \mathbf{mapr}(\mathbf{subtrees}, t)\,. \ ■ \end{aligned} \tag{8}$$

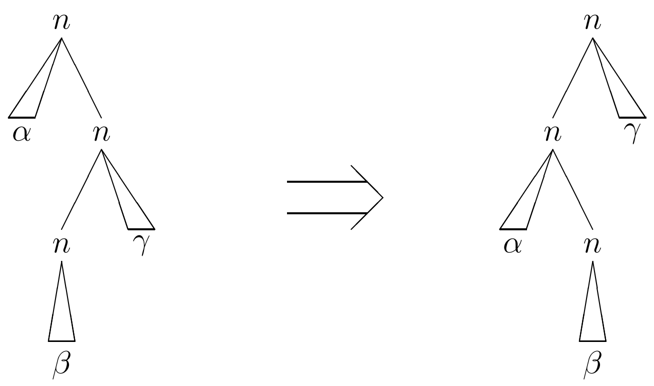

Figure 1: Left-association.

4 Association

The structural transformation of a tree shown in Figure 1 is called *left association*. The inverse transformation is called *right association*. The *associative closure* of a tree t is the set of all other trees that can be formed by a finite number of left or right associative transformations, either to t itself or to any of its subtrees.

Associative closure is a little too complicated to define formally in one stroke, so several incidental functions will be defined first. First, function **la** performs the left associative transformation illustrated in the figure, mapping a tree t into the singleton set containing its left association if possible, or into the empty set if t is not of the correct form.

- $\mathbf{la}: T \to \mathcal{P}(T)$

$$\text{where} \quad \mathbf{la}(t) = \{\tau \in T \mid \exists n \in S_N \text{ and } \alpha, \beta, \gamma \in T^* \text{ such that } t = \langle n, \alpha\langle n, \langle n, \beta\rangle\gamma\rangle\rangle \text{ and } \tau = \langle n, \langle n, \alpha\langle n, \beta\rangle\rangle\gamma\rangle\}\,. \tag{9}$$

Function a_0 maps a tree t into the set of all trees that can be reached from t by left or right association in not more than one step. Note that $a_0(t)$ must be a set of either one, two, or three elements.

- $a_0: T \to \mathcal{P}(T)$

$$\text{where} \quad a_0(t) = \{t\} \cup \mathbf{la}(t) \cup \{\tau \mid t \in \mathbf{la}(\tau)\}\,. \tag{10}$$

Function a_1 is the basis for the inductive construction of associative closure. It maps a tree t into the set of trees that can be reached from t by applying left or right association, at most once, either to t or to one of its subtrees.

- $a_1 \colon T \to \mathcal{P}(T)$

$$\begin{aligned} \text{where} \quad & \forall t \in T_0, \quad a_1(t) = \{t\} \\ & \forall t \in T_N, \quad a_1(t) = a_0(t) \cup \{\langle n, \alpha\tau'\beta\rangle \mid t = \langle n, \alpha\tau\beta\rangle \text{ and } \\ & \qquad\qquad\qquad\qquad \tau' \in a_1(\tau)\} \,. \end{aligned} \tag{11}$$

- $\forall k \geq 2,\ a_k \colon T \to \mathcal{P}(T)$

$$\text{where} \quad a_k(t) = \bigcup_{\tau \in a_{k-1}(t)} a_1(\tau) \,. \tag{12}$$

Finally, **assoc** maps a tree into its associative closure.

Definition 4.1
assoc: $T \to \mathcal{P}(T)$

$$\text{where} \quad \mathbf{assoc}(t) = \bigcup_{k \geq 1} a_k(t) \,. \ \blacksquare$$

Theorem 4.2 If $\tau \in \mathbf{assoc}(t)$ then $\mathbf{assoc}(\tau) = \mathbf{assoc}(t)$. ∎

Proof. Follows immediately from the definition of associative closure. ∎

Theorem 4.3 If $\tau \in \mathbf{assoc}(t)$ then $\mathbf{rules}(\tau) = \mathbf{rules}(t)$. ∎

Proof. Left association preserves the value of **rules**. The desired result follows by induction. ∎

5 Parsers

A *parser* maps a nonterminal n, grammar g, and terminal string w into a set of trees that use rules of g to derive w from n.

Formally, we define first the *maximal parser* $f_\top$, which returns the set of all syntax trees formally consistent with the inputs; then a *parser* in general must return a subset of what $f_\top$ would return. The *minimal parser*, $f_\bot$, always returns the empty set.

Definition 5.1
Maximal parser $f_\top \colon S_N \times G \times S_T^* \to \mathcal{P}(T_N)$

$$\begin{aligned} \text{where} \quad f_\top(n, g, w) = \{t \in T \mid\ & \mathbf{lhs}(t) = n \text{ and} \\ & \mathbf{rules}(t) \subseteq g \text{ and} \\ & \mathbf{fringe}(t) = w\} \,. \end{aligned} \tag{13}$$

Set F of all parsers $F = \{f \mid f \subseteq f_\top\}$. (14)

Minimal (or, empty) parser $f_\bot \in F$

$$\text{where} \quad f_\bot(n, g, w) = \{\}\,. \ ■ \tag{15}$$

Theorem 5.2 If $f \in F$, then $f_\bot \subseteq f \subseteq f_\top$. ■

Proof. Follows immediately from the definitions of $f_\top$, F, and $f_\bot$. ■

6 Properties

Most of our analysis will concern properties of a parser relative to a grammar. To facilitate this treatment, we define the following set Z.

Definition 6.1 Set $Z = F \times G$. ■

The following function **allows** maps a pair $(f, g) \in Z$ into the set of all trees that can be generated by f given g.

Definition 6.2
allows: $Z \to \mathcal{P}(T_N)$

$$\text{where} \quad \textbf{allows}(f, g) = \bigcup_{n \in S_N,\ w \in S_T^*} f(n, g, w)\,. \ ■$$

It is now possible to define properties of parsers relative to grammars by constructing subsets of Z.

Definition 6.3 A subset of Z is a *property* (of parsers relative to grammars). ■

Pairs considered as members of a property are usually denoted using parentheses rather than angle brackets, to distinguish them visually from trees and rules; thus, $(f, g) \in Z$, $\langle n, \omega \rangle \in T_N$.

6.1 Well-behavedness

A parser f is *unambiguous* on a grammar g iff f given g generates at most one tree for any n and w.

Definition 6.4
Set $Z_U = \{(f, g) \in Z \mid \forall n \in S_N \text{ and } w \in S_T^*,\ |f(n, g, w)| \leq 1\}$. ■

f is *cumulative* on g iff whether a tree t is allowed depends only on rules used by t.

Definition 6.5
Set $Z_C = \{(f,g) \in Z \mid \forall t \in T$ and $h \in G$ such that $\mathbf{rules}(t) \subseteq h \subseteq g,$
$t \in \mathbf{allows}(f,g) \Leftrightarrow t \in \mathbf{allows}(f,h)\}$. ■

f is *inductive* on g iff, whenever a tree t is allowed, all of its subtrees are also allowed.

Definition 6.6
Set $Z_I = \{z \in Z \mid \forall t \in \mathbf{allows}(z),\ \mathbf{subtrees}(t) \subset \mathbf{allows}(z)\}$. ■

Theorem 6.7 Suppose $(f,g) \in Z_C$, and $h \subseteq g$. Then the following all hold.
$(f,h) \in Z_C$.
If $(f,g) \in Z_U$ then $(f,h) \in Z_U$.
If $(f,g) \in Z_I$ then $(f,h) \in Z_I$. ■

Proof. Suppose f is cumulative on g, and $h \subseteq g$. Suppose $t \in T$ and $h' \in G$ such that $\mathbf{rules}(t) \subseteq h' \subseteq h$. Since $\mathbf{rules}(t) \subseteq h \subseteq g$ and f is cumulative on g, $t \in \mathbf{allows}(f,g)$ iff $t \in \mathbf{allows}(f,h)$. Since $\mathbf{rules}(t) \subseteq h' \subseteq g$ and f is cumulative on g, $t \in \mathbf{allows}(f,g)$ iff $t \in \mathbf{allows}(f,h')$. Therefore, $t \in \mathbf{allows}(f,h)$ iff $t \in \mathbf{allows}(f,h')$. Therefore, f is cumulative on h.

Suppose f is cumulative and unambiguous on g, and $h \subseteq g$. Suppose $n \in S_N$ and $w \in S_T^*$. Because f is cumulative on g, $f(n,h,w) \subseteq f(n,g,w)$. But f unambiguous on g implies $|f(n,g,w)| \leq 1$, therefore $|f(n,h,w)| \leq 1$. So f is unambiguous on h.

Suppose f is cumulative and inductive on g, $h \subseteq g$, $t \in \mathbf{allows}(f,h)$, and $\tau \in \mathbf{subtrees}(t)$. Since f is cumulative on g, $t \in \mathbf{allows}(f,g)$. Since f is inductive on g, $\tau \in \mathbf{allows}(f,g)$; and since $\mathbf{rules}(t) \subseteq h$, $\mathbf{rules}(\tau) \subseteq h$; therefore, since f is cumulative on g, $\tau \in \mathbf{allows}(f,h)$. So f is inductive on h. ■

f is *well-behaved* on g iff f is unambiguous, cumulative, and inductive on g.

Definition 6.8
Set $Z_W = Z_U \cap Z_C \cap Z_I$. ■

Theorem 6.9 If $(f,g) \in Z_W$ and $h \subseteq g$, then $(f,h) \in Z_W$. ■

Proof. Follows immediately from Theorem 6.7. ■

6.2 Completeness

The next several properties are various forms of completeness, meaning they guarantee that certain kinds of trees are allowed. The weakest such property is *language-completeness*: f is language-complete on g if, for every $n \in S_N$ and $w \in S_T^*$, $f(n,g,w)$ is nonempty iff $f_\top(n,g,w)$ is nonempty.

Definition 6.10
Set $Z_L = \{(f,g) \in Z \mid \forall t \in \textbf{allows}(f_\top, g)\ \exists \tau \in \textbf{allows}(f,g)$ such that
$\textbf{lhs}(\tau) = \textbf{lhs}(t)$ and $\textbf{fringe}(\tau) = \textbf{fringe}(t)\}$. ■

f is *association-complete* on g if, for every tree t allowed by $f_\top$, there is a tree in $\textbf{assoc}(t)$ allowed by f.

Definition 6.11
Set $Z_A = \{(f,g) \in Z \mid \forall t \in \textbf{allows}(f_\top, g)\ \exists \tau \in \textbf{allows}(f,g)$ such that
$\tau \in \textbf{assoc}(t)\}$. ■

f is *tree-complete* on g if every tree (using g) allowed by $f_\top$ is allowed by f.

Definition 6.12
Set $Z_T = \{(f,g) \in Z \mid \textbf{allows}(f,g) = \textbf{allows}(f_\top, g)\}$. ■

Note that $f_\top$ is tree-complete on every $g \in G$.

Theorem 6.13 $Z_T \subset Z_A \subset Z_L$. ■

Proof. From the definitions, it is immediate that $z \in Z_A$ implies $z \in Z_L$, and $z \in Z_T$ implies $z \in Z_A$. So $Z_T \subseteq Z_A \subseteq Z_L$.

Since there are trees t such that $\textbf{assoc}(t)$ has more than one element, there exist $z \in Z_A - Z_T$, hence $Z_T \subset Z_A$.

There exist trees t, τ that derive the same sentence from the same nonterminal, but $\tau \notin \textbf{assoc}(t)$; let $g = \textbf{rules}(t) \cup \textbf{rules}(\tau)$. In order for a parser to be association-complete on g, it must allow representatives of the disjoint equivalence classes $\textbf{assoc}(t)$ and $\textbf{assoc}(\tau)$; but it can still be language-complete without allowing both. So there exist $z \in Z_L - Z_A$, hence $Z_A \subset Z_L$. ■

It is particularly of interest to consider various forms of completeness in combination with well-behavedness.

Definition 6.14
Set $Z_{WL} = Z_W \cap Z_L$.
Set $Z_{WA} = Z_W \cap Z_A$.
Set $Z_{WT} = Z_W \cap Z_T$. ■

Theorem 6.15 Suppose $(f,g) \in Z_W$, and $h \subseteq g$. Then the following both hold.
If $(f,g) \in Z_T$ then $(f,h) \in Z_T$.
If $(f,g) \in Z_A$ then $(f,h) \in Z_A$. ■

Proof. Suppose f is well-behaved and tree-complete on g, and $h \subseteq g$. Suppose $t \in \mathbf{allows}(f_\top, h)$. Then $t \in \mathbf{allows}(f_\top, g)$, and since f is tree-complete on g, $t \in \mathbf{allows}(f, g)$. But f is cumulative on g, ergo $t \in \mathbf{allows}(f, h)$. So f is tree-complete on h.

Suppose f is well-behaved and association-complete on g, and $h \subseteq g$. Suppose $t \in \mathbf{allows}(f_\top, h)$. Then $t \in \mathbf{allows}(f_\top, g)$, and since f is association-complete on g, $\exists \tau \in \mathbf{allows}(f, g)$ such that $\tau \in \mathbf{assoc}(t)$. By Theorem 4.3, $\mathbf{rules}(t) = \mathbf{rules}(\tau)$; hence, $\mathbf{rules}(\tau) \subseteq h \subseteq g$. Since f is cumulative on g, $\tau \in \mathbf{allows}(f, g)$ implies $\tau \in \mathbf{allows}(f, h)$. So f is association-complete on h. ■

However, the above theorem does not generalize to language-completeness.

Theorem 6.16 There exist $(f, g) \in Z_{WL}$ and $h \subseteq g$ such that $(f, h) \in Z_W - Z_L$. ■

Proof. Let $n_1, n_2 \in S_N$ be nonterminals; $w \in S_T^*$ be a terminal string; $r_1, r_2, r_3 \in R$ be rules $r_1 = \langle n_1, w \rangle$, $r_2 = \langle n_2, w \rangle$; $r_3 = \langle n_1, n_2 \rangle$; and $t_1, t_2, t_3 \in T_N$ be trees $t_1 = r_1$, $t_2 = r_2$, $t_3 = \langle n_1, t_2 \rangle$. Let $g = \{r_1, r_2, r_3\}$.

Let $f \in F$ such that $\mathbf{allows}(f, g) = \{t_2, t_3\}$, and f is cumulative on g. Since f is unambiguous and inductive on g, f is well-behaved on g. Since $\mathbf{allows}(f_\top, g) = \{t_1, t_2, t_3\}$, and t_1 has the same **rhs** and **fringe** as t_3, f is language-complete on g. However, $\mathbf{allows}(f, \{r_1, r_2\}) = \{t_2\}$, while $\mathbf{allows}(f_\top, \{r_1, r_2\}) = \{t_1, t_2\}$; so f is not language-complete on $\{r_1, r_2\}$. ■

7 Classes of grammars and parsers

So far, we have dealt with properties, i.e., classes of (f, g) pairs. Now we will use these properties to construct classes of parsers, and classes of grammars.

Definition 7.1
$\mathcal{G}\colon \mathcal{P}(Z) \to \mathcal{P}(G)$

$$\text{where} \quad \mathcal{G}(X) = \{g \in G \mid \exists f \in F \text{ such that } (f, g) \in X\}\,. \tag{16}$$

$\mathcal{F}\colon \mathcal{P}(Z) \to \mathcal{P}(F)$

$$\text{where} \quad \mathcal{F}(X) = \{f \in F \mid \forall g \in \mathcal{G}(X),\ (f, g) \in X\}\,. \ ■ \tag{17}$$

Note that $\mathcal{G}$ uses a weak criterion, while $\mathcal{F}$ uses a strong one. $g \in \mathcal{G}(X)$ means only that X pairs grammar g with some parser; but $f \in \mathcal{F}(X)$ means that X pairs parser f with *every* grammar in $\mathcal{G}(X)$. f is then said to be *universal for* X.

Theorem 7.2 $\mathcal{G}(Z_W) = G$. ■

That is, every grammar has a well-behaved parser.

Proof. By definition, $\mathcal{G}(Z_W) \subseteq G$. It remains to show that $G \subseteq \mathcal{G}(Z_W)$.

Suppose $g \in G$. The empty parser, $f_\perp$, is unambiguous on every grammar; cumulative on every grammar; and inductive on every grammar. Therefore, $(f_\perp, g) \in Z_W$. So $\mathcal{G}(Z_W) = G$. ■

Theorem 7.3 $\mathcal{G}(Z_{WL}) = G$. ■

That is, every grammar has a well-behaved language-complete parser.

Proof. By definition, $\mathcal{G}(Z_{WL}) \subseteq G$. It remains to show that $G \subseteq \mathcal{G}(Z_{WL})$.

Suppose $g \in G$. We will construct a parser f that is well-behaved and language-complete on g.

Impose an ordering on the set T_N of nonterminal trees; this is possible because T_N is countably infinite. Let $X \subset T_N$ be the set of trees enumerated as follows.

```
Enumerate all trees in $\mathbf{allows}(f_\top, g) \cap T_1$.
For $k := 2$ to $+\infty$ do
    For $t \in \mathbf{allows}(f_\top, g) \cap (T_k - T_{k-1})$ do
        If all subtrees of $t$ have been enumerated, then
            If no previously enumerated tree derives $\mathbf{fringe}(t)$ from $\mathbf{lhs}(t)$, then
                Enumerate $t$.
```
(18)

Let f be the following parser.

$$f(u) \quad = \quad X \cap f_\top(u) \tag{19}$$

By construction, f is cumulative on g. By construction of X, f is unambiguous and inductive on g. Hence, f is well-behaved on g. (In fact it is well-behaved on G, i.e. on all grammars.)

Suppose $n \in S_N$, $w \in S_T^*$, and $t \in T$ such that $t \in f_\top(n, g, w)$. It will be shown that $\exists \tau \in f(n, g, w)$. Proceed by induction on the depth of t.

Base case. Suppose $t \in T_1$. Then since $t \in \mathbf{allows}(f_\top, g)$, $t \in X$.

Inductive step. Suppose $k \geq 2$, $t \in T_k - T_{k-1}$, and the proposition holds for all trees in T_{k-1}. Let $t = \langle n, w_0 t_1 w_1 t_2 \cdots t_j w_j \rangle$, where the w_i are terminal strings and the t_i are nonterminal trees. Each t_i has depth at most $k-1$, so by inductive hypothesis, there exists a tree $\tau_i \in X$ that derives the same sentence from the same nonterminal as t_i does. Let $t' = \langle n, w_0 \tau_1 w_1 \tau_2 \cdots \tau_j w_j \rangle$. Then $t' \in f_\top(n, g, w)$. Since $t' \in \mathbf{allows}(f_\top, g)$ and $\mathbf{subtrees}(t') \subseteq X$, the only reason why t' might not be enumerated in X is that some other tree of w from n is enumerated before t' is considered. Either way, some tree of w from n is enumerated. Hence f is language-complete on g.

(Although the theorem does not require f to be computable, in fact it is: it is decidable whether $f_\top(n, g, w)$ is nonempty (membership in a CFL); and if it is nonempty, the enumeration can be used to find $t \in f(n, g, w)$ in finite time.) ■

Note that the above theorem holds for language-completeness, but not for association-completeness or tree-completeness. Contrast this with Theorem 6.15, which holds for association-completeness and tree-completeness, but not for language-completeness.

Theorem 7.4 $\mathcal{F}(Z_W) \neq \{\}$. ∎

That is, there exists a universally well-behaved parser.

Proof. As noted earlier (in the proof of Theorem 7.2), the empty parser $f_\perp$ is well-behaved on all grammars. Thus, $f_\perp \in \mathcal{F}(Z_W)$. ∎

Theorem 7.5 $\mathcal{F}(Z_{WL}) = \{\}$. ∎

That is, no parser is universally well-behaved language-complete: for every parser f, there exists a grammar g such that *some* parser is well-behaved language-complete on g, but f isn't.

Proof. Suppose f is cumulative and language-complete on all grammars. Let $n, n_1, n_2 \in S_N$ be nonterminals, and $w \in S_T^*$ a terminal string. Consider the following grammars.

$$\begin{aligned} g_1 &= \{\langle n, n_1\rangle, \langle n_1, w\rangle\} \\ g_2 &= \{\langle n, n_2\rangle, \langle n_2, w\rangle\} \\ g &= g_1 \cup g_2 \end{aligned} \tag{20}$$

Since f is language-complete on all grammars, it is language-complete on g_1 and g_2. Therefore, $\langle n, \langle n_1, w\rangle\rangle \in \mathbf{allows}(f, g_1)$ and $\langle n, \langle n_2, w\rangle\rangle \in \mathbf{allows}(f, g_2)$. Since f is cumulative on all grammars, it is cumulative on g. Therefore, $\langle n, \langle n_1, w\rangle\rangle \in \mathbf{allows}(f, g)$ and $\langle n, \langle n_2, w\rangle\rangle \in \mathbf{allows}(f, g)$. But then f is ambiguous on g, so $f \notin \mathcal{F}(Z_{WL})$. ∎

It has been shown that the condition of well-behaved language-completeness is a kind of degenerate case: it admits all grammars, and no parsers. The next two theorems show that well-behaved tree-completeness is degenerate in somewhat the opposite sense: it admits exactly all unambiguous grammars, and exactly the parsers that mimic $f_\top$ on all unambiguous grammars.

Theorem 7.6 $\mathcal{G}(Z_{WT}) = \{g \in G \mid (f_\top, g) \in Z_U\}$. ∎

Proof. $f_\top$ is cumulative, inductive, and tree-complete on all grammars, by the definitions of those properties. So for all $g \in G$, $(f_\top, g) \in Z_{WT}$ iff $(f_\top, g) \in Z_U$. It remains to show that $g \in \mathcal{G}(Z_{WT})$ implies $(f_\top, g) \in Z_{WT}$; that is to say, $f_\top \in \mathcal{F}(Z_{WT})$.

Suppose $g \in \mathcal{G}(Z_{WT})$. Then there exists f such that $(f, g) \in Z_{WT}$. Since f is tree-complete on g, $\mathbf{allows}(f, g) = \mathbf{allows}(f_\top, g)$; but then, since f is unambiguous on g, $f_\top$ is unambiguous on g. So $(f_\top, g) \in Z_{WT}$ ∎

Theorem 7.7 $\mathcal{F}(Z_{WT}) = \{f \in F \mid \forall g \in \mathcal{G}(Z_{WT}),$
$\mathbf{allows}(f, g) = \mathbf{allows}(f_\top, g)\}$. ∎

Proof. The theorem says that $f \in F$ is well-behaved and tree-complete on $\mathcal{G}(Z_{WT})$ iff it is tree-complete on that set. Implication left-to-right is trivial; right-to-left remains to be shown. Suppose f is tree-complete on $\mathcal{G}(Z_{WT})$, and $g \in \mathcal{G}(Z_{WT})$.

Then $\mathbf{allows}(f, g) = \mathbf{allows}(f_\top, g)$. Therefore, since $f_\top$ is unambiguous on g, so is f; and since $f_\top$ is inductive on g, so is f.

Since $f_\top$ is well-behaved on g, by Theorem 6.9 it is unambiguous on all subsets of g, so by Theorem 7.6, all these subsets are in $\mathcal{G}(Z_{WT})$. Hence, f is tree-complete on all subsets of g. Therefore, f is cumulative on g. ∎

While Z_{WL} and Z_{WT} are somewhat degenerate, Z_{WA} is not, as the following theorems show. It admits exactly those grammars that are unambiguous except for association, and admits parsers that may behave differently on such grammars.

Theorem 7.8 $\mathcal{G}(Z_{WA}) = \{g \in G \mid \forall t \in \mathbf{allows}(f_\top, g),$
$f_\top(\mathbf{lhs}(t), g, \mathbf{fringe}(t)) = \mathbf{assoc}(t)\}$. ∎

Proof. Suppose $g \in \mathcal{G}(Z_{WA})$ and $t \in \mathbf{allows}(f_\top, g)$. Let $n = \mathbf{lhs}(t)$ and $w = \mathbf{fringe}(t)$. By Theorem 4.3, $\mathbf{assoc}(t) \subseteq f_\top(n, g, w)$. Since $g \in \mathcal{G}(Z_{WA})$, there exists f such that $(f, g) \in Z_{WA}$. Since f is association-complete on g, there exists $\tau \in \mathbf{assoc}(t) \cap f(n, g, w)$. Since f is unambiguous on g, $f(n, g, w) = \{\tau\}$. Since, again, f is association-complete on g, $f_\top(n, g, w) \subseteq \mathbf{assoc}(\tau)$. By Theorem 4.2, $\mathbf{assoc}(\tau) = \mathbf{assoc}(t)$; so $f_\top(n, g, w) = \mathbf{assoc}(t)$.

On the other hand, suppose $g \in G$, and for all $t \in \mathbf{allows}(f_\top, g)$, $f_\top(\mathbf{lhs}(t), g, \mathbf{fringe}(t)) = \mathbf{assoc}(t)$. Let f_{right} be the unique parser that behaves just as $f_\top$ except that from each **assoc** equivalence class it allows only the tree that is grouped strictly to the right. f_{right} is cumulative (Theorem 4.3), inductive, and association-complete on all grammars. Suppose $t \in \mathbf{allows}(f_\top, g)$; then $f_\top(\mathbf{lhs}(t), g, \mathbf{fringe}(t)) = \mathbf{assoc}(t)$, and $|f_{right}(\mathbf{lhs}(t), g, \mathbf{fringe}(t))| = 1$. So f_{right} is unambiguous on g. ∎

Theorem 7.9 $\mathcal{F}(Z_{WA}) \neq \{\}$. Moreover, there exist $f_1, f_2 \in \mathcal{F}(Z_{WA})$ and $g \in \mathcal{G}(Z_{WA})$ such that $\mathbf{allows}(f_1, g) \neq \mathbf{allows}(f_2, g)$. ∎

Proof. It was shown incidentally, in the second half of the proof of Theorem 7.8, that the right-associating parser $f_{right} \in \mathcal{F}(Z_{WA})$. By a similar argument, the left-associating parser $f_{left} \in \mathcal{F}(Z_{WA})$. Let $g \in G$ be any grammar that has associative ambiguity, but is otherwise unambiguous; then $g \in \mathcal{G}(Z_{WA})$, but $\mathbf{allows}(f_{right}, g) \neq \mathbf{allows}(f_{left}, g)$. ■

The parsers f_{left} and f_{right} resolve associative ambiguity in a way that gives all operators (= grammar rules) equal precedence, and groups all classes of operators (= nonterminals) in the same direction. Other parsers in $\mathcal{F}(Z_{WA})$ can be constructed in which not all operators have the same precedence, and/or not all classes of operators group in the same direction. In this way, an infinity of parsers can be constructed in $\mathcal{F}(Z_{WA})$ such that no two of them behave exactly the same way on all grammars in $\mathcal{G}(Z_{WA})$.

8 Well-behaved completeness

On examining Theorems 7.5, 7.7, and 7.9, the question arises,

What is the weakest completeness property Z_{weak} *such that* $\mathcal{F}(Z_W \cap Z_{weak}) \neq \{\}$ *?*

That is, what is the weakest completeness property for which there is a universally well-behaved-complete parser?

To address this question, we must first provide a more formal definition of what constitutes a "completeness" property.

Definition 8.1 Let $\equiv_L$ be the following equivalence on T_N.

$$\begin{aligned} \equiv_L \;=\; \{(t,\tau) \in T_N \times T_N \mid\; & \mathbf{lhs}(t) = \mathbf{lhs}(\tau) \text{ and} \\ & \mathbf{fringe}(t) = \mathbf{fringe}(\tau)\} \end{aligned} \tag{21}$$

A set $Z_x \subseteq Z$ is a *completeness property* iff there exists an equivalence $\equiv_x \subseteq \equiv_L$ on T_N such that

$$\begin{aligned} Z_x \;=\; \{(f,g) \in Z \mid\; & \forall t \in \mathbf{allows}(f_\top, g)\ \exists \tau \in \mathbf{allows}(f,g) \\ & \text{such that } t \equiv_x \tau\} \end{aligned} \tag{22}$$

The equivalence $\equiv_x$ is said to *generate* Z_x. ■

The completeness properties defined in §6.2 all satisfy this definition. Moreover, the following generalization of Theorem 6.13 verifies that language-completeness is the weakest completeness property, and tree-completeness the strongest.

Theorem 8.2 If Z_x is a completeness property, then $Z_T \subseteq Z_x \subseteq Z_L$. ■

Proof. Let $\equiv_x$ be an equivalence that generates Z_x. Z_L is generated by $\equiv_L$, as defined above. Z_T is generated by the equality relation on T_N; call this $\equiv_T$. The definition of completeness property guarantees that $\equiv_x \subseteq \equiv_L$, and the reflexive law guarantees that $\equiv_T \subseteq \equiv_x$. Therefore, $Z_T \subseteq Z_x \subseteq Z_L$. ■

Definition 8.3 A set $Z_{Wx} \subseteq Z$ is a *well-behaved completeness property* iff $Z_{Wx} = Z_W \cap Z_x$ for some completeness property Z_x. Any equivalence $\equiv_x$ that generates Z_x is also said to generate Z_{Wx}. ■

Hereafter we will always prefer Z_{Wx} as a shorthand for $Z_W \cap Z_x$.

The question can now be stated more precisely.

For which completeness properties does $\mathcal{F}(Z_{Wx}) = \{\}$, and for which does $\mathcal{F}(Z_{Wx}) \neq \{\}$? In particular, is there a completeness property Z_x such that for all completeness properties Z_y, $\mathcal{F}(Z_{Wy}) \neq \{\}$ iff $Z_y \subseteq Z_x$?

Recall that the weakest completeness property, Z_L, has no universal parser, while stronger properties Z_A and Z_T do have universal parsers. As equivalence $\equiv_x$ grows stronger (with fewer equivalent pairs $\langle t, \tau \rangle$), completeness property Z_x requires the parser to admit more trees. For a parser to admit these larger numbers of trees, and yet still be unambiguous on grammar g, g must not support more than one such tree; so there are a decreasing number of grammars in $\mathcal{G}(Z_{Wx})$. This makes the task of a universal parser easier, since it only has to achieve well-behaved completeness across a smaller set of grammars.

Theorem 8.4 If Z_x is a completeness property, then $f_\top \in \mathcal{F}(Z_x)$. ■

Proof. Follows immediately from the definition of completeness property. ■

Theorem 8.5 If Z_x is a completeness property generated by $\equiv_x$, $g \in \mathcal{G}(Z_{Wx})$, and $t, \tau \in$ **allows**$(f_\top, g)$, then $t \equiv_L \tau$ iff $t \equiv_x \tau$. ■

Proof. By the definition of completeness property, $\equiv_x \subseteq \equiv_L$. On the other hand, suppose $t \equiv_L \tau$. Since $g \in \mathcal{G}(Z_{Wx})$, there exists $(f, g) \in Z_{Wx}$. Since f is x-complete on g, **allows**(f, g) must have some $\tau' \equiv_x \tau$ and some $t' \equiv_x t$. Since f is unambiguous on g, $\tau' = t'$. Therefore, $\tau \equiv_x t$. ■

The following result is based on a generalization of the proof of Theorem 7.3.

Theorem 8.6 If Z_x is a completeness property, $g \in \mathcal{G}(Z_{Wx})$, and $h \subseteq g$, then $h \in \mathcal{G}(Z_{Wx})$. ■

Proof. Suppose Z_x is a completeness property, $g \in \mathcal{G}(Z_{Wx})$, and $h \subseteq g$. Let $\equiv_x$ be an equivalence that generates Z_x. We will construct a parser f such that $(f, h) \in Z_{Wx}$.

Impose an ordering on the set T_N of nonterminal trees; this is possible because T_N is countably infinite. Let $X \subset T_N$ be the set of trees enumerated as follows.

```
Enumerate all trees in $\mathbf{allows}(f_\top, h) \cap T_1$.
For $k := 2$ to $+\infty$ do
    For $t \in \mathbf{allows}(f_\top, h) \cap (T_k - T_{k-1})$ do
        If all subtrees of $t$ have been enumerated, then
            If no previously enumerated tree is $\equiv_L$ to $t$, then
                Enumerate $t$.
```
(23)

Let f be the following parser.

$$f(u) \quad = \quad X \cap f_\top(u) \tag{24}$$

By construction, f is cumulative on h. By construction of X, f is unambiguous and inductive on h. Hence, f is well-behaved on h.

Suppose $n \in S_N$, $w \in S_T^*$, and $t \in T$ such that $t \in f_\top(n, h, w)$. It will be shown that $\exists \tau \in f(n, h, w)$ with $\tau \equiv_x t$. Proceed by induction on the depth of t.

Base case. Suppose $t \in T_1$. Then since $t \in \mathbf{allows}(f_\top, h)$, $t \in X$.

Inductive step. Suppose $k \geq 2$, $t \in T_k - T_{k-1}$, and the proposition holds for all trees in T_{k-1}. Let $t = \langle n, w_0 t_1 w_1 t_2 \cdots t_j w_j \rangle$, where the w_i are terminal strings, and the t_i are trees. Each t_i has depth at most $k - 1$, so by inductive hypothesis, there exists a tree $\tau_i \in X$ such that $\tau_i \equiv_x t_i$. Let $t' = \langle n, w_0 \tau_1 w_1 \tau_2 \cdots \tau_j w_j \rangle$. Then $t' \in f_\top(n, h, w)$, and $t' \equiv_L t$. Since $t' \in \mathbf{allows}(f_\top, h)$ and $\mathbf{subtrees}(t') \subseteq X$, the only reason why t' might not be enumerated in X is that some other tree $\equiv_L$ to t' is enumerated before t' is considered. Either way, some tree $\tau \equiv_L t$ is enumerated. By Theorem 8.5, $\tau \equiv_x t$. So $(f, h) \in Z_x$. ■

To simplify the statement of subsequent results, we introduce the notion of the *rule base* of a tree t, which is the set of all minimal subsets of $\mathbf{rules}(t)$ needed to construct trees $\equiv_L$ to t. That is, out of all the subsets of $\mathbf{rules}(t)$, take just the ones that can be used to construct trees $\equiv_L$ to t; and then out of those subsets, take just the ones that have no proper subset with that property.

Definition 8.7
$\mathbf{rbase}\colon T_N \to \mathcal{P}(G)$

$$\text{where} \quad \mathbf{rbase}(t) = \mathit{min}\{\mathbf{rules}(\tau) \mid \tau \equiv_L t \text{ and } \mathbf{rules}(\tau) \subseteq \mathbf{rules}(t)\}\,. \ ■$$

For example, consider nonterminals $n, n' \in S_N$; nonempty terminal string $w \in S_T^+$; rules $r_1 = \langle n, nn \rangle$ $r_2 = \langle n, n' \rangle$, $r_3 = \langle n, w \rangle$, $r_4 = \langle n', w \rangle$; and tree $t = \langle n, \langle n, w \rangle \langle n, \langle n', w \rangle \rangle \rangle$. $\mathbf{rules}(t) = \{r_1, r_2, r_3, r_4\}$; but tree $t_1 = \langle n, \langle n, w \rangle, \langle n, w \rangle \rangle$ is $\equiv_L t$ with $\mathbf{rules}(t_1) = \{r_1, r_3\}$, and tree $t_2 = \langle n, \langle n, \langle n', w \rangle \rangle, \langle n, \langle n', w \rangle \rangle \rangle$ is $\equiv_L t$ with $\mathbf{rules}(t_2) = \{r_1, r_2, r_4\}$. Further, these are the only proper subsets of $\mathbf{rules}(t)$ that can be used to construct a tree $\equiv_L t$. So $\mathbf{rbase}(t_1) = \{\{r_1, r_3\}\}$; $\mathbf{rbase}(t_2) = \{\{r_1, r_2, r_4\}\}$; and $\mathbf{rbase}(t) = \{\{r_1, r_3\}, \{r_1, r_2, r_4\}\}$.

So far, the only proven result of the form $\mathcal{F}(Z_{Wx}) = \{\}$ is Theorem 7.5. The technique used to prove that theorem suggests the following result.

Theorem 8.8 Suppose Z_x is a completeness property. Then $\mathcal{F}(Z_{Wx}) \neq \{\}$ iff $\forall g \in \mathcal{G}(Z_{Wx})$ and $t \in \textbf{allows}(f_\top, g)$, both of the following hold.
$|\textbf{rbase}(t)| = 1$; and
$\forall \tau \equiv_L t$, if $\tau \in \textbf{allows}(f_\top, g)$ then $\textbf{rbase}(\tau) = \textbf{rbase}(t)$. ■

In words: there exists a universally well-behaved x-complete parser iff, for every grammar g that has a well-behaved x-complete parser, and every tree t over g, (1) there is only one set of rules in the rule base of t, and (2) all trees over g that are $\equiv_L$ to t have that same rule base. To see where these criteria come from: For every set h in the rule base of any allowed $\tau \equiv_L t$, let $\tau' \equiv_L t$ be some tree that uses exactly h. (Some such τ' must exist, by the definition of rule base.) A Z_{Wx}-universal parser must be x-complete on h, so it must allow on h some $\tau'' \equiv_x \tau'$. Since this parser is cumulative on g, it must retain on g all such trees τ''; and since the number of trees τ'' equals the number of sets h in the rule bases of the $\equiv_L$-class on g of t, unambiguity requires that there is only one such set, i.e., all trees in the $\equiv_L$-class on g have the same singleton rule base.

Proof. Suppose Z_x is a completeness property generated by $\equiv_x$.

Suppose $f \in \mathcal{F}(Z_{Wx})$, $g \in \mathcal{G}(Z_{Wx})$, and $t \in \textbf{allows}(f_\top, g)$. By the definition of completeness property, $\exists t' \in \textbf{allows}(f, g)$ such that $t' \equiv_x t$. It will be shown that $\forall \tau \equiv_L t$ with $\textbf{rules}(\tau) \subseteq g$, $\textbf{rules}(t') \subseteq \textbf{rules}(\tau)$. Suppose $\tau \equiv_L t$, $\textbf{rules}(\tau) \subseteq g$, and $r \in \textbf{rules}(t') - \textbf{rules}(\tau)$. Let $g' = g - \{r\}$. By Theorem 8.6, $g' \in \mathcal{G}(Z_{Wx})$; and $f \in \mathcal{F}(Z_{Wx})$, therefore $(f, g') \in Z_{Wx}$. But then, since $\tau \in \textbf{allows}(f_\top, g')$, there must be some $\tau' \in \textbf{allows}(f, g')$ such that $\tau' \equiv_x \tau$. And since f is cumulative on g, $\tau' \in \textbf{allows}(f, g)$. But if t' and τ' are both in $\textbf{allows}(f, g)$, then f is ambiguous on g, which is a contradiction. Therefore there cannot exist any $r \in \textbf{rules}(t') - \textbf{rules}(\tau)$. This proves the first half of the theorem (left-to-right implication).

On the other hand, suppose that $\forall g \in \mathcal{G}(Z_{Wx})$ and $t \in \textbf{allows}(f_\top, g)$, $|\textbf{rbase}(t)| = 1$ and $\forall \tau \equiv_L t$ if $\tau \in \textbf{allows}(f_\top, g)$ then $\textbf{rbase}(\tau) = \textbf{rbase}(t)$. We will construct a parser $f \in \mathcal{F}(Z_{Wx})$.

Impose an ordering on the set T_N of all trees; this is possible because T_N is countably infinite. Let $s: T_N \to T_N$ be the following partial function.

$$\begin{array}{rcl} s(t) & = & \text{the earliest } t' \text{ in the ordering such that} \\ & & \quad t' \equiv_L t \\ & & \quad \forall \tau \equiv_L t \text{ if } \textbf{rules}(\tau) \subseteq \textbf{rules}(t) \text{ then } \textbf{rules}(t') \subseteq \textbf{rules}(\tau) \\ & & \quad \forall \tau \in \textbf{subtrees}(t'),\ s(\tau) = \tau\,. \end{array} \tag{25}$$

This is a non-circular definition of $s(t)$, because the recursion checks only proper subtrees of t', which are strictly smaller than t'. (In fact, although we do not require $s(t)$ to be computable, it is; the key to computing it is that, without actually enumerating the possibly-infinite set of all trees $\equiv_L t$ on given subset of g, one can decide whether any such trees exist, because this is just the membership problem for a CFL.)

It will be shown that $s(t)$ is defined for all trees t such that $\exists g \in \mathcal{G}(Z_{Wx})$ with $t \in \textbf{allows}(f_\top, g)$. Proceed by induction on the depth of t.

Base case. Suppose $t \in T_1$. Then t has no subtrees. $\mathbf{rules}(t) = \{t\}$, so the only $\tau \equiv_x t$ with $\mathbf{rules}(\tau) \subseteq \mathbf{rules}(t)$ is $\tau = t$. So $s(t) = t$.

Inductive step. Suppose $k \geq 2$, $t \in (T_k - T_{k-1}) \cap \mathbf{allows}(f_\top, g)$ for some $g \in \mathcal{G}(Z_{Wx})$, and the proposition holds for trees in T_{k-1}. By supposition, $|\mathbf{rbase}(t)| = 1$ and $\forall \tau \equiv_L t$ if $\tau \in \mathbf{allows}(f_\top, g)$ then $\mathbf{rbase}(\tau) = \mathbf{rbase}(t)$. Since $|\mathbf{rbase}(t)| = 1$, there exists a tree $t' \equiv_L t$ with $\mathbf{rbase}(t) = \{\mathbf{rules}(t')\}$. Let $t' = \langle n, w_0 t'_1 w_1 t'_2 \cdots t'_j w_j \rangle$, where the w_i are terminal strings and the t'_i are nonterminal trees. Each $t'_i \in T_{k-1} \cap \mathbf{allows}(f_\top, g)$, so by inductive hypothesis, $s(t'_i)$ exists. Let $t'' = \langle n, w_0\, s(t'_1)\, w_1\, s(t'_2) \cdots s(t'_j)\, w_j \rangle$. Then $t'' \equiv_L t$. By supposition, $\mathbf{rules}(s(t'_i)) \subseteq \mathbf{rules}(t'_i)$; and they both use the same top-level rule; therefore, $\mathbf{rules}(t'') \subseteq \mathbf{rules}(t')$. Also, t' was chosen so that $t'' \equiv_L t$ implies $\mathbf{rules}(t') \subseteq \mathbf{rules}(t'')$; therefore, $\mathbf{rules}(t'') = \mathbf{rules}(t')$. So t'' is a tree satisfying the candidate criteria for $s(t)$, and the only reason for it not be chosen as $s(t)$ is that some other candidate was enumerated first. Either way, $s(t)$ is defined.

Let f be the following parser.

$$f(u) \quad = \quad \{t \in T \mid s(t) = t\} \cap f_\top(u)\,. \tag{26}$$

By construction, f is cumulative on G. By construction of s, f is inductive on G. Suppose $g \in \mathcal{G}(Z_{Wx})$ and $t \in \mathbf{allows}(f_\top, g)$. Then $s(t)$ is defined. Further suppose $\tau \equiv_L t$ and $\mathbf{rules}(\tau) \subseteq g$. By the definition of s, $s(\tau) = s(t)$. By Theorem 8.5, $\tau \equiv_x t$. By construction of f, $s(\tau) = s(t)$ guarantees that (f, g) is unambiguous, and $\tau \equiv_x t$ guarantees that (f, g) is x-complete. So $f \in \mathcal{F}(Z_{Wx})$. ■

Theorem 8.9 Suppose Z_x and Z_y are completeness properties. If $Z_{Wx} \subseteq Z_{Wy}$ and $\mathcal{F}(Z_{Wy}) \neq \{\}$, then $\mathcal{F}(Z_{Wx}) \neq \{\}$. ■

Proof. Follows immediately from the definition of $\mathcal{F}$. ■

Theorem 8.8 implies that in order to maximize $\mathcal{G}(Z_{Wx})$ without nullifying $\mathcal{F}(Z_{Wx})$, trees t, τ with rule base of size 1 must be $\equiv_x$ exactly when they are $\equiv_L$ and $\mathbf{rbase}(t) = \mathbf{rbase}(\tau)$. Moreover, the equivalence classes of trees with larger rule bases *don't matter*, because grammars that support such trees are excluded from $\mathcal{G}(Z_{Wx})$.

There is no largest equivalence satisfying these conditions. To see why, suppose $t_1 \equiv_L t_2$ have different rule bases of size one, so that the conditions require $t_1 \not\equiv_x t_2$, and $\tau \equiv_L t_1$ has a larger rule base. Then, without violating the conditions, $\equiv_x$ can be chosen so that $\tau \equiv_x t_1$, and another $\equiv_y$ can satisfy the conditions with $\tau \equiv_y t_2$; but any equivalence $\equiv_z$ that contains both, $(\equiv_x \cup \equiv_y) \subseteq \equiv_z$, must have $t_1 \equiv_z t_2$, violating the conditions.

So there is no largest completeness property Z_x such that $\mathcal{F}(Z_{Wx}) \neq \{\}$. However, since the trees with large rule bases are exactly the ones that are excluded by all Z_{Wx}, there *is* a unique largest Z_{Wx} such that $\mathcal{F}(Z_{Wx}) \neq \{\}$, and there is a decidable equivalence that generates it.

Definition 8.10 Let $\equiv_R$ be the following equivalence on T_N: $\forall x, y \in T_N,\ x \equiv_R y$ iff $x \equiv_L y$ and $\mathbf{rbase}(x) = \mathbf{rbase}(y)$.

The generated completeness property Z_R is called *rulebase-completeness.* ■

A modified choice of $\equiv_R$, excluding all non-reflexive relations $t \equiv_R \tau$ where t has a large rule base, would have produced the *smallest* equivalence that generates the property of interest; but since trees with large rule bases are intrinsically not of interest, the formally simpler definition has been used.

Theorem 8.11 $\equiv_R$ is decidable. ■

Proof. Deciding $\equiv_L$ is trivial. Suppose $t \in T_N$. For each of the finitely many subsets h of $\mathbf{rules}(t)$, membership of w in the language generated by h from n is decidable. Therefore, $\mathbf{rbase}(t)$ is computable. ■

Theorem 8.12 Suppose Z_x is a completeness property. Then $\mathcal{F}(Z_{Wx}) \neq \{\}$ iff $Z_{Wx} \subseteq Z_{WR}$. ■

Proof. Suppose Z_x is a completeness property, $\mathcal{F}(Z_{Wx}) \neq \{\}$, and $(f, g) \in Z_{Wx}$. It will be shown that $(f, g) \in Z_R$.

Suppose $t \in \mathbf{allows}(f_\top, g)$. Since $(f, g) \in Z_x$, $\exists t' \equiv_x t$ such that $t' \in \mathbf{allows}(f, g)$. By Theorem 8.8, $|\mathbf{rbase}(t)| = 1$, and $\forall \tau \equiv_L t$ if $\tau \in \mathbf{allows}(f_\top, g)$ then $\mathbf{rbase}(\tau) = \mathbf{rbase}(t)$. In particular, $\mathbf{rbase}(t') = \mathbf{rbase}(t)$, and by definition of $\equiv_R$, $t' \equiv_R t$. Therefore $(f, g) \in Z_R$.

On the other hand, suppose Z_x is a completeness property and $Z_{Wx} \subseteq Z_{WR}$. By Theorem 8.9, it suffices to show that $\mathcal{F}(Z_{WR}) \neq \{\}$. The parser constructed in the proof of Theorem 8.8 is in $\mathcal{F}(Z_{WR})$. ■

Theorem 8.13 $\mathcal{G}(Z_{WR})$ is undecidable ■

Proof. The problem is to decide, for an arbitrary grammar $g \in \mathcal{G}$, whether or not $g \in \mathcal{G}(Z_{WR})$.

In the proof of [HoUl79, Theorem 8.9], ambiguity is shown to be undecidable for arbitrary pointed CFGs of the following form. Let $w_1, \ldots w_n$ and $x_1, \ldots x_n$ be two lists of terminal strings. Let $a_1, \ldots a_n$ be a list of terminal symbols that do not occur in any of the w_k or x_k. Let n, n_w, n_x be nonterminal symbols. Let g be the following set of rules.

$$\begin{array}{lll} & \langle n, n_w \rangle & \\ & \langle n, n_x \rangle & \\ \forall 1 \leq k \leq n, & \langle n_w, w_k n_w a_k \rangle & \langle n_w, w_k a_k \rangle \\ & \langle n_x, x_k n_x a_k \rangle & \langle n_x, x_k a_k \rangle \, . \end{array} \tag{27}$$

The pointed CFG has rules g and start symbol n.

For every terminal sentence v, there is at most one way to derive v from n_w, and at most one from n_x. Therefore, g is ambiguous iff the pointed CFG is ambiguous.

Every syntax tree t over g has $|\mathbf{rbase}(t)| = 1$; and there exist $t \equiv_L \tau$ over g with different rule bases iff g is ambiguous.

So $g \in \mathcal{G}(Z_{WR})$ iff g is unambiguous; and ambiguity of g is undecidable in general, therefore $g \in \mathcal{G}(Z_{WR})$ is undecidable in general. ■

9 Comments

A basic difficulty with extensible-syntax languages is that, in attempting to casually peruse source code written under an arbitrarily extended syntax, a programmer may not even be sure what the syntactic structure of the source code is.[4] This is an ergonomic problem before it can become a computational one, because the difficulty of compiling extensible-syntax languages —thus, of parsing some extensions, and of rejecting others— can only be judged once one knows which extensions are ergonomically admissible — hence, which must be parsed, and which must be rejected. Although the treatment here has been mathematical, the choice of which mathematical properties to study was based on conjectured ergonomic relevance to the programmer.

The question effectively addressed by this paper has been, given the particular well-behaved parsing criteria defined, *how much ambiguity* can be tolerated in the grammar? It was shown that there is a greatest tolerable degree of ambiguity (whereas one might have imagined that some kinds of ambiguity would be tolerable individually but not when taken together); and that whether or not an arbitrary grammar has the intolerable type of ambiguity is formally undecidable (which is not too surprising, since ambiguity in general is famously undecidable for CFGs).

Supposing, for a moment, that this mathematical sense of "tolerability" is in fact necessary and sufficient to render syntax extension ergonomically tolerable, there are two main strategies possible to cope with the undecidability result. The lazy strategy is to admit syntax extensions regardless of ambiguity, and then check for ambiguity on whatever particular source string is compiled; this exploits the fact that ambiguity on any one given sentence is decidable, even though the existence of ambiguous sentences is not. Ambiguity seems, however, to be an error in the syntax extension rather than in the source string, and as such the designer of the extension would want to know about the ambiguity at extension time. An eager strategy may therefore be preferable, which detects and rejects ambiguities when the grammar is extended. However, the grammar admission criterion must be decidable,

[4]Allusions to this problem were scattered through the extensible language symposia of 1969 and '71; for example, [Sha69], [Du71], [Wo71].

so it cannot be membership in $\mathcal{G}(Z_{WR})$; and the admission criterion cannot admit any grammar outside $\mathcal{G}(Z_{WR})$, because we are supposing well-behaved universal parsing ergonomically necessary; therefore, the admission criterion must be strictly stronger than membership in $\mathcal{G}(Z_{WR})$. One would prefer this stronger criterion to be not only decidable but *efficiently* decidable — say, checkable in time logarithmic or less in the size of the grammar. Further, it might be possible to choose the criterion so that it facilitates parser efficiency as well.

However, even if one stipulates that the mathematical criteria satisfy the purposes for which they were subjectively chosen, they weren't chosen to be ergonomically necessary and sufficient: they were chosen (by intent, whether achieved or not) to be ergonomically *necessary.* That is, it was supposed that parser behavior violating these criteria would be counter-intuitive for the programmer, and therefore undesirable; but no hypothesis was formed as to whether guaranteeing these behaviors would actually guarantee that the programmer would be able to cope with syntax extensions. Indeed, considering the arbitrary nature of the parsing algorithm described in the proof of Theorem 8.8, it does not seem to require behavior intuitively obvious to a human observer. One might therefore wish to stipulate for an extensible-syntax programming language that the *shape* of every syntax tree —that is, the arities of its parent nodes and contents of its leaves— must conform to a reasonably small context-free grammar. An example (not necessarily an exemplar) of a programming language with this property is Lisp, whose parse-tree shape is unambiguously determined by requiring unexcepted use of reserved parentheses.[5] Under that stipulation, the mathematical treatment here would be mostly trivial; eager admissibility-testing of extensions would be not only decidable, but probably logarithmic-time or better in the size of the grammar; and adaptive grammars, while possibly of interest for describing the language, would be superfluous to parsing it.

References

[ChrSh69] Carlos Christensen and Christopher J. Shaw, editors, *Proceedings of the Extensible Languages Symposium*, Boston Massachusetts, May 13, 1969 [*SIGPLAN Notices* 4 no. 8 (August 1969)].

[DeJo90] Pierre Deransart and Martin Jourdan, editors, *Attribute Grammars and their Applications* [*International Conference WAGA*] [*Lecture Notes in Computer Science* 461], New York: Springer-Verlag, 1990.

[5]The extreme parenophile syntax of Lisp isn't necessary for unambiguous tree shape, of course — although, somewhat ironically from the current perspective, efforts to reduce the density of parentheses in Lisp have often concentrated on introducing associative ambiguity with disambiguating operator-priorities (see [StGa93, §3.5.1]). One might actually produce more readable code if parentheses on most small expressions were left in place, and parentheses were eliminated instead from *large* expressions, perhaps using significant linebreaks and indentation, and other multi-line matching constructs such as braces or begin/end statements.

[Du71] J. J. Duby, “Extensible Languages: A Potential User’s Point of View”, in [Sc71], pp. 137–140.

[Ea70] Jay Earley, “An Efficient Context-Free Parsing Algorithm”, *Communications of the ACM* 13 no. 2 (February 1970), pp. 94–102.

[HoUl79] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Reading, Massachusetts: Addison-Wesley, 1979.

[Kn90] Donald E. Knuth, “The Genesis of Attribute Grammars”, in [DeJo90], pp. 1–12.

[OrCrAl07] A. Ortega, M. de la Cruz, and M. Alfonseca, “Christiansen Grammar Evolution: Grammatical Evolution with Semantics”, *IEEE Transactions on Evolutionary Computation* 11 no. 1 (February 2007), pp. 77–90.

[Sc71] Stephen A. Schuman, *Proceedings of the International Symposium on Extensible Languages*, Grenoble, France, September 6–8, 1971 [*SIGPLAN Notices* 6 no. 12 (December 1971)].

[Sha69] Christopher J. Shaw, Chair, “Extensible Language Symposium — Panel of Language Authors”, in [ChrSh69], pp. 37–39.

[Shu03] John N. Shutt, “Recursive Adaptable Grammars”, M.S. Thesis, WPI CS Department, 10 August 1993, emended 16 December 2003. Available (as of October 2008) at URL:
`http://www.cs.wpi.edu/~jshutt/thesis/top.html`

[Shu08] John N. Shutt, “Well-behaved parsing of extensible-syntax languages”, technical report WPI-CS-TR-07-14, Worcester Polytechnic Institute, Worcester, MA, October 2008. Available (as of October 2008) at URL:
`http://www.cs.wpi.edu/Resources/techreports.html`

[StGa93] Guy L. Steele Jr. and Richard P. Gabriel, “The Evolution of Lisp”, *SIGPLAN Notices* 28 no. 3 (March 1993) [Preprints, *ACM SIGPLAN Second History of Programming Languages Conference*, Cambridge, Massachusetts, April 20–23, 1993], pp. 231–270.

[We92] Wolfgang Wechler, *Universal Algebra for Computer Scientists* [*EATCS Monographs on Theoretical Computer Science* 25], New York: Springer-Verlag, 1992.

[Wo71] P. L. Wodon, “A Syntax Macro Processor”, in [Sc71] pp. 48–50.