

Some Theoretical and Practical Results in Context-Sensitive and Adaptive Parsing

Quinn Tyler Jackson

Jackson Solutions, Port Coquitlam, British Columbia
qjackson@shaw.ca

Abstract

We introduce a fifth language accepting machine called the PDA-T, demonstrate some of its interesting formal properties, and show its role in the §-Calculus¹. Based upon this new machine and the §-Calculus' other properties, we demonstrate the §-Calculus' formal Turing Power, and then propose a formal language classification (the §-Hierarchy), derived largely from the Chomsky Hierarchy, but with a fifth class of language accepted by the PDA-T. We show that this modified hierarchy yields several conceptual benefits over the standard four machine Chomsky Hierarchy. We also provide some practical examples of the use of §-grammars in context-sensitive and semantic parsing.

Keywords: *context-sensitive parsing, adaptive grammars, §-Calculus, push down automata, predicated parsing, Chomsky Hierarchy*

1 Introduction

The efficient parsing of context-sensitive languages holds the interest of many researchers, since it has application in natural language parsing [Boullier 1998], in the formalization of programming language semantics [Christiansen 1988 and many others], and in the solution of classically difficult to parse languages [Boullier 1999b]. Although linguistic formalisms such as those first introduced by [Chomsky] technically and theoretically allow for the parsing of all decidable languages, pure grammar formalisms can be somewhat cumbersome to work with when attempting to write correct grammars for non-trivial Type < 2 languages, and moreover, devising efficient algorithms to process the formalisms is a difficult problem.

Many different syntactic extensions to standard pure grammar formalisms have been suggested: from [van Wijngaarden] grammars, to attribute

grammars [Knuth], to extended attribute grammars [Watt & Madsen], and onward to self-modifying (or adaptive) models. Our own research has been in the area of an adaptive formalism called the §-Calculus. In this paper, we discuss this formalism more rigorously than we have to date, and demonstrate that §-grammars are a strong contender for those parsing tasks that require context-sensitivity brought about by an efficient, compact notation that also has shown itself in real-world parsing tasks to map well to efficient implementation.

1.1 Prior Art in Adaptive Models

Since [Christiansen 1990] and [Shutt] already adequately survey many adaptive grammar models, we will only briefly discuss adaptive formalisms that have already been proposed, without providing a comprehensive survey.

Adaptive language models have appeared under various names, including: modifiable grammars [Burshteyn], adaptable grammars [Christiansen 1990], dynamic grammars [Boullier 1994], recursive adaptable grammars (later called recursive adaptive grammars) [Shutt], Christiansen Grammars (so called by [Shutt]), and §-grammars [Jackson 2000a]. [Shutt] defines the more general *adaptive grammar model* as: “A grammatical formalism that allows rule sets (a.k.a. sets of production rules) to be explicitly manipulated within a grammar,” and further divides specific models into *imperative* (those whose rule sets vary over time) and *declarative* (those varying over space). Since Shutt's time or space division does not take into account that adaptive models may allow for grammars that vary in both time and space, we will define such grammars as time-space² adaptive grammars, rather than

¹ The § symbol, used in this context, is pronounced *meta-ess*.

² First coined in [Jackson 2000c] in an informal context and sufficient for our purposes.

as “imperative-declarative” adaptive grammars, which we feel would be an awkward appellation.

Another model, following findings first described by [Neto], which appears to be along the lines of research into Self-Modifying Finite Automata [Shutt & Rubinstein], evolved into adaptive formalisms as discussed in [Neto & Moraes], and [Iwai & Neto].

Various features of adaptive grammars have also surfaced in [Furse] and [Balmas], but these will not be discussed, except to note that adaptive features have appeared in various formalisms as augmentations to grammars use to achieve some end within a larger context.

[Carmi] recently introduced Adapser, an LALR(1) based adaptive parser, and has presented that system’s LR core as a fundamental strength of the Adapser system. However, since many of the above formalisms are proven to be Turing Powerful by their respective inventors, and we, too, later in this paper, will show the §-Calculus to be a Turing Powerful formalism, we shall avoid assumptions of a model’s utility based upon the underlying implementation, since all TP systems shall be held to be of theoretically accepting power above that of their core engines.

1.2 The §-Calculus

Definition 1.0: As first introduced in [Jackson 2000a] and later refined in [Jackson 2000b], [Jackson 2001], and [Jackson & Langan], and further refined in the present paper, a §-grammar is defined by the septuple³ $\xi = (V_N, V_T, C, S, Q, P, \Phi)$ such that:

- V_N is a finite set of *non-terminal variables*
- V_T is a finite set of *terminal variables*
- C is a potentially infinite set of sets (called *classes*) of sequences of terminal symbols or rules in the form allowable to P (below)
- $S \in V_N$ is a distinguished element of V_N called the *start symbol* (or *axiom*) of the §-grammar
- $Q \subset (V_N \cup C)$ is a distinguished potentially infinite set of *predicates* (or *terminal qualifiers*)

³ Formerly a sextuple. The component Φ is first introduced herein. Although the §-Calculus has been augmented, the properties of previous versions of the calculus still hold in this newer version.

- P is a set of *productions* of the form⁴ $X \rightarrow Y$ such that:
 - $X \in (V_N \cup C)^+$ and $Y \in (V_N \cup V_T \cup C \cup \Phi)^*$ (See note⁵.)
 - Y , through ϕ -expressions or through *binding expressions* in the form $\langle \alpha \rangle_\beta$, where $\alpha \in (V_N \cup V_T \cup C \cup \Phi)^*$, can (optionally) either populate or depopulate C , where population *adds* members to C , and depopulation *removes* members from C . Population may be either global in scope or local to the production, and may be conditional upon the successful acceptance of the invoking (or *super*) production.
- Φ is a finite set of *ϕ -expressions*. (Discussed in some detail later.) \square

2 Results

Our interesting results can be categorized as theoretical (new observations in language theory) and practical (empirical observations). We will focus herein mostly upon the first group of results.

2.1 Some Theoretical Results to Date

We have found that the traditional four accepting machines (that is: finite state automata, push down automata, linear bounded automata, and Turing Machines), although surely sufficient to describe all of the machines required to parse each of the four classifications of languages known as the Chomsky Hierarchy, leave a bit to be desired when one arrives at the Type 1 languages. Indeed, that the four class system has shown to be not entirely sufficient conceptually is seen in the fact that some models propose extensions to grammars in order to parse a subset of Type 1 languages that has come to be known as the *mildly context-sensitive languages* (*q.v.* [Boullier 1999a, 1999b]). We will now describe a formal extension to push down automata that has yielded several interesting results.

⁴ The allowable form of productions has changed since originally put forth in our previous papers.

⁵ Q was formerly included as a union expression in the definitions of both X and Y , but since $Q \setminus (V_N \cup C) = \emptyset$, this inclusion relation is implicit, and the formal definitions of X and Y no longer need include it.

2.1.1 Extending Push Down Automata with Name Indexed Tries

Definition 2.0: Let a PDA-T be a push down automaton that has been augmented such that it exhibits the following additional properties:

- The symbol \langle , known as the *anchor*, which is not in Σ , but may be in any production, which may be pushed to the stack when encountered in a production, and stores with it on the stack the absolute position of the parse in a stream of input.
- The symbol \rangle_{β} , known as the *cast*, which is not in Σ , but which must occur once for every corresponding leading \langle in a production, where β is a named index that refers to a trie. (β may also be in the forms (β) , $\beta!$, $-\beta$, and $-(\beta)$, described later in this definition.)
- Upon encountering the cast symbol, the machine behaves as if any reference to β that occurs in the production is either:
 - the string σ that begins at the input as marked by the corresponding anchor and ends at the point of that cast, or
 - the trie β .
- If, ultimately, the entire production matches, and β is not followed by the symbol $!$, the string σ is incorporated into the trie β ⁶. If, however, the index name is preceded by the symbol $-$, it is removed from the trie β . If the symbol is enclosed in parentheses, the most recent addition to trie β is fetched, and the expression (β) is first replaced by this substituted name instead. (If the β -expression is followed by the \uparrow symbol, committal to trie β occurs iff the *referring production* is accepted.) \square

What are some properties of this new machine PDA-T?

First, it is clear that PDA-Ts satisfy Shutt's criterion (*ibid.*) for grammar adaptability, since the extension to the machine allows for the explicit manipulation of rules in a grammar. The named trie β has all of the properties of a regular named production that is essentially a series of disjunct literals, so that the addition or deletion of strings to the trie has the effect of adding or removing simple productions

⁶ The set of all name-indexed tries is, in fact, the set C of the \S septuple given in Definition 1.0.

to or from the grammar. Furthermore, it can be proven that PDA-Ts accept some context-sensitive languages⁷:

Theorem 1.0: There exist PDA-Ts that accept context-sensitive languages.

Proof: Let $\mathcal{L} = \{wcw \mid w \in \{a,b\}^*\}$, a well known context-sensitive language [Boullier 1999a]. Let the \S -grammar \mathcal{G} be:

```
S → ⟨W+⟩X c X; // See Note8
W → a | b;
```

Figure 2.00

It is clear that \mathcal{G} is acceptable to a PDA-T, but not to a PDA, and that \mathcal{G} accepts \mathcal{L} . \square

As presented in Theorem 1.0, it is clear that a subset of PDA-Ts behave much in the same way the common regular expression extension known as *back-referencing* (*q.v.* [Gerdemann & van Noord]), and indeed, for those regular expression matchers that accept back referencing, the \S -grammar of Figure 2.00 could be written as:

```
([ab]*)c\1
```

Figure 2.01

which, according to Definition 2.0, would be expressed:

```
⟨(a|b)*⟩N! N
```

Figure 2.02

Theorem 2.0: There exist context-sensitive languages that are not accepted by any PDA-T.

Proof: Let $\mathcal{L}_2 = \{ww \mid w \in \{a,b\}^+\}$. (Note the subtle difference between \mathcal{L}_2 and \mathcal{L} above.)

```
S → ⟨W+⟩X X;
W → a | b;
```

Figure 2.03

For any input string $\{a,b\}^+$, W^+ will always accept that entire string and therefore it is always the case that $|X| \geq 1$. Once the entire string has been accepted, the only remaining possible accepting production would be required to accept λ , and since

⁷ Which obviously no PDA will do.

⁸ In previous papers, we used the notation $N \leftarrow S \rightarrow \langle W^* \rangle c N$ to denote what is now shown in this production.

$|X| \neq |\lambda|$ for any X, S can never be satisfied, there is at least one context-sensitive language not acceptable to any PDA-T. \square

Since PDA-Ts do not accept all context-sensitive languages, it may be tempting to classify the languages accepted by PDA-Ts as *weak* context-sensitive languages, however this would imply that the subset of languages acceptable to a PDA-T that are not also acceptable to a standard PDA is small. It is possible to show, however, that for every context-free language acceptable to a PDA, there is at least one corresponding context-sensitive language that is acceptable to a PDA-T.

Theorem 3.0: For every PDA-acceptable context-free language, there exists at least one PDA-T acceptable context-sensitive language.

Proof: Let \mathcal{L} be an arbitrary context-free language accepted by the machine $\text{PDA}_{\mathcal{M}}$. Let Σ be the finite alphabet of \mathcal{L} . Let δ be some symbol such that $\Sigma \cap \delta = \emptyset$. We can now construct a context-sensitive language $\mathcal{L}_2 = \{\langle \mathcal{L} \rangle_N \delta N\}$, which clearly not a context-free language but is acceptable to some machine $\text{PDA-T}_{\mathcal{M}}$. \square

So far, the languages under consideration have all involved languages that may include strings of indeterminate length. Are there any interesting properties of PDA-T acceptable languages that are of determinate length? It turns out that, in much that same manner that permutation phrases [Cameron] act as a compression notation for a finitely large underlying Type 2 grammar, PDA-Ts of determinate length also serve as a compression notation. (See also [Wolff].)

Theorem 4.0: If, for a \S -grammar \mathcal{G} that accepts \mathcal{L} , every marked subexpression of a PDA-T accepts input of determinate length, there exists a rewritten grammar \mathcal{G}_2 that is acceptable to a PDA that also accepts \mathcal{L} . (That is, if no trie in \mathcal{G} is of indeterminate size, there exists a Type 2+ grammar \mathcal{G}_2 that also accepts the \mathcal{L} .)

Proof: Consider the language $\mathcal{L} = \{w\delta w \mid w = \{\Sigma\}^n, 1 \leq n < \infty, \delta \cap \Sigma = \emptyset\}$. The number of possibilities for w is finite; if we represent each unique permutation for a given n as Σ_i , it is clear that a production

\mathcal{P}_i can be written in the form: $\mathcal{P}_i \rightarrow \Sigma_i \delta \Sigma_i$ for every w . The axiom of grammar \mathcal{G}_2 can be written in the form $S \rightarrow P_0 \mid P_1 \mid P_2 \mid \dots \mid P_k$, where k is the number of unique permutations formable from $\{\Sigma\}^n$. Since k is finite and a finite grammar of the form of \mathcal{G}_2 is clearly Type 2+, there is a finite expansion of G that accepts \mathcal{L} . \square

It is also interesting to note that if every reference to a trie β is substituted with the name of a production α , and α is formulated such that it adequately describes all binds to β , the context-free language that results from this substitution can be checked for correctness in the standard manner, and we can therefore know that the PDA-T grammar is at least as correct as its expanded counterpart.

Theorem 5.0: If a \S -grammar \mathcal{G} describes a language \mathcal{L} acceptable to a PDA-T, that grammar is at least as correctly formed as some corresponding context-free grammar \mathcal{G}_2 .

Proof: Let \mathcal{A} be the set of all α -expressions bound to the named trie β in \mathcal{G} . Let $n = |\mathcal{A}|$. Let \mathcal{B} be the set of all such sets \mathcal{A} . For every \mathcal{B}_i , we can construct a context-free production $\mathcal{P}_i \rightarrow \mathcal{A}_0 \mid \dots \mid \mathcal{A}_n$. If we then replace every expression $\langle \mathcal{A}_i \rangle_\beta$ with \mathcal{A}_i and replace every reference to \mathcal{B}_i with \mathcal{P}_i , it is clear that we now have a grammar \mathcal{G}_2 that also accepts at least \mathcal{L} . \square

There are languages acceptable to PDA-Ts that are interesting on their own merits (that is, languages that do not necessarily demonstrate any particular formal property, but are interesting on a pragmatic level). Consider the language $\mathcal{L}_{\text{pair}}$, defined by the PDA-T acceptable grammar $\mathcal{G}_{\text{pair}}$:

```
S → A+ ⟨W⟩N " is a " (N) .T "?" ;
A → ⟨W⟩N " : " ⟨W⟩(N) .T " ; ;
W → '[a-z]+' ; // See Note9
```

Figure 2.04

⁹ Expressions in single quotes are understood to be standard regular expressions.

In other words, $\mathcal{G}_{\text{pair}}$ accepts any of the following strings:

```
dog=canine;sedan=car;dog is a canine?
cat=feline;a=letter;a is a letter?
```

Figure 2.05

but rejects the following strings:

```
dog=canine;sedan=car;dog is a car?
cat=feline;a=letter;a is a thing?
```

Figure 2.06

As shown in [Jackson 2001], PDA-Ts can also be used to create what are therein called *disambiguating antecedents*, allowing for the proper parse of classically ambiguous sentences as “Fruit flies like a banana” and “Time flies like an arrow”:

```
The word "flies" is a noun.
Fruit flies like a banana.
The word "flies" is a verb.
Time flies like an arrow.
```

Figure 2.07

2.1.2 Extending §-Grammars with Predicates

If we further extend our formalism to allow the inclusion of predicate expressions in productions, further results are yielded. Our predicates are similar to those found in Range Concatenation Grammars [Boullier 2001 and 1999a], except that, unlike in RCGs, predicates are not parameterized, and there are no built-in predicates for such properties such as string length. In the §-Calculus, predicate expressions exist in two forms: *free* and *bound*.

Bound predicates are expressed in the form $\{\alpha\}^\theta$, where $\alpha = (V_N \cup V_T \cup Q \cup C \cup \Phi)^*$ of the § septuple and $\theta \in Q$, where Q is a set of distinguished productions in the grammar¹⁰. Consider the §-grammar \mathcal{G} :

```
S → {A}^W;
A → '.*';
W → "(" '[a-z]+' ")";
```

Figure 2.08

The production A matches any amount of anything. That matched substring is then tested against

¹⁰ That is, $Q \in (V_N \cup C)$, where V_N and C are those two components of the § septuple.

the predicate W , which only accepts strings in the form of an alphabetic sequence surrounded by parentheses. In other words, the *terminal lexeme* satisfied by A is further *qualified* by W (which is why predicates are also called *terminal qualifiers*). The language accepted by S is therefore the intersection of languages accepted by A and W .

Free predicates are expressed as a Ψ -expression in the form $\Psi(\alpha)^\theta$, where $\alpha \in \{V_N, V_T, C\}^*$ and θ is as for bound predicates. Unlike in the case of bound predicates, α does not accept input, but instead *generates* input that will be fed to θ by Ψ . Consider the §-grammar \mathcal{G}_2 :

```
S → ⟨A⟩N! Ψ(" (" N ") ")^W;
A → '.*';
W → "(" '[a-z]+' ")";
```

Figure 2.09

It is clear that \mathcal{G}_2 no longer accepts the same language as \mathcal{G} shown in Figure 2.08, even though W has not changed. The local bind to N of the lexeme that satisfies production A , being the *most recent* bind to N , is expanded by the Ψ -expression. The Ψ -expression generates input for W that is not the same input that was encountered in the input during the acceptance of A . Taken as a whole, \mathcal{G}_2 now is seen to accept strings of lowercase alphabetic of any length.

2.1.3 On the Notational Compactness of the §-Calculus

In practice, the §-Calculus has proven to be a concise notation.

Demonstration 1.0: Let $\mathcal{L} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$, a context-sensitive relationship referred to as *cross agreement*. Let \mathcal{G} be the §-grammar:

```
S → ⟨a+⟩w! ⟨b+⟩x! ⟨c+⟩y! Ψ(W Y)^A ⟨d+⟩z! Ψ(X Z)^B;
A → a [A] c;
B → b [B] d;
```

Figure 2.10

The language \mathcal{L} is expressed in the §-Calculus notation as three productions. Since none of the binding operations are global, the grammar varies over space, and \mathcal{G} is therefore declarative, by

Shutt's categorization. Moreover, in practice, when a parser generated by the Meta-S Grammar Development System¹¹ is presented with an input stream that contains \mathcal{L} prefixed with “junk” (that is, a string of potentially but ultimately non-matching “noise”), it is located within the stream in near $\mathcal{O}(n)$ time. \square

Demonstration 2.0: Let $\mathcal{L}_2 = \{ww \mid w \in \{a,b\}^+\}$, the context-sensitive redoubling language discussed in Theorem 2.0. Let \mathcal{G}_2 be the \S -grammar:

```
S → φ(λ)A.X R;
R → (⟨a|b⟩C! φ(A.X C)A.X φ(A.X)N! N) | R;
```

Figure 2.11

Of note are the ϕ -expressions¹² in the form $\phi(\alpha)_\beta$ (where $\alpha \in \{V_N, V_T\}^*$). For each distinguished β , the \S -grammar is understood to contain an implied production $\mathcal{P} \rightarrow \langle \omega \rangle_\beta$ and the ϕ -expression is then understood to actually be the Ψ -expression $\Psi(\alpha)^\mathcal{P}$. (The ω operator in $\langle \omega \rangle_\beta$ accepts any and all input, since it is the sole operator of the production.) Thus, every ϕ -expression has the subtle but important effect of binding to a named trie β even in the absence of an associated left subexpression in a production, which is to say, it behaves as an unconditional assignment to a trie β by means of an implied associated predicate. It follows that a \S -grammar containing a non-empty Φ set is not acceptable to a PDA-T, since the PDA-T definition says nothing of predicates. (More on this will follow later.) Since no trie may be referenced by α or by any production in the grammar until it has been part of a binding expression or ϕ -expression, the fixed bind of λ to $A.X$ in S (namely $\phi(\lambda)_{A.X}$) has the effect of instantiating the trie $A.X$ so that it may later be referenced in the production R as part of an α -expression.

The language \mathcal{L}_2 is thus expressible in the \S notation in as few as two productions. As in the case of Demonstration 1.0, in practice, when the

Meta-S parser accepts an arbitrary-prefixed \mathcal{L}_2 in near $\mathcal{O}(n)$ time. \square

Demonstration 3.0: Let $\mathcal{L}_3 = \{\{a_0, a_1, \dots, a_n\} \mid a_0 \geq 1, a_{i+1} - a_i = 1, i < n\}$. If we consider that in successor notation $1 = s(0)$, it is clear that $\mathcal{L}_3 = \{s(n), s(s(n)), \dots\}$ for any n , and we can express this language as an \S -grammar \mathcal{G}_3 thus:

```
S → φ("S")N φ("S")P "{" E " ";
E → φ(S.N)X! X '0-9'+ S.P φ(S.N "S")S.N
    φ(S.P " ")S.P [", " E];
```

Figure 2.12

Of note in the above \S -grammar is the scoping of the tries N and P . When referred to in the production that instantiates these tries, they are simply N and P . Outside the context of S , however, they must be referred to using a dot notation that specifies the context in which they were created. The use of the temporary bind to X in E . This is required due to the fact that $S.N$ is a trie, and not a simple string. Since when a trie is an argument in a ϕ -expression, it expands to the most recently committed member of the trie, $\phi(S.N)_{X!} X$ has the effect of creating a string in X , rather than the complete trie. (Otherwise, \mathcal{G}_3 would accept an entirely different language: a sequence of integers that are successively one higher than the highest integer already encountered *or* any of those integers that have already been encountered.)

As with Demonstrations 1.0 and 2.0, in practice, \mathcal{G}_3 accepts an arbitrarily-prefixed \mathcal{L}_3 in near $\mathcal{O}(n)$ time. \square

Demonstration 4.0: Let $\mathcal{L}_4 = \{a^n b^m c^{mn} \mid m, n \geq 1\}$.

Let \mathcal{G}_4 be the \S -grammar:

```
S → φ(λ)D.X A B Ψ(A.X)M C Ψ(D.X C.X)D;
A → ⟨'a+'⟩X!;
B → ⟨'b+'⟩X!;
C → ⟨'c+'⟩X!;
M → ("a" φ(D.X B.X)D.X)+;
D → "b" [D] "c";
```

Figure 2.13

Of note is M , an *adaptive predicate* [Jackson 2000b] that has the side-effect of generating a string

¹¹ A commercial grammar development system available at the URL: <http://www.sand-stone.com/Meta-S.htm>.

¹²As mentioned in Definition 1.0, ϕ -expressions are a new component of the \S -Calculus, first introduced formally herein.

of mn b 's. (For every a in $A.X$, one $B.X$ is appended to $D.X$.) For every b in the generated $D.X$, the predicate D accepts an equal number of c 's, and thus \mathcal{G}_4 accepts \mathcal{L}_4 . \square

With these extensions of the \S -Calculus (the PDA-T machine, and Ψ and ϕ -expressions) now firmly in place, we can proceed to our proof of the formalism's Turing Power.

Theorem 6.0: For every recursively enumerable set \mathcal{L} there exists an \S -grammar \mathcal{G} that recognizes \mathcal{L} .

Proof: It is known that for every recursively enumerable set there exists a Turing Machine that accepts that set. Let the Turing Machine \mathcal{M} be defined as the septuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$. Each of these seven components has a direct mapping to a \S construct:

- Q is a finite set of states. State in \S can be represented by some string σ in some named class \mathcal{S} .
- Σ is a finite alphabet. This alphabet can be represented in \S by a regular expression \mathcal{E} that accepts this same alphabet.
- Γ is a finite set of tape symbols. This can be represented in \S by two tries T_1 and T_2 that behave as stacks¹³, where T_1 represents the tape "to the left" of the read head of the machine, and T_2 represents the tape "to the right" of the read head, and the catenation $T_1T_2 = \Gamma$. Let the read head be represented as \mathcal{H} .
- δ is a transition function. This can be represented in \S by a predicate production \mathcal{P} that accepts a catenation of \mathcal{S} and \mathcal{H} , with each unique \mathcal{SH} mapping to a modifying set of statements that set the current state \mathcal{S} , and either "moving right" (popping from T_2 into a temporary register r , pushing r into T_1), "moving left" (popping from T_1 into r and pushing r into T_2), "erasing" the current symbol (popping from T_2 and push-

ing a blank into T_1), or replacing the current symbol (popping from T_2 , and pushing the new symbol into T_1).

- q_0 is the initial state. This can be represented in \S by setting \mathcal{S} to the start state of the machine before consuming any of the tape.
- B is the blank symbol. This can be easily represented in \S and requires no further explanation.
- F is a set of final states. This can be easily represented in \S and requires no further explanation.

It follows that, since each of the components of an arbitrary Turing Machine \mathcal{M} has a mapping in the \S -Calculus, for every \mathcal{M} there is a corresponding \S -grammar \mathcal{G} . \square

2.1.4 The \S -Hierarchy of Languages

The groundwork has now been laid for the introduction of a modified hierarchy of language types:

Definition 3.0: Let us break with tradition for a moment and modify the Chomsky Hierarchy, calling our modified version the \S -Hierarchy so that it can be distinguished from the standard hierarchy. It shall consist of:

- Those languages acceptable to a finite state automaton, which we shall call the \S -Type 3 languages, in keeping with the Chomsky Hierarchy's classification of such languages.
- Those languages acceptable to a push down automaton, which we shall call the \S -Type 2a languages.
- Those languages acceptable to a PDA-T, which we shall call the \S -Type 2b languages.
- Those languages acceptable to a linear bounded automaton, which we shall call the \S -Type 1 languages.
- Those languages acceptable to a Turing Machine, which we shall call the \S -Type 0 languages. \square

¹³ Since the most recently added member of any trie is returned when referenced the α -expression of any Ψ or ϕ -expression, it follows that that most recent addition behaves as any string would, and since a string may be used to emulate a "stack," by adding and removing items from its head, these tries also can emulate stacks.

If we diagram their relationships to one another as shown below in Figure 2.14, we can see that those languages that are classed by Definition 3.0 as the §-Type 2b languages fall into what are traditionally the Chomsky Type 1 languages:

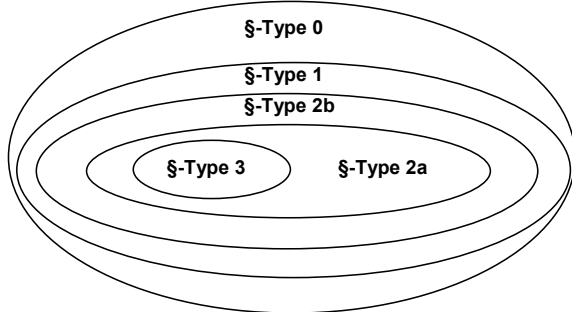


Figure 2.14

Is there any new insight offered by this slightly more granular classification? Indeed, by classifying some of what were once a subset of Type 1 languages as Type 2b, several observations become evident:

The §-Predicate Lemma: No §-grammar \mathcal{G} is §-Type < 2 unless $|\mathcal{G}.Q| \geq 1$. (Where $\mathcal{G}.Q$ is \mathcal{G} 's "Q set" in the § septuple.)

Proof: Consider the §-grammar shown in Demonstration 2.0, which, as already discussed, contains ϕ -expressions. These are really a notational shorthand for a special form of Ψ -expressions. For every ϕ -expression in an §-grammar, there is a corresponding *implied* Ψ -expression and its predicate. Thus, any §-grammar with a non-empty Φ set implicitly has a non-empty Q set. Since §-Type 2b are by definition PDA-T acceptable languages, and since, for PDA-T acceptable languages, $Q = \emptyset$, it follows that every §-Type 1 or 0 §-grammar must have a non-empty Q set. \square

The §-Predicate Lemma turns out to have some interesting implications. Since $Q \subset (V_N \cup C)$, we know that:

Theorem 7.0: Every §-Type < 2 §-grammar is a grammar-union of two or more §-grammars that are §-Type > 1 .

Proof: Every member \mathcal{P}_i of Q is effectively the axiom of a subgrammar within a §-grammar whose primary axiom is S . If any of these subgrammars in turn reference predicates, they effectively have Q sets of their own, recursively, until there exists

some subgrammar level such that $Q = \emptyset$. A consequence of the lemma is that every §-Type < 2 §-grammar consists of two or more §-Type > 1 §-grammars. \square

Why is it important to show that every §-Type 1 or 0 §-grammar is a grammar-union of §-Type > 1 §-grammars? Since Theorem 5.0 shows that such grammars are as correctly formed as their less restrictive context-free counterparts, we hypothesize that the number of context-sensitive languages that can be rigorously checked for ambiguity and other errors by checking their constituent §-Type 2+ subgrammars is greater than the number of languages that can be checked for such ambiguities using a pure grammar formalism such as Chomsky Normal Form. It is true that the \mathcal{G} of Theorem 5.0 potentially accepts fewer languages than \mathcal{G}_2 , since \mathcal{G} is more constrained than \mathcal{G}_2 , but since \mathcal{G}_2 at least accepts \mathcal{L} , checking for the standard errors in \mathcal{G}_2 at the very least verifies the overall correctness of \mathcal{G} . (That is if \mathcal{G}_2 is not correct, \mathcal{G} cannot possibly be.)

Also, since a practical implication of Theorem 4.0 is that many (what appear to be) context-sensitive §-grammars in fact contain tries of determinate length, these practical languages are therefore at least in principle acceptable to a PDA, even if a PDA-T is used in practice.¹⁴

Finally, since §-Type 0 and Chomsky Type 0 correspond exactly, we can know that a language is not Type 0 if it can be accepted by a §-grammar that contains no ϕ -expressions and has an empty Q set. This is not as constructive a proof system as the pumping lemma, since it relies in part on the grammar construction skill of human beings¹⁵, but it has some utility in practice.

2.2 Other Practical Results to Date

In Demonstrations 1.0 through 4.0 we gave examples of classical context-sensitive languages that are compactly expressed in the §-Calculus notation

¹⁴ Consider, for instance, that no programming language source file is infinitely long.

¹⁵ That is, ϕ -expressions and predicates are so useful in practice that it is tempting to use them when developing grammars even if they are not formally necessary. Eliminating them if they are not strictly needed is somewhat of an art.

and parsed in practice within optimal expected performances. Such canonical languages have, by far, not been the extent of our parsing successes with \S -grammars.

We have implemented an \S -grammar, expressed in A-BNF¹⁶, that accepts a dialect of a BASIC language we call Fermat.¹⁷ This grammar demonstrates a formal definition of a language that enforces these conditions::

- Variables must be of a declared or built-in type at declaration.
- Variables must be declared in any one of the following ways before use:
 - Globally,
 - As a parameter in the function header at function implementation,
 - By the use of a DIM statement.
- Variables can be declared once and only once in a given scope.
- Functions must be declared before use, in one of the following ways:
 - By the use of the declare function statement in the global scope,
 - By the implementation of a body before invocation.
- Function can be declared multiple times, but must only be implemented once.
- Boolean expressions are expected in the *expr* position of IF ... THEN END IF blocks, and boolean expressions must consist only of built-in boolean operations on variables, numeric variables, or functions declared as returning a BOOLEAN type.
- FOR NEXT blocks must have a NEXT that refers to the same variable as was referred to in the FOR statement.
- The iteration variable of a FOR statement must be of a numeric type.
- Any function that is part of a string concatenation expression must have been declared as returning a STRING. (Variables in such expressions must be of STRING type.)

¹⁶ Adaptive Backus-Naur Form: an extended form of EBNF that allows the expression of \S -Calculus operations in a form that is easily saved in flat ASCII files.

¹⁷ The \S -grammar described here is available from the URL: <http://members.shaw.ca/qjackson/cs/metasp/fermatparse.html>

The \S -grammar that effects all of the above checks consists of 63 productions and has shown near $\mathcal{O}(n)$ parse times in practice on cooked input.

In [Jackson 2002], we discuss an \S -grammar that begins accepting HTML *or* XML, but upon accepting an XML tag, accepts only XML (that is, stricter) style tags from that point in the parse forward. This experiment yielded an adaptive grammar that accepted properly formed XML in $\mathcal{O}(n)$ time.

3 Conclusions

In this paper we have introduced a fifth language accepting machine model, the PDA-T, and have shown some properties of this machine as it relates to our \S -Calculus. We have also further explored the formal underpinnings of the \S -Calculus, and have proposed a modified hierarchy of languages and briefly explored some of the \S -Hierarchy's implications. Finally, we have discussed some of the results of practical tests carried out with our adaptive parsing engine, the Meta-S Grammar Development System.

4 Areas for Future Work

As has been noted in [Jackson 2002], we feel work in grammar inference from input such as that done by [Nevill-Manning] should be further explored in order to determine if grammatical inference has any promise when intermingled with adaptive grammar theory. Some of our present findings suggest that it may be interesting to explore how the \S -Hierarchy fares when seen in this context. In this same area, although an error recovery mechanism has been placed into the parsing engine itself, its consequences should be further explored, since error repair (as opposed to simple recovery), in an adaptive grammar formalism, may benefit from work such as that of [Denis] and the genetic algorithm grammar inference of [Losee].

Finally, we feel that more investigation should take place on the implications of the five machine \S -Hierarchy, in order to determine if there are significant conceptual insights if this model is pursued further. Hypothesis 1.0 bears formal investigation. Our focus in this area will be to further investigate the formal properties of the \S -Type 2b languages, such as the notion that these grammars are a formal compression equation for \S -Type 2a grammars,

even for those that accept languages of indeterminate length [Wolff].

5 Acknowledgements

Many thanks go to Chris F. Clark, Boris Burshteyn, Shane F. Jackson, and Jeffrey A. Medina for their feedback on earlier drafts of this paper.

6 References

[Balmas] Françoise Balmas, “An Augmented Pattern Matcher as a Tool to Synthesize Conceptual Descriptions of Programs,” *Knowledge-Based Software Engineering Conference*, Monterey (CA), September 1994.

[Boullier 2001] Pierre Boullier, “From Contextual Grammars to Range Concatenation Grammars,” *Electronic Notes in Theoretical Computer Science*, Vol. 53, 2001.

[Boullier 1999a] Pierre Boullier, “A Cubic Time Extension to Context Free Grammars,” INRIA Research Report No. 3611, January 1999.

[Boullier 1999b] Pierre Boullier, “Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars,” *Proceedings of EACL*, 1999, pp. 53-60.

[Boullier 1998] Pierre Boullier, “Proposal for a Natural Language Processing Syntactic Backbone,” INRIA Research Report No. 3342, January 1998.

[Boullier 1994] Pierre Boullier, “Dynamic Grammars and Semantic Analysis,” INRIA Research Report No. 2322, August 1994.

[Burshteyn] Boris Burshteyn, “Generation and Recognition of Formal Languages by Modifiable Grammars,” *ACM SIGPLAN Notices*, 25/12, 1990, pp. 45-53.

[Cameron] Robert D. Cameron, “Extending Context-Free Grammars with Permutation Phrases,” *LOPLAS* Vol. 2 No. 1-4, 1993, pp. 85-94.

[Carmi] Adam Carmi, “Adapser: An LALR(1) Adaptive Parser,” *The Israeli Workshop on Programming Languages & Development Environments*, July 1, 2002.

[Chomsky] Noam Chomsky, *Syntactic Structures*, Mouton, The Hague, 1957.

[Christiansen 1990] Henning Christiansen, “A Survey of Adaptable Grammars,” *ACM SIGPLAN Notices*, Vol. 25 No. 11, November 1990, pp. 35-44.

[Christiansen 1988] Henning Christiansen, “The syntax and semantics of extensible languages,” *Datalogiske skrifter* 14, Roskilde University, 1988.

[Denis] François Denis, “Learning Regular Languages from Simple Positive Examples,” *Machine Learning*, Vol. 44 No. 1/2, 2001, pp. 37-66.

[Furse] Edmund Furse, “An Adaptive Parser for Mathematics,” Technical Report No. CS-93-4, Department of Computer Studies, University of Glamorgan, 1993.

[Gerdemann & van Noord] Dale Gerdemann & Gertjan van Noord, “Transducers from Rewrite Rules with Backreferences,” *9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, Norway, 1999.

[Iwai & Neto] Margarete Keiko Iwai & João José Neto, “*Introdução às Gramáticas Adaptativas*,” Technical Report, PCS-POLI-USP, São Paulo, Brasil, 2000.

[Jackson 2002] Quinn Tyler Jackson, “Efficient Formalism-Only Parsing of XML/HTML Using the \S -Calculus,” *Noesis-E*, Vol. 2 No. 2, June 2002.

[Jackson 2001] Quinn Tyler Jackson, “An Introduction to PAISLEI,” *Noesis-E*, Mar. 2001.

[Jackson 2000a] Quinn Tyler Jackson, “Disambiguation as a Quantifiable Computational Process,” *Perfection*, No. 3, Villeurbanne, France, March 2000.

[Jackson 2000b] Quinn Tyler Jackson, “Adaptive Predicates in Natural Language Parsing,” *Perfection*, Vol. 1 No. 4, April 2000.

[Jackson 2000c] Quinn Tyler Jackson, “Linguistic Time-Space,” *Perfection*, Vol. 1 No. 8, Sept. 2000.

[Jackson & Langan] Quinn Tyler Jackson & Christopher Michael Langan, “Adaptive Predicates in Empty-Start Natural Language Parsing,” *Noesis-E*, Vol. 1 No. 6, November 2001.

[Knuth] Donald E. Knuth, “Semantics of Context-Free Languages,” *Mathematical Systems Theory*, Vol. 2 No. 2, 1968, pp. 127-145.

[Lämmel 2001] Ralf Lämmel, “Grammar Adaptation,” *Proceedings of Formal Methods Europe (FME)*, Vol. 2021, 2001, pp. 550-570.

[Losee] Robert M. Losee, “Learning Syntactic Rules and Tags with Genetic Algorithms for Information Retrieval and Filtering: An Empirical Basis for Grammatical Rules,” *Information Processing & Management*, Vol. 32 No. 2, 1996, pp. 185-197.

[Nevill-Manning] C. G. Nevill-Manning, C.G. & I.H. Witten, “Inferring lexical and grammatical structure from sequences,” *Proceedings of Compression and Complexity of Sequences 1997*, Positano, June 1997.

[Neto] João Jose Neto, “Adaptive Automata for Context-Sensitive Languages,” *SIGPLAN Notices*, Vol. 29, No. 9, September, 1994, pp. 115-124.

[Neto & Moraes] João Jose Neto & Miryam de Moraes, “Formalismo adaptativo aplicado ao reconhecimento de linguagem natural,” *Anais da Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, Orlando, Florida, July 19-21, 2002.

[Shutt & Rubinstein] John Shutt & Roy Rubinstein, “Self-Modifying Finite Automata,” in B. Pehrson and I. Simon, editors, *Technology and Foundations: Information Processing '94* Vol. I: *Proceedings of 13th IFIP World Computer Congress*, Amsterdam: North-Holland, 1994, pp. 493-498.

[Shutt] John N. Shutt, *Recursive Adaptable Grammars*, Master’s Thesis, Worchester Polytechnic Institute, 1993.

[van Wijngaarden] Aad van Wijngaarden, “Orthogonal design and description of a formal language,” Technical Report MR 76, Mathematisch Centrum, Amsterdam, 1965.

[Watt & Madsen] David A. Watt & Ole Lehrmann Madsen, “Extended attribute grammars”, University of Glasgow, Report No. 10, July 1977.

[Wolff] J. Gerard Wolff, “Computing As Compression By Multiple Alignment, Unification And Search,” School of Electronic and Engineering and Computing Systems, University of Wales, 1995, available at: <http://citeseer.nj.nec.com/128596.html>