Simultaneous Productions: a Fully General Parsing Method to Make Progress on the Halting Problem

Daniel McClanahan

2020-06-08

1 Background

This paper will prove that the *S.P.* parsing algorithm is *streamable* and *cacheable*, and that the *S.P.* grammar-grammar is recursively enumerable (i.e. that *T.M.* can be reduced to *S.P.*). We will then prove that *streamability* and *cacheability* are sufficient to produce a parsing algorithm that can handle stack cycles in linear (???/whatever runtime we find) time. Finally, we will demonstrate that a *T.M.*'s runtime increases superlinearly (???) as k-context-sensitivity increases, thereby defining a strict superset of *T.M.*s called *S.P*s, of which *S.P.* is a member.

2 Concepts

2.1 The S.P. Grammar-Grammar

- *Describe why it's called a grammar-grammar, then describe the elements of the (simple) grammar-grammar, including nonterminals, terminals, ellipses, cases, and productions.*
- *Describe the relationship to the Chomsky formulation.*
- *Describe what differs from the Chomsky formulation.*
- *Describe the concept of stack cycles in an S.P. grammar.*
- *Describe the grammar-grammar in relationship to an S.P. fully-realized "grammar" vs e.g. a context-free grammar.*

2.2 ***Streamability*** and ***Cacheability***

- *Define streamability and cacheability as mathematical properties in terms of parsing algorithms in general.*

- *The point of these is to parameterize the qualities that S.P. has which other parsing algorithms lack. The idea is to make it more clear that S.P. is a *paradigm* of parsing, not a single algorithm.*
- *If possible, we want to prove the performance characteristics and correctness *in terms of* streamability and cacheability to demonstrate how to slot in a new "backend" for the algorithm.*

2.3 k-context-sensitivity

A context-sensitive language has at least one situation in which the parse tree can have multiple valid values for a sub-parse depending on the status of a super-parse. A k-context-sensitive language is one in which the depth of the stack that determines a sub-parse is bounded by a constant k. **TODO: VALIDATE!** In recursively enumerable languages, the depth of the stack of symbols needed to determine the correct sub-parse is instead bounded by the length of the input n.

3 The S.P. Parsing Algorithm

3.1 Architecture Overview

List and briefly describe the phases of the algorithm.

3.2 Data Structures and Techniques

- *lexicographic BFS / partitioning (cite Spinrad's book, etc)*

3.3 Phases

This part should be useful for implementors of the algorithm.

3.3.1 Preprocessing the S.P. Grammar

3.3.2 Setting up a Parse

3.3.3 Parsing

3.3.4 Resolving the Matched Input

4 Correctness

4.1 Proof of Streamability and Cacheability

4.2 Equivalence of S.P and T.M.

- *This demonstrates that S.P. can parse recursively enumerable languages.*

4.3 Equivalence of Stack Cycles and k-context-sensitivity

5 Performance

5.1 Parsing a Context-Free Language

5.2 Parsing a k-Context-Sensitive Language

5.3 Parsing a Recursively Enumerable Language

If "k-context-sensitivity" is general enough to cover this, we may not need a separate section.

5.4 Parallelism

Describe the runtime of the algorithm if k independent CPUs are provided.

6 Effect on the Halting Problem

- *Describe/Prove how the Halting Problem applies to T.M.s vs S.P.s, referencing the Performance section.*
- *Describe how S.P. was created by just thinking about having more than one state at a time (so giving insight into what underlying issues caused S.P. not to be found until now, and how others could have figured this out instead of me).*
- *Technically, this makes T.M.s a strict subset (!!!) of S.P.s that can only have a single state at a time and can only respond to the halting problem by running forever.* **THIS IS SUPER IMPORTANT AND POWERFUL!!!** *Make it clear that this applies to any construct that satisfies "streamability" and "cacheability".*

7 Conclusions and Future Work

- *Contrast S.P. to the Chomsky formulation.*
- *We define the "streamability" and "cacheability" properties.*
- *Those properties are shown to be (???) sufficient to create a construct better than a T.M.*
- *S.P. is an example of this superior construct.*
- *Describe how S.P. is the "holy grail" of parsing algorithms, and what parsing theory should focus on next.*
- *Mention the benchmarks paper.*

- *Mention running it backwards into a Monte Carlo Search Tree.*