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

Daniel McClanahan

2020-06-08

1 Relevant Prior Art / Notes

1.1 Overview

Recall that since we tried comparing the S.P. parsing algorithm to a T.M., there are two completely different use cases we run into with prior art: *parsing* and *execution*.

A 1.2 **Petri Net** is the closest model I have found to S.P.'s current *evaluation* method, i.e. how it models the parsing algorithm *after* preprocessing the provided S.P grammar, *upon* a given input. However, a 1.3 **Range Concatenation Grammar** appears to be the closest model I have found to the S.P. grammar-grammar (how grammars can be specified), and it seems close enough that **proofs of the RCG's Turing-equivalence may be transferable to S.P. (!!!!)**.

1.2 Papers

- **Freedom from the diagonal argument for circle-free Turing Machines:** *../literature/Martin Davis - The Undecidable_ Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions (2004)_ocr.pdf* – pages 137-138

- **Strong similarity to Petri Nets:** *../literature/Memo-95-decision-problems-petri-nets_ocr.pdf* – pages 21-22 *../literature/tr63_ocr.pdf* – "the reachability problem for vector addition systems is shown to require at least exponential space" – unclear still how vector addition systems relate to S.P.

- **Potential freedom from principles II, III, IV of the Gandy automata criterion for computability:** *../literature/gandy1980.pdf* – specifically page 133, but really all four principles. further inline notes in notability app

- **Potential termination proof (or not) from plotkin's powerdomains:** *../literature/plotkin-powerdomains-1976.pdf* – page 4, specifically "Now Konig's lemma says that if every branch of a finitary tree is finite, then so is the tree itself."

1.3 Wikipedia

- **description of a grammar-grammar VERY VERY SIMILAR to S.P.'s, which allows negation:** `https://en.wikipedia.org/wiki/Range_concatenation_grammars` – *without* negations is equivalent to a T.M., and a proof of equivalence could apply to S.P. (*separate* from the question of termination)
- **descriptions of unbounded automata, especially regarding termination of "infinite" sequences:** `https://en.wikipedia.org/wiki/Fair_nondeterminism` – notes on plotkin's result, as well as clinger: "Though each node on an infinite branch must lie on a branch with a limit, the infinite branch need not itself have a limit. Thus the existence of an infinite branch does not necessarily imply a nonterminating computation."
- **strong similarity to petri nets:** `https://en.wikipedia.org/wiki/Petri_nets`
- **the actor model doesn't really seem applicable, but the semantics of it might be:** `https://en.wikipedia.org/wiki/Denotational_semantics_of_the_Actor_model`
- **a closed actor system may represent S.P.:** `https://en.wikipedia.org/wiki/Indeterminacy_in_concurrent_computation`
- **chaitin's constant may be interesting:** `https://en.wikipedia.org/wiki/Chaitin%27s_constant`
- **it seems godel's results may imply that computability is "absolute" – is S.P. absolute?** `https://en.wikipedia.org/wiki/Church%E2%80%93Turing_thesis#complexity-theoretic_Church%E2%80%93Turing_thesis`
- **recursively enumerable sets:** `https://en.wikipedia.org/wiki/Recursively_enumerable`
- **cantor's diagonal argument:** `https://en.wikipedia.org/wiki/Cantor%27s_diagonal_argument`
- **halting problem:** `https://en.wikipedia.org/wiki/Halting_problem` – it's possible S.P. is immune to the naive result, as it can "analyze" the "else: loop forever" statement independently of the first branch
- **turing machine:** `https://en.wikipedia.org/wiki/Turing_machine`

- **quantified boolean formula:** `https://en.wikipedia.org/wiki/Quantified_Boolean_formula_problem` – interesting thought experiment for universal quantifier – is that the same power that S.P. has to infer a T.M.?

- **savitch's theorem:** `https://en.wikipedia.org/wiki/Savitch%27s_theorem` (!!!) – nondeterministic T.M.s only use a square root of the space of a deterministic T.M., as opposed to the possibly-exponential time bound difference between the two (!!!)

2 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.

3 Concepts

3.1 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.*

3.2 *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.*

3.3 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.

4 The S.P. Parsing Algorithm

4.1 Architecture Overview

List and briefly describe the phases of the algorithm.

4.2 Data Structures and Techniques

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

4.3 Phases

This part should be useful for implementors of the algorithm.

4.3.1 Preprocessing the S.P. Grammar

4.3.2 Setting up a Parse

4.3.3 Parsing

4.3.4 Resolving the Matched Input

5 Correctness

5.1 Proof of Streamability and Cacheability

5.2 Equivalence of S.P and T.M.

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

5.3 Equivalence of Stack Cycles and k-context-sensitivity

6 Performance

6.1 Parsing a Context-Free Language

6.2 Parsing a k-Context-Sensitive Language

6.3 Parsing a Recursively Enumerable Language

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

6.4 Parallelism

Describe the runtime of the algorithm if k independent CPUs are provided, taking into account the cost of moving data across cores.

This section will likely require an entirely separate analysis. **NOTE: It might be extremely enlightening to make this the first goal in analyzing the algorithm, and then consider the single-CPU case as a special case.**

7 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".*

8 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.*