Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Title: A Formally Verified Parser for a JavaScript Subset in a New Extensible Framework for Synthesizing Efficient Verified Parsers

Submitted by: J. S. Gross
258 Prospect Street, Apartment # 1L
Cambridge, MA 02139

Signature of author: _____

Date of Submission: August 19, 2015

Expected Date of Completion: August 2015

Laboratory where thesis will be done: CSAIL

Brief Statement of the Problem:

Parsers have a long history in computer science. This thesis proposes a novel framework for synthesizing efficient, verified parsers by refinement, and a demonstration of this framework by synthesizing a JavaScript parser competitive with existing open-source parsers. The benefits of this framework may include more flexibility in the parsers that can be described, more control over the low-level details when necessary for performance, and automatic or mostly automatic correctness proofs.

Supervision Agreement:

The program outlined in this proposal is adequate for a Master's thesis. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

A. Chlipala, Assist. Prof. of Elec. Eng. and Comp. Sci.

1 Introduction

The goal of this thesis is to construct an efficient verified-correct parser for the JavaScript language in Coq.

Much work has been put into the study and design of parsers in computer science. When computers were had less RAM and slower CPUs, many approaches to parsing involved handling only strict subsets of context free grammars, allowing guarantees of needing only a finite number of characters to decide how to parse at a given point in the string. More recently, the functional programming community has experimented with *parser combinators* [16], where parsers are built by combining a small set of typed combinators into higher-order functions. In the past decade or so, a number of new parsing approaches have been proposed or popularized, including parsing expression grammars (PEGs) [14], derivative-based parsing [19], and GLL parsers [25].

A few other past projects have verified parsers with proof assistants, applying to derivative-based parsing [3] and SLR [4] and LR(1) [17] parsers. Several projects have used proof assistants to apply verified parsers within larger programming-language tools. RockSalt [21] does run-time memory-safety enforcement for x86 binaries, relying on a verified machine-code parser that applies derivative-based parsing for regular expressions. The verified Jitawa [22] and CakeML [18] language implementations include verified parsers, handling Lisp and ML languages, respectively.

The proposed development will start from an algorithm that is essentially the same, algorithmically, as the one demonstrated by Ridge with a verified parser-combinator system [24]: the starting point is a naïve recursive-descent parser, combined with a layer to prevent infinite loops by pruning duplicate recursive calls. However, rather than fixing a particular algorithm for all grammars, this research will involve developing a framework for synthesizing parsers via refinement, specialized to the grammar being handled; one goal of this research is to demonstrate the viability of constructing parsers incrementally by refinement. As far as I know, no one has yet tried to build a framework for implementing verified parsers by refinement.

The target language for parsing will be JavaScript, which, to my knowledge, does not currently have an efficient, verified parser.

2 Timeline

The end result will be a parser of JavaScript, together with a proof of its correctness. To date, I have implemented the following:

- A formalization of context free grammars and parse trees
- A recursive-descent parser for an arbitrary context free grammar, parameterized on a splitter which determines the computational complexity and algorithmic strategy of the parsing algorithm
- A soundness proof and completeness proof for the parser

- A brute-force splitter that allows one to parse any grammar in exponential time
- A framework for incrementally refining splitters into more efficient splitters
- Refinement rules to handle fixed-length productions and productions with at most one non-terminal
- A linear parser for a grammar involving only parentheses and numbers

I plan to have the following timeline for incremental progress on efficient parsing:

Date	Additional Incremental Progress by That Date
May 5	Parse grammars involving +s and numbers
May 10	Parse grammars involving parentheses
May 20	Parse all arithmetical expressions in JavaScript
June 1	Formalize the JavaScript grammar
June 10	Parse variable assignments JavaScript
(June 12–14)	[Away for MIRI Decision Theory Workshop]
June 20	Performance Comparisons on Parsers
(June 22–30)	[Away for Coq Coding Sprint, Coq Workshop, UF/HoTT at TLCA 2015]
July 5	Parse sequences of statements, function bodies and function definitions
July 10	Submit POPL paper on the above
July 15	Parse function calls
July 20	Parse strings & Regular Expressions
(July 20–25)	[Away for Mathcamp]
August 2	Adapt POPL paper into thesis draft
August 4	Submit Masters Thesis
August 5	Masters Thesis Submission Deadline

3 Source of Data

For this thesis, the only data used will be pre-existing JavaScript parsers, such as [1, 2] and JavaScript programs to run comparison tests against.

References

[1] Esprima: ECMAScript parsing infrastructure for multipurpose analysis. `http://esprima.org/`.

[2] V8 JavaScript engine. `https://code.google.com/p/v8/`.

[3] José Bacelar Almeida, Nelma Moreira, David Pereira, and Simão Melo de Sousa. Partial derivative automata formalized in Coq. In *Proceedings of the 15th International Conference on Implementation and Application*

of Automata, CIAA'10, pages 59–68, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] Aditi Barthwal and Michael Norrish. Verified, executable parsing. In *Proceedings of the 18th European Symposium on Programming Languages and Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, ESOP '09, pages 160–174, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] Jean-Philippe Bernardy and Moulin Guilhem. Type-theory in color. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 61–72, New York, NY, USA, 2013. ACM.

[6] Samuel Boutin. Using reflection to build efficient and certified decision procedures. In *Proc. TACS*, 1997.

[7] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM (JACM)*, 11(4):481–494, 10 1964.

[8] Nigel P Chapman. *LR Parsing: Theory and Practice*. CUP Archive, 1987.

[9] Adam Chlipala. The Bedrock structured programming system: Combining generative metaprogramming and Hoare logic in an extensible program verifier. In *Proc. ICFP*, pages 391–402. ACM, 2013.

[10] The Coq Development Team. *The Coq Reference Manual, version 8.4*, Aug 2012. Available electronically at `http://coq.inria.fr/doc`.

[11] H. B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584–590, 1934.

[12] David Delahaye. A tactic language for the system Coq. In *Logic for Programming and Automated Reasoning*, pages 85–95. Springer, 2000.

[13] Benjamin Delaware, Clément Pit-Claudel, Jason Gross, and Adam Chlipala. Fiat: Deductive synthesis of abstract data types in a proof assistant. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 689–700. ACM, 2015.

[14] Bryan Ford. Parsing expression grammars: A recognition-based syntactic foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, pages 111–122, New York, NY, USA, 2004. ACM.

[15] William A. Howard. The formulae-as-types notion of construction. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. Original paper manuscript from 1969.

[16] Graham Hutton. Higher-order functions for parsing. *Journal of Functional Programming*, 2(3):323–343, July 1992.

[17] Jacques-Henri Jourdan, François Pottier, and Xavier Leroy. Validating LR(1) parsers. In *Proceedings of the 21st European Conference on Programming Languages and Systems*, ESOP'12, pages 397–416, Berlin, Heidelberg, 2012. Springer-Verlag.

[18] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 179–191, New York, NY, USA, 2014. ACM.

[19] Matthew Might, David Darais, and Daniel Spiewak. Parsing with derivatives: A functional pearl. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 189–195, New York, NY, USA, 2011. ACM.

[20] Robert C Moore. Removing left recursion from context-free grammars. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 249–255. Association for Computational Linguistics, 2000.

[21] Greg Morrisett, Gang Tan, Joseph Tassarotti, Jean-Baptiste Tristan, and Edward Gan. RockSalt: better, faster, stronger SFI for the x86. In *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, PLDI '12, pages 395–404, New York, NY, USA, 2012. ACM.

[22] Magnus O. Myreen and Jared Davis. A verified runtime for a verified theorem prover. In *Proceedings of the Second International Conference on Interactive Theorem Proving*, ITP'11, pages 265–280, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] Terence Parr, Sam Harwell, and Kathleen Fisher. Adaptive LL(*) parsing: The power of dynamic analysis. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 579–598. ACM, 2014.

[24] Tom Ridge. Simple, functional, sound and complete parsing for all context-free grammars. In *Proceedings of the First International Conference on Certified Programs and Proofs*, CPP'11, pages 103–118, Berlin, Heidelberg, 2011. Springer-Verlag.

[25] Elizabeth Scott and Adrian Johnstone. GLL parsing. In *Proceedings of the Ninth Workshop on Language Descriptions Tools and Applications*, LDTA '09, pages 177–189, 2009.

[26] Masaru Tomita. *Efficient Parsing for Natural Language: A fast algorithm for practical systems*, volume 8. Springer Science & Business Media, 2013.