Certified Parsing

Background

Parsing is the act of transforming plain text into some structure that can be analyzed by computers for further processing. One might think that parsing has been studied to death, and after *yacc* and *lex* no new results can be obtained in this area. However recent developments and novel approaches make it increasingly clear, that this is not true anymore.

We propose to on parsers from a certification point of view. Increasingly, parsers are part of certified compilers, like *CompCert*, which are guaranteed to be correct and bug-free. Such certified compilers are crucial in areas where software just cannot fail. However, so far the parsers of these compilers have been left out of the certification. This is because parsing algorithms are often ad hoc and their semantics is not clearly specified. Unfortunately, this means parsers can harbour errors that potentially invalidate the whole certification and correctness of the compiler. In this project, we like to change that with the help of theorem provers.

Only in the last few years, theorem provers have become good enough for establishing the correctness of some standard lexing and parsing algorithms. For this, the algorithms still need to be formulated in way so that it is easy to reason about them. In our earlier work about lexing and regular languages, we showed that this precludes well-known algorithms based automata. However we showed also that regular languages can be formulated and reasoned about entirely in terms regular expressions, which can be easily represented in theorem provers. This work uses the device of derivatives of regular expressions. We like to extend this device to parsers and grammars. The aim is to come up with elegant and practical useful parsing algorithms whose correctness can be certified in a theorem prover.

Proposed Work

A recent development in parsing is Parsing Expression Grammars (PEG), which are an extension of the weel-known Context Free Grammars (CFG) [6]. The extension introduces new regular operators, such as negation and conjunction, on the right-hand sides of grammar rules, as well as priority orderings. With these extensions, PEG parsing becomes much more powerful. For example disambiguation, formerly expressed by semantic filters, can now be expressed directly using grammar rules.

However, there is serious disadvantage of PEG for applications: is does not support grammrs involving left recursion [5]. Although a new PEG parsing algorithm has been proposed that can deal with left recursion [11], there is no correctness proof, not even in “paper-and-pencil” form. One aim of this research is to solve this sorry state-of-affairs by either certifying this algorithm or inventing a new one. For this we will first formalize a fixed point semantics of PEG, based on which an efficient, certified parsing algorithm can be given given. For this we take as starting point the paper [6], which does not treat left-recursion, but gives an operational semantics for PEG parsing. For the semantics, it seems plausible that we can adapt work on Boolean Grammars [9], which are similar to PEGs, and for which the paper [7] gives a semantics to negation operators, but not to Kleene’s star operation.

For the parsing algorithm, we might also be able to draw inspiration from parsers based on Cocke-Younger-Kasami (CYK) algorithms [7] and Early [4, 2] parsers. The defect CYK algorithms is that the original grammar specification needs to be transformed into a normal form. This transformation may lead to grammar explosion and inefficient parsing. We will investigate whether this transformation can be avoided. Early style parsers, which have recently been certified by Ridge [???], need to be extended to PEG parsing in order to be helpful for us.

Finally, we want to investigate whether derivatives of regular expressions [3, 1, 10, 8] can be extended to parsing. Lexing based on derivatives gives rise to very elegant regular expression matchers that can be certified in a theorem prover with ease. We will study whether the idea of taking a derivative of a regular expression can be extended to rules in grammars. The problem that needs to be overcome again arises from possible left recursion in parsing.

References

[1] J. B. Almeida, N. Moriera, D. Pereira, and S. M. de Sousa. Partial Derivative Automata Formalized in Coq. In *Proc. of the 15th International Conference on Implementation and Application of Automata*, volume 6482 of *LNCS*, pages 59–68, 2010.

[2] Aycock and Horspool. Practical Earley Parsing. *COMPJ: The Computer Journal*, 45, 2002.

[3] J. A. Brzozowski. Derivatives of Regular Expressions. *Journal of the ACM*, 11:481–494, 1964.

[4] J. Earley. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM (CACM)*, 13(2), Feb. 1970.

[5] B. Ford. Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, 2002.

[6] B. Ford. Parsing Expression Grammars: A Recognition-based Syntactic Foundation. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 111–122, New York, NY, USA, 2004. ACM.

[7] V. Kountouriotis, C. Nomikos, and P. Rondogiannis. Well-founded Semantics for boolean Grammars. *Inf. Comput*, 207(9):945–967, 2009.

[8] M. Might and D. Darais. Yacc is Dead. *CoRR*, abs/1010.5023, 2010. informal publication.

[9] A. Okhotin. Boolean Grammars. *Inf. Comput.*, 194(1):19–48, 2004.

[10] S. Owens, J. Reppy, and A. Turon. Regular-Expression Derivatives Re-Examined. *Journal of Functional Programming*, 19(2):173–190, 2009.

[11] A. Warth, J. R. Douglass, and T. D. Millstein. Packrat Parsers Can Support Left Recursion. In R. Glück and O. de Moor, editors, *PEPM*, pages 103–110. ACM, 2008.

[12] C. Wu, X. Zhang, and C. Urban. A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions (Proof Pearl). In *Proc. of the 2nd International Conference on Interactive Theorem Proving*, volume 6898 of *LNCS*, pages 341–356, 2011.