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 results and novel approaches make it increasingly clear, that this is not true anymore.

We propose to approach the subject of parsing from a certification point of view. Parsers are increasingly 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.

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 need to be formulated in way so that it is easy to reason about them. In earlier work about lexing and regular languages, the authors showed that this precludes well-known algorithms working over graphs. However 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 useful parsing algorithms whose correctness and the absence of bugs can be certified in a theorem prover.

Proposed Work

One new development in formal grammar is the introduction of Parsing Expression Grammar (PEG) as an extension of the standard Context Free Grammar (CFG)[6]. The extension introduces new regular operators such as negation and conjunction to the right hand side of productions, as well as well as an priority ordering on productions. With these extensions, PEG becomes more powerful such that disambiguation formerly expressed using semantic filters can now be expressed directly using production expressions. This means a simpler and more systematic treatment of ambiguity and more concise grammar specification for programming languages.

However, one disadvantage of PEG is that it does not allow left recursion in grammar specification, because the accompanying algorithms of PEG[5] can not deal with left recursions. Although some authors claimed new PEG parsing algorithm for left recursion[11], there is no correctness proof, not even in paper-and-pencil form. One aim of this research is to formalize a fixed point semantics of PEG, based on which an efficient, certified parsing algorithm is given.

There are several existing works we can draw upon:

1. The works on PEG.
   (a) An operation semantics for PEG has already been given in [6], but it is not adequate to deal with left recursions. But this work gives at least a precise description of what the original PEG meant for. This will serve an a basis to show the conservativeness of the fixed point semantics we are going to develop.
   (b) The new algorithm[11] which claimed to be able to deal with left recursions. Although there is no correctness proof yet, this may provide some useful inspirations for our new algorithm design.
2. The works on Boolean Grammar[9]. Boolean Grammar is very closely related to PEG, because it also contains negative and conjunctive grammars. The main differences are: First, Boolean Grammar has no ordering on productions; Second: Boolean Grammar does not contain STAR operator. There are two works about Boolean Grammar which might be useful for this research:
   (a) A fixed point semantics for Boolean Grammar[7]. The idea to define the semantics of negative and conjunctive operators is certainly what we can borrow. Therefore, this work gives the basis on which we can add in production ordering and STAR operator.

(b) A parsing algorithm for Boolean Grammar based on CYK parsing[7]. The draw back of CYK parsing is that: the original grammar specification needs to be transformed into a normal form. This transformation may lead to grammar explosion and is undesirable. One aim of this research is to see whether this transformation can be avoided. For this purpose, other parsing style may provide useful inspirations, for example:

   i. Derivative Parsing[3, 1, 10, 8]. Christian Urban has used derivative methods to establish the correctness of a regular expression matcher, as well the the finite partition property of regular expression[12]. There are well founded envisage that the derivative methods may provide the foundation to the new parsing algorithms of PEG.

   ii. Early parsing[4, 2]. It is a refinement of CYK parsing which does not require the transformation to normal forms, and therefore provide one possible direction to adapt the current CYK based parsing algorithm of Boolean Grammar for PEG grammar.

   iii. The new parsing algorithm proposed by Tom Ridge[???]. Recently, T. Ridge has proposed and certified an combinator style parsing algorithm for CFG, which borrows some ideas from Early parsing. The proposed algorithm is very simple and elegant. We are going to strive for a parsing algorithm as elegant as this one.

   Which of the above possibilities will finally get into our final solutions is an interesting point about this current research.

Based on these works, we are quite confident that our idea may lead to some concrete results.

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.