Ambiguity detection for programming language grammars
Basten, H.J.S.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)
Chapter 8

Conclusions

“I left the ending ambiguous, because that is the way life is.”
Bernardo Bertolucci

This thesis investigates ambiguity detection methods for context-free grammars. Its goal is to advance the usability of ambiguity detection to a level that is practical for checking grammars of real world programming languages. The main result of this thesis is a novel approach called AMBiDEXTER, which detects production rules that do not contribute to ambiguity. By filtering these rules from a grammar, the runtime of further ambiguity detection can be reduced significantly. Furthermore, we present DR. AMBIGNITY, an expert system that automatically proposes possible cures for ambiguity. This section concludes the thesis by summarizing our contributions, and discussing future work.

8.1 Contributions to Research Questions

In the introduction we posed a series of research questions that focus on improving the usability of ambiguity detection. The subsequent chapters each try to answer one or more of these questions. The following summarizes our contributions to each of the questions.

Research Question 1

How to assess the practical usability of ambiguity detection methods?

An ambiguity detection method (ADM) is practically usable on a given grammar if it can tell within a reasonable amount of time whether the grammar is ambiguous or not. Therefore,
performance, accuracy and termination are very important usability criteria. Furthermore, a method becomes more usable if it can be run with various accuracy settings, such that its behaviour can be adjusted to the available time and memory. Finally, the usefulness of an ADM increases if, in case of ambiguity, it can indicate the causes of the ambiguity in the grammar, and advise on possible solutions. All these usability criteria are discussed in Chapter 2.

Research Question 2

What is the usability of the state-of-the-art in ambiguity detection?

Also in Chapter 2, we investigate the usability of three different ADMs. Their implementations are tested on two sets of benchmark grammars: one set with 84 ambiguous and unambiguous grammar snippets, and one set containing 25 ambiguous and unambiguous variants of 5 real programming language grammars. The methods under investigation are the sentence generator AMBER by Schröer [Sch01], the Noncanonical Unambiguity (NU) Test by Schmitz [Sch07b], and the LR(\(k\)) test by Knuth [Knu65]. Their scores on each of the above mentioned criteria are measured and analyzed, and the methods are compared to each other.

Two of the investigated ADMs are quite usable on our grammars. The approximative NU Test shows good accuracy, performance, and termination characteristics, but is only able to decide unambiguity. On the other hand, the exhaustive sentence generator AMBER can only detect the existence of ambiguity, with reasonable performance, but it will never terminate on unambiguous grammars.

Research Question 3

How to improve the performance of ambiguity detection?

In Chapter 3 we present AMBIDEXTER, a novel ambiguity detection approach that uses grammar filtering to speed up exhaustive searching. It uses an extension of the NU Test that enables the detection of harmless production rules in a grammar. These are the rules that certainly do not contribute to ambiguity. If all productions of a grammar are identified as harmless then the grammar is unambiguous. Otherwise, the harmless rules can safely be filtered to produce a smaller grammar that contains the same ambiguities as the original one. This filtered grammar can then be searched with an exhaustive technique in less time, because of its smaller language.

The effectiveness of this approach is tested on the same set of programming language grammars that was used in Chapter 2. The results show that the filtering of harmless rules from these grammars significantly reduces sentence generation times, sometimes with several orders of magnitude.

In Chapter 5 we present a series of extensions to our grammar filtering approach, to make it suitable for filtering character-level grammars. These grammars include the full lexical definitions of their languages, and are therefore more ambiguity-prone. We present extensions for including disambiguation filters in the test, as well as a grammar unfolding technique to deal with the increased complexity of character-level grammars.
Again, an implementation of the extensions is evaluated on a series of real-world grammars. Although the reported gains in sentence generation time are not as large as for token-based grammars, our technique proves to be very useful on all but one grammar.

**Research Question 4**

*How to improve the accuracy of approximative ambiguity detection?*

In Chapter 4 we present the theoretical foundations for our grammar filtering technique and prove its correctness. We show how to extend both the Regular Unambiguity (RU) Test and the more accurate Noncanonical Unambiguity (NU) Test to find harmless production rules. With the RU Test our approach is able to find production rules that can only be used to derive unambiguous strings. With the NU Test it can also find productions that can only be used to derive unambiguous substrings of ambiguous strings. We also show that the number of detected harmless rules can be increased if the filtering is applied in an iterative fashion. This shows that grammar filtering has an accuracy increasing effect on the approximative RU Test and the NU Test.

Furthermore, the character-level extensions presented in Chapter 5 also increase the accuracy of our grammar filtering technique and the RU Test and NU Test in general. By taking disambiguation filters into account, the tests can ignore ambiguities that are already solved by the grammar developer. Furthermore, the general grammar unfolding technique increases accuracy by taming the approximation in relevant areas of the grammar.

**Research Question 5**

*How to improve the usefulness of ambiguity detection reports?*

Our contribution to this question is twofold. First, the harmless production rules produced by our grammar filtering approach serve as useful ambiguity reports, because they can give confidence about the unambiguity of certain parts of a grammar. Second, in Chapter 7 we present an expert system called DR. AMBIGUITY, that can automatically propose applicable cures for ambiguous sentences that are for instance found by a sentence generator. The chapter gives an overview of different types of ambiguity and ambiguity resolutions, and shows how they are linked together by DR. AMBIGUITY. The usefulness of the expert system is evaluated by applying it on a mature character-level grammar for Java. We remove several disambiguation filters from the grammar and test whether DR. AMBIGUITY is able to detect them as a possible solution. Initial results show that in all cases the removed solution was among the proposed cures.

**8.2 Discussion**

Despite the performance and accuracy characteristics on an ADM, its practical usability on a certain grammar also depends on the shape of the grammar, and available resources like time, memory and computing power. It is therefore very hard to determine whether the current state of the art is usable enough for checking real programming language grammars. Furthermore,
Chapter 8. Conclusions

because the ambiguity detection problem is undecidable, there are grammars for which one can never find a satisfying answer.

Nevertheless, as the results presented throughout this thesis show, grammar filtering is a general technique that can have very beneficial effects in many situations, independent of the checked grammar, available hardware or applied detection method. Filtering harmless production rules from a grammar can significantly reduce the runtime of exhaustive ADMs, as well as improve the accuracy of approximative ADMs. Furthermore, detected harmless production rules serve as a useful ambiguity report that guarantee the unambiguity of part of a grammar.

Our experiments also show that most grammars can already be filtered in an efficient manner on present day hardware. Also, a small investment in filtering time leads to the earlier detection of ambiguities in existing programming language grammars. Therefore, grammar filtering is ready to be included in language workbenches, preferably in combination with an exhaustive sentence generator and an expert system like DR. Ambiguity.

8.3 Future Work

An advantage of the undecidability of the ambiguity problem is that there will always be room for improvement. We envisage that research into improving the performance, accuracy and reporting of ambiguity detection will always be relevant. To encourage further investigations, we list a number of possible research directions.

ADM Comparison The evaluation of ADMs in Chapter 2 only includes the implementations of three methods. It would be interesting to also compare the methods of Brabrand, Giegerich and Møller [BGM10], Chueng and Uzgalis [CU95], CFG Analyzer [AHL08], and our own sentence generator described in Chapter 6.

Approximative Testing Although grammar approximation has already been studied quite substantially [MN01, Sch07a, BGM10, YJ03], we hope it is still possible to find better approximations that are suitable for ambiguity detection of programming language grammars. We see the following challenges and possible directions:

- Since we target grammars of programming languages, we could exploiting their typical characteristics. Very often, most syntactic structures of a programming language are regular, and context-free embedding is only used for scoping and expression nesting. If the regular structures can be separated from the context-free ones, only the latter have to be approximated.

- Furthermore, the effectiveness of various approximation improvements can vary a lot per grammar. It would be helpful if certain properties in a grammar can be found that indicate which type of approximation will result in suitable accuracy and performance.

- Another challenge for approximation techniques is to ignore known or intended [vdBKMV03] ambiguities in a grammar. This way, they can provide a guarantee that no other unknown ambiguities exist.
8.3. Future Work

• In Section 4.7 we discussed refining the approximation precision with every iteration of filtering a grammar. Like in model checking [CGJ+00], we envisage this can be done in an automatic way. More research is needed to investigate this approach in the setting of ambiguity detection.

• From process theory it is known that bisimulation is decidable for processes described by context-free grammars [HM95]. Bisimulation equivalence of grammars also implies language equivalence, but it is strictly weaker. Since language equivalence is closely related to the ambiguity problem, these results might be extended into an approximative ambiguity test.

Harmless Production Rules  The technique for detecting harmless production rules as described in Chapter 4 is able to detect production rules that can only derive unambiguous sentences or subsentences. However, there can also be productions that can derive ambiguous sentences while they are unambiguous themselves. These productions then only appear in parse trees above ambiguity nodes, and do therefore not directly contribute to ambiguity. We expect these productions can be found using the $\text{maEq}^*$ relations described in Section 4.4.2. More research is needed to work out this idea.

Sentence Generation  Sentence generation could possibly be sped up enormously by sharing the many common substrings that exist in the language of a grammar. This is presumably easier to achieve with breadth first searching. The results of Chapter 3 show that the breadth first sentence generator CFG ANALYZER [AHL08], which uses an incremental SAT-solver for its searching, performs very well. Our guess is that by using more domain knowledge about grammars in the searching, this SAT-based approach might be beaten.

Reporting  Initial experiments show that DR. AMBIGUITY is able of reporting all possible cures for an ambiguous sentence. However, which one to apply is a language design question, and also depends on its effects on other parse trees. More research is needed to help a grammar developer to make an informed choice from the — possibly long — list of available solutions.