A Note on the Parsing of Complete VHDL-2002

Luboš Lorenc[1], Rudolf Schönecker[1], and Zbyněk Křivka[1]

Dept. of Information Systems,
Faculty of Information Technology,
Brno University of Technology,
Božetěchova 1, 612 66 Brno, Czech Republic
lorenc@fit.vutbr.cz, schonec@fit.vutbr.cz, krivka@fit.vutbr.cz

Abstract. The paper gives a brief view on the process of analysis and adapting formal description of the current version of the VHDL language, described by VHDL 1076-2002 standard. The work searches the possibility to build the effective syntax analyser of the entire language by using automatic tools for parsers generation.

Keywords: computer languages, EBNF, LALR, parsing, VHDL

1 Introduction

VHDL is the acronym for VHSIC Hardware Description Language. It is a language that can be used to describe the structure of digital circuits or even behavior of the hardware design at the various levels of abstraction. It is a powerful language, but always restricted by the capabilities and aim of used tools.

The fundamental motivation for the construction of the complete VHDL parser is to be able to analyze arbitrary VHDL source code without dependency on the source tool. In past, there were introduced some methods (e.g. [4]), but only for very old version of VHDL standard. The concrete utilization is in automatic or semi-automatic VHDL-based modules generation that represents the user components design.

2 VHDL Language

The VHDL-2002 standard (see [1]) describing VHDL using EBNF consists of approximately 800 productions. Due to the repeated revisions and extensiveness of the description, the description is inconsistent on many places and contains errors. Used extensions allow to describe VHDL language in an advantageous and economical way, unfortunately they include both the syntax and also semantic features in the actual syntax description. These semantic features are not supported by parser generators nor other tools for dealing with grammars.

The form of the extension is given by italicized prefix of the nonterminal's name. The prefix carries crucial semantic information. For example, nonterminals

*type*_name and *subtype*_name are both syntactically equivalent to name, but carry the semantic information about the context in which they were derived. In this paper, call this extension *semantic condition.*

Semantic conditions are unusual in parsing theory (see [2] and/or [3]) and turn up some questions about the relation between poor syntax and on the other hand the semantic analysis when describing the VHDL language:

- Is it possible to describe poor VHDL syntax with context-free grammar?
- Is it possible to express the semantic conditions in current standard in another way to obtain LALR grammar that can be simply processed by parser generator tools?
- Is there any other way to parse entire VHDL?

3 Experiments with Parsing

In the case of our fundamental research, the existing EBNF description of the VHDL-2002 language has been transcripted into the input grammar for the Bison tool. The semantic conditions were left unchanged in this step.

The second part of the transformation had to separate the syntax part of the description from the semantic information. All nonterminals with italicized prefixes were renamed onto original names without these prefixes. In this moment the grammar stopped to remember the context of transformed nonterminals, this context has to be added later.

The context had previously influenced the selection process of currently used rule when deriving sentential form by parsing the input VHDL source. The consequence is that the transformed grammar became ambiguous due arisen reduce/reduce conflicts. The basic task is to eliminate the ambiguity in the grammar and simultaneously preserve the ability of the grammar to generate the entire VHDL-2002. The grammar constructed by standard rewriting of the EBNF contains 743 productions, 78 shift/reduce conflicts and foremostly 579 reduce/reduce conflicts.

The huge ambiguity in VHDL grammar is solved in two independent ways. The former uses conditional building of syntax tree in the process of syntax analysis. For making decision about the next advance of its production there is used some semantic knowledge acquired by the analysis of already processed source code. The latter way uses consecutive merging of all corresponding conflicting syntactic categories into one. This way consecutively eliminates ambiguities but it also removes the original semantic dependencies established in the VHDL-2002 standard.

3.1 Conditional Building of the Syntax Tree

This method of semantic conditions elimination utilizes inserting of new auxiliary terminal symbols to the input of the parser. It means that the output of the lexical analyser is not directly connected to the input of the parser but there is an

auxiliary generator that they are connected through. This generator is tightly bound to the semantic analyser and syntactic analyser and during analysis it inserts auxiliary terminals at the right places in the input stream of tokens (simplified schema of such analyser is shown on the figure 1). The binding between auxiliary generator and parser has to be created as special semantic actions in the parser code.

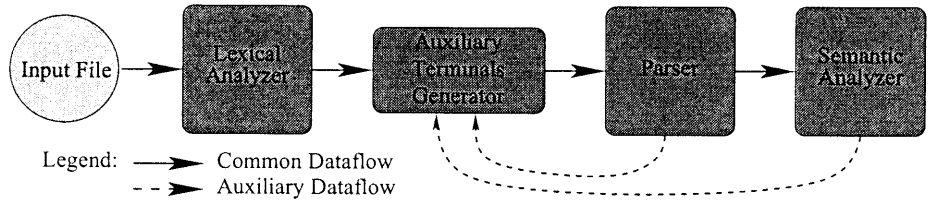

Fig. 1. Simple schema of an partially complete VHDL analyser with generator of auxiliary terminals.

The auxiliary terminals are inserted in the grammar immediately after conflicting (really only after conflicting not all semantically conditioned) nonterminals originally containing semantic conditions. Each inserted terminal has its name deduced from this semantic condition. Consequently during analysis, if it is detected such a nonterminal is in progress, the auxiliary terminal generator is activated. The decision whether insert an appropriate auxiliary terminal or not is based on the information provided by the semantic analyser. For example, there is shown a solution of conflicts in part of VHDL-2002 grammar containing nonterminals *type*_name and *subtype*_name in the Example 1.

Example 1. Solution of the typical reduce/reduce conflict in VHDL grammar. The symbols _TYPE_ and _SUBTYPE_ are the new auxiliary terminals.

```
subtype_declaration : SUBTYPE id IS subtype_indication ';'
      ;
subtype_indication ⇒ ... ⇒ type_mark
type_mark : name _TYPE_
      | name _SUBTYPE_
      ;
report_statement : REPORT expression ';'
      ;
expression ⇒ ...⇒ primary
primary : name
      ;
```

□

There was eliminated all the reduce/reduce conflicts in the VHDL grammar using this method. But practical usability of this method has not been yet

verified because it needs a working basic semantic analyser providing sufficient information to the auxiliary generator. However, the semantic analysis was not the main focus of this research phase and will be handled in the future.

This method seems to be a good candidate for the construction of a complete analyser of VHDL, because it does not modify original form of VHDL grammar. Therefore, it may be possible to use it as a base of an universal tool enabling both simulation and synthesis without the need of separate working with some VHDL code for simulations and another VHDL code for synthesis.

3.2 Creation of an Unambiguous Grammar

It is certainly the best solution to use an unambiguous grammar as a base of a parser. Thus, we are working on an unambiguous version of VHDL grammar. Currently, we have solved about 500 reduce/reduce conflicts and 76 remain for future work. As a result of this work, there should be an universally usable VHDL grammar suitable as a base of standalone syntactic analyser of VHDL. Because of the big number of modifications in the grammar structure, it is supposed to be too awkward as a base of complete VHDL analyser or synthesis tool.

As in the first approach, here is one big problem too. The grammar is too complex and eliminating of conflicts is very hard task. In addition, it is not known whether it is possible to describe entire VHDL using LALR grammar or not. There are some additional possibilities like for example generalized LR parsing provided by Bison, but these powerful analysers have generally very poor performance in case of complex languages with many conflicts.

4 Conclusions

VHDL is a very complex language. It contains a number of semantic enhancements in its standard syntax definition. From this fact follows that the process of the universal VHDL parser creation is very hard task. Currently, we have found some ways which may tend to achieve this goal. These ways were briefly described in this paper same as some open problems.

This work has been supported by Ministry of Education, Youth and Sports of the Czech Republic grant MŠMT 2C06008 "Virtual Laboratory of Microprocessor Technology Application".

References

1. IEEE Std 1076-2002 (Revision of IEEE Std 1076, 2000 Edition), *IEEE Std VHDL Language Reference Manual*, http://stdsbbs.ieee.org/descr/1076-2002
2. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman: *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 2006, ISBN 0-321-42890-0
3. *Bison, GNU parser generator*, http://www.gnu.org/software/bison
4. Rodney Farrow, Alec G. Stanculescu: *A VHDL compiler based on attribute grammar methodology.* In: Proceedings of the ACM SIGPLAN 1989, ACM Press, New York, 1989, pp. 120-130, ISSN 0362-1340