Syntaktická analýza VHDL

Luboš Lorenc, Rudolf Schönecker, Zbyněk Křivka

WFM ’07, Hradec nad Moravicí

April 23 - 25

Outline

1. Syntactic analysis of VHDL
    - VHDL grammar
    - R/R conflicts in VHDL
    - Solving of R/R conflicts
2. Conditional building of syntactic tree
    - Example
    - How it works . . .
    - Results
3. Open problems

VHDL2002

- Probably impossible to describe complete VHDL2002 language using context-free grammar.
- Standard is defined using an semantically extended EBNF grammar.
- Particular derivations are permitted using semantic extensions.
- Impossible to remove semantic conditions:
    - Rising of a few hundred of R/R conflicts

A typical VHDL2002 R/R conflict

```
start ⇒ ··· ⇒ type_mark ';'
start ⇒ ··· ⇒ report_statement

type_mark
    :  type_name
    | subtype_name
    ;
report_statement
    :  REPORT expression ';'
    ;

expression ⇒ ...⇒ primary

primary
    :  name
    ;
```

Solving of VHDL2002 R/R conflicts

- It is impossible to left out the semantic information
- There was used two distinct approaches:
  - Conditional building of syntactic tree
    - It seems to be a hopefull way
    - But we are in doubt about realisation
  - Creation of an unambiguous grammar
    - Meanwhile not succesfull way
    - There still remains unsolved R/R conficts
- Probably, there are some other approaches:
  - Use of another type of analyser
  - Use of huge lookahead
  - Analysis based on context-sensitive grammmars
  - ...

The use of auxiliary terminals — example

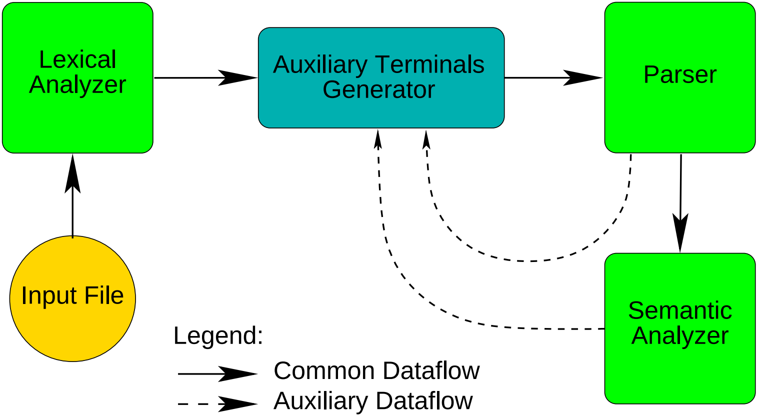
```
start ⇒ ··· ⇒ type_mark ';'
start ⇒ ··· ⇒ report_statement

type_mark
    :  name _TYPE_
    | name _SUBTYPE_
    ;
report_statement
    :  REPORT expression ';'
    ;

expression ⇒ ...⇒ primary

primary
    :  name
    ;
```

How it works . . .


Lexical Analyzer
Auxiliary Terminals Generator
Parser
Input File
Semantic Analyzer
Legend:
Common Dataflow
Auxiliary Dataflow

Conditional building of syntactic tree — results

- All the R/R conflicts solved
- Input file does not contain auxiliary terminals
- Semantic actions of conflicting rules are used to insert auxiliary terminals into input stream
- There is not fully solved needed semantic analysis yet
- Problems may be arisen by an impact of semantics to the syntax (context-sensitive dependencies)

Open problems

- Using auxiliary terminals, there is removed ambiguity from the grammar. But there are two ways how the original VHDL grammar has been created:
    - Authors of VHDL2002 created the grammar as a true *unambiguous context-sensitive grammar*:
        - Our grammar with auxiliary terminals should be able to simulate the behavior of such a grammar.
        - The parser should be able to analyze the whole VHDL2002 language.
    - Authors of VHDL2002 created the grammar in some another manner:
        - ???