Static Analysis of VHDL

Dipl.-Inform. Marc Schlickling
schlickling@cs.uni-sb.de

Compiler Design Lab
Prof. Dr. Reinhard Wilhelm
Universität des Saarlandes
Content

- Data flow analysis
- Control Flow Representation Language
- Program Analyzer Generator
- Frontend
  - Parser
  - Generation of Crl2
- VHDL Semantics
- Analysis framework
- Example: Constant Propagation

Static Analysis of VHDL
Goal

- Using abstract interpretation for analyzing arbitrary VHDL code
- Finding abstractions for simplifying the given VHDL code
Data flow analysis (1)

- A technique for collecting run-time information about data in programs without actually executing them

- Based on a control flow graph (CFG) describing a superset of paths through a program
  - Nodes describe program statements
  - Edges describe possible control flow
Data flow analysis (2)

- Each node of the CFG gets assigned a transfer function transforming the input data abstraction to a new output abstraction

\[ dfi_{\text{post}} = \text{transfer}(dfi_{\text{pre}}) \]

- Different techniques to compute solution
  - In this talk: fixed point iteration
Crl2

- Crl2 was designed by AbsInt GmbH
- Provides textual description of a control flow graph
- Hierarchically organized in:
  - Routines
  - Basic blocks
  - Instructions
  - Edges (of different types)
- Easy extendible with an attribute-value concept
PAG

- Analogous to flex and bison
- For generating efficient program analyzers
- Crl2 frontend available for translating Crl2 descriptions into data-structures describing the control flow graph
Example: Constant Propagation (1)

- Analysis determines for each program point whether a variable has a constant value whenever execution reaches that point.

- Domain: $\text{Var} \rightarrow (\mathbb{Z} \cup \{T\}) \cup \{\perp\}$

  - $\perp$ indicates not yet considered program points.
  - $T$ indicates, that there is no constant value for that variable determinable.
Example: Constant Propagation (2)

DATLA-specification

SET

DOMAIN
value = flat( snum )
dom = str -> value

OPTLA-specification

PROBLEM ConstProp
direction: forward
carrier: dom
init: bot
init_start: [ -> top ]
combine: lub
equal: eq(a,b) = a = b

TRANSFER
normal_node(_):
    //do something
Reminder

- Data flow analysis
- Control Flow Representation Language
- Program Analyzer Generator

Frontend
- Parser
  - Generation of Crl2

VHDL Semantics
- Analysis framework

Example: Constant Propagation

Static Analysis of VHDL
Frontend

- Converting VHDL to Crl2
- Organized into 4 stages
  - Parsing
  - Static semantics check
  - Elaboration
  - Generation of Crl2
- Each stage
  - Fully visualizable
  - Provides mechanism for reconstruction of VHDL code
Parser

- Syntactic modifications of VHDL code
  - Transformation of 'switch'-statements into 'if'-statements
  - Conversion of 'concurrent signal assignments' and 'concurrent procedure calls' into processes

- Implementation:
  - Syntax tree consists of 80 classes
  - Dynamic reloading of VHDL-libraries
  - Already parsed modules are stored in a Design Library
    (Leon: 275 modules, ≈ 7Mb)
Static semantics

- Checking is done during parsing
- Organized in
  - Type checking of expressions and assignments
  - Type checking in presence of overloaded functions

→ Bachelor thesis
Elaboration

- According to VHDL standard 1076
  - Recursively embedding all component instantiations
  - Unrolling `generate`-statements
  - Unifying all occurring names

- For static analysis
  - „Unrolling“ of record-structures
  - Resolving range-attributes to concrete ranges

for i in d`RANGE ⇒ for i in d`LEFT to d`RIGHT
Visualization

- Debugging:
  ```vhdl
  variable x is
  std_logic_vector(0 to 1);
  case x is
  when "00" => A
  when "01" => B
  when others => C
  end case;
  ```

- Traceability:
  ```vhdl
  variable x is
  std_logic_vector(0 to 1);
  case x is
  when "00" => A
  when "01" => B
  when others => C
  end case;
  ```
Visualization

- **Debugging:**

```
variable x is
   std_logic_vector(0 to 1);
```

```
case x is
   when "00" => A
   when "01" => B
   when others => C
end case;
```

- **Traceability:**

variable x is
   std_logic_vector(0 to 1);

```c
case x is
   when "00" => A
   when "01" => B
   when others => C
end case;
```
Visualization

- **Debugging:**

```vhd
variable x is std_logic_vector(0 to 1);
if x = "00"
  then A
elsif x = "01"
  then B
else C
endif;
```

- **Traceability:**

```vhd
variable x is std_logic_vector(0 to 1);
case x is
  when "00" => ...
  when "01" => ...
  when others => ...
end case;
```
Visualization

- Debugging:

```vhdl
variable x is std_logic_vector(0 to 1);

if x = "00"
then A
elsif x = "01"
then B
else C
endif;
```

- Traceability:

```vhdl
variable work_package_x is std_logic_vector(0 to 1);

if work_package_x="00"
then A
elsif work_package_x="01"
then B
else C
endif;
```
Generation of Crl2 (1)

- Organized into 3 stages
  - Generating the required routines
  - Generating the instructions and the basic block structure
  - Generating the analysis framework

- For static analysis
  - Transforming loops into recursive routines
  - Attribution of generated instructions, blocks and routines
    - definition-use classification
    - etc…
Generation of Crl2 (2)

- Processes, functions and procedures are mapped to routines
- Statements are mapped to instructions
- Basic block structure can be automatically derived from the number of predecessor / successor statements
- Function and procedure calls are modeled via call/return nodes in Crl2
VHDL Semantics (1)

- Two level semantics
  - Process execution
  - Synchronization + Restart + Time

- Process Execution
  - Sequential, imperative semantics
  - But: assignments to signals are delayed
  - I.e. we have 'future' values for signals

- Process executes until suspended
  - Suspension by 'wait'-statement
VHDL Semantics (2)

- **Second level**
  - After all processes have suspended
  - Check if a signal's value changed which is in sensitivity list of a process
    - Yes: restart all such processes (Delta Cycle)
    - No: wait for timeout (Theta Cycle)
  - Repeat :-)

- **Observations:**
  - Ordering of process execution not in semantics
  - Simulators for VHDL make this semantics sequential
Transformed Semantics (1)

- Ordering of process execution not important
  - Variables are local
  - Signal assignments take effect only at synchronization
  - I.e: we can choose a fixed ordering

- We can view signal assignments as assignments to 'new' signals
  - \( S <= '1' \iff S_{\text{new}} := '1' \)
  - At synchronization we copy new->act
    - \( S := S_{\text{new}} \)
Transformed Semantics (2)

- Two level semantics difficult
  - Transform to one level sequential semantics
- Upper level 'pushed down'
- Always execute processes in fixed order in a loop
- Add 'guard' to process header to check if we need to execute process in this loop iteration
  - Guard true, if process restarted at synchronization of previous iteration
Sequential Semantics

- Move all 'wait'-statements to one big wait at end of loop body

PROC_IF (s11, s12, s13)
   P1

PROC_IF (s21, s22)
   P2

PROC_IF (s31, s32, s33, s34)
   P3

s11_old := s11
s11 := s11_new
...

WAIT (s11, s12, ...)

If s11_old /= s11 or s12_old /= s12 or s13_old /= s13 THEN
   P1

Synchronization of signal values

If s11 == s11_old and s12 == s12_old and ...
   Then
   wait for timeout Loop

Static Analysis of VHDL
Analysis framework (1)

- Modelling the sequential execution by a special 'simul' process
  - Each "process execution" is guarded by a 'simul_if' modelling the sensitivity list of the process
  - The analyzer decides, if the true edge to the call is taken or not
- 'wait'-statement is represented by the 'simul_wait' node
Analysis framework (2)

- Problem
  - Only closed design can be analyzed
- For analyzing partial design, e.g. entities, there has to be a special process simulating the environment
- Using a transfer function to integrate changes of external signals into analysis
Analysis framework (3)

- Problem
  - Clock events has to be modeled separately
- Introducing special ‘clock’ routine signalizing rising or falling events via special attribute
- Suppress uninteresting clock events for analyzed design, e.g. Leon completely triggered on rising edge
Analysis framework (4)

- For PAG
  - Analysis must not start with a recursive routine
  - Simply adding a calling procedure 'analysis start'
  - Thus, each analysis has an unique start point

- Feature
  - Existing PAG-iterator suffices for analyzing VHDL code
Questions so far?

I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions.