An Ecore Based Front-end for a VHDL Compiler

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Hao Wu *hao.wu@nuim.ie*
Computer Science Department

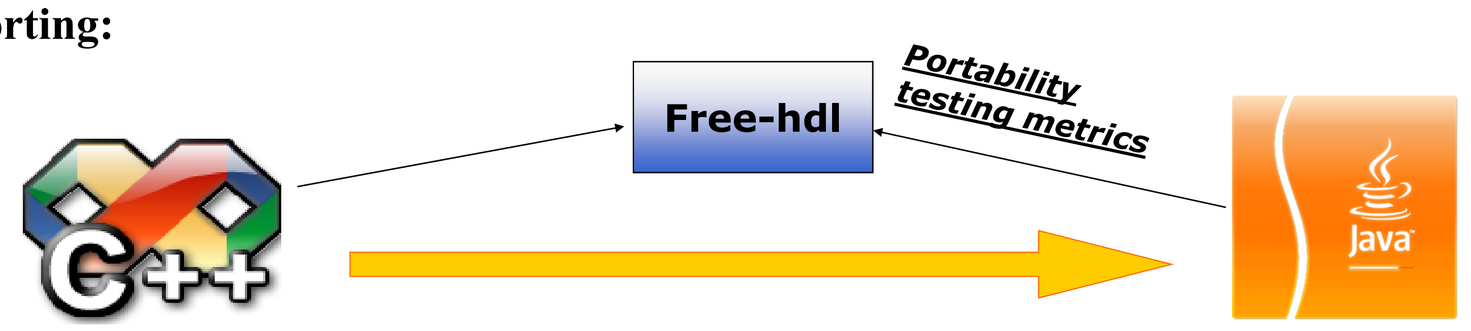
Challenges:

. No portability supported
. No testing metrics provided

Solution:

Solve these two problems by abstracting front-end of compiler designer (based on Ecore).

Porting:



Pick free-hdl as a case study for porting from C++ font-end to java side.

Free-hdl

AST Node (Scheme Definition)

```
(defnode IIR_PosInfo_TextFile (IIR_PosInfo)
((fire_string file_name)        ; XXX - IR_String
(int         line_number)))
```

JGen

Fire

C++ oriented Source code

Grammar File (.yy)

Bison

VAUL

Ecore Based VHDL

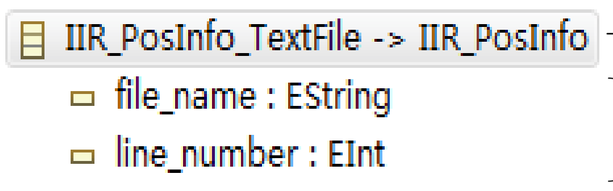
Annotated Java

```
/* generated class */
package AST;
import java.util.List;
/**
 * @model
 */
public interface IIR_PosInfo_TextFile extends
IIR_PosInfo {
        /* generated attributes */
      /**
        * @model
        */
      public String getFile_name();
      /**
        * @model
        */
      public int getLine_number();
};
```

Annotation tag

EMF

AST Meta-model



Tree node

Attributes /Fields

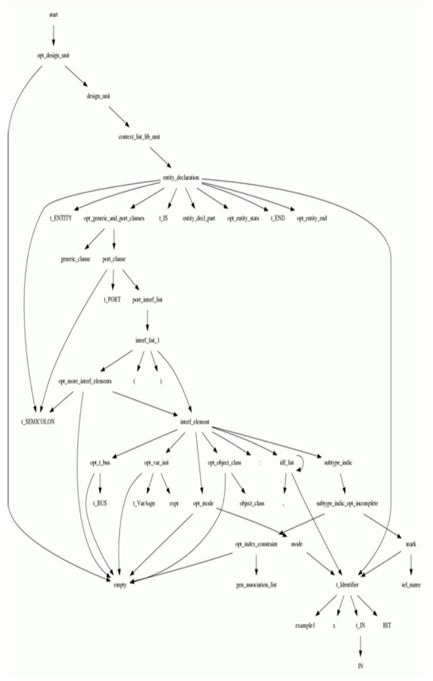
Why Ecore?

. Metamodel provided
. Auto implementation code generation
. Schmea based XML Serialization

Portability

Put together

. Understand free-hdl AST creation and join
. Jweever relieves a bit
. Works with smallest program
. Plot map for bison and javacc grammar
. Walk through both maps rule by rule
. Create and link AST nodes



JC4 (JavaCC with Code Coverage)

How to use it ?

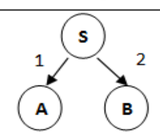
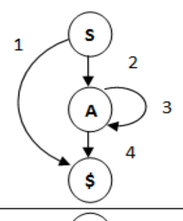
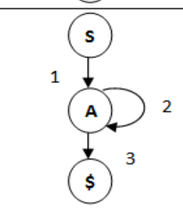
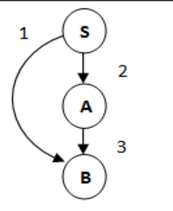
```
void numeric_literal()  :{}{
    LOOKAHEAD(physical_literal()) physical_literal()
  | abstract_literal()
}
```

Javacc –CODE_COVERAGE vhdl.jj

```
================================
numeric_literal
[2084,5 19313225 Choice]:Uncovered
[2084,5 19313325 Choice]:Covered
Numeric_literal:50.0%
```

Both local and global coverage are calculated each time after running the parser that is generated by JC4.

How does that work? 4 schemes are calculated.

Production Rule	Path diagram	Testing Strategy
S → A | B		Cover both path 1 and path 2.
S → A*		Cover path 1 , and make sure to go through path 2,3,4 at least once.
S → A+		Cover paths 1, 2, and 3, and make sure path 2 is executed at least once.
S → [A] B		Cover path 1 and paths 2 and 3.

Condition testing strategy is applied.

Any combination of these 4 scenarios can be calculated.

What's the impact?

Testing metrics are provided

works for any grammar

Software used

EMF: Eclipse Modeling Framework, relationship model based, code generation mechanism, etc.

Free-hdl: An open source VHDL compiler. Simulator, Debugger and waveform viewer supported.

JavaCC: Java Compiler Compiler. Java code based LL (k) parser generator.

Links:

http://www.eclipse.org/modeling/emf/

http://freehdl.seul.org/

https://javacc.dev.java.net/