

# Embedding Proof-Carrying Components into Isabelle

## *MASTER THESIS PROJECT PLAN*

Project period: September 1<sup>st</sup> 2008 – March 1<sup>st</sup> 2009  
Student name: Bruno Hauser  
Status: 9<sup>th</sup> Semester  
Email address: hauser.bruno@gmail.com  
Supervisor name: Martin Nordio

### 1. PROJECT DESCRIPTION

#### *Overview*

Proof-Carrying Components (PCC) are a form of trusted components, for which the guarantee of quality is perhaps the strongest one possible: a mathematical proof, machine-checkable, that the component satisfies specific properties, known as the contract for the component. These properties can be more or less extensive: they might characterize all that's interesting about the component's behavior, or just some specific aspects, such as absence of "null-pointer dereferencing" or other run-time failures.

Proof-Carrying Components can be automatically generated using *Proof-Transforming Compilers* [4, 5, 6]. PTCs are similar to certifying compilers in PCC, but take a source proof as input and produce the bytecode proof. An important property of Proof-Transforming Compilers is that they do not have to be trusted. If the compiler produces a wrong specification or a wrong proof for a component, the proof checker will reject the component.

To show the feasibility of Proof-Transforming Compilers, Nordio, Karahan, Guex and Hess [7,8,9] have implemented a PTC for a subset of Eiffel. The compiler takes a proof of an Eiffel program in XML format and produces the bytecode proof. However, the bytecode proof produced as result is not embedded in any theorem prover.

This project consists of embedding the Proof-Carrying Components into Isabelle. The components are automatically generated by the PTC. The compiler produces an AST of the component. The goal of this project is embedding the component into Isabelle.

#### *Scope of the work*

This project will develop a translator for Proof-Carrying Components to Isabelle. The task consists of embedding boolean expressions and CIL instruction into Isabelle. Boolean expressions are used to express both contracts and pre-postcondition of the proof. The formal grammar is the following:

**datatype** *EiffelContract* = **Requires** *boolExpr*  
| **ensures** *boolExpr*

**datatype** *boolExpr* = **Const** *bool*  
| **Neg** *boolExpr*  
| **And** *boolExpr boolExpr*  
| **Or** *boolExpr boolExpr*  
| **AndThen** *boolExpr boolExpr*  
| **OrElse** *boolExpr boolExpr*  
| **Xor** *boolExpr boolExpr*  
| **Impl** *boolExpr boolExpr*  
| **Eq** *expr expr*  
| **NotEq** *expr expr*  
| **Less** *expr expr*  
| **Greater** *expr expr*  
| **LessE** *expr expr*  
| **GreaterE** *expr expr*  
| **Type** *typeFunc*

**datatype** *typeFunc* = **ConformsTo** *typeExpr typeExpr*  
| **IsEqual** *typeExpr typeExpr*  
| **IsNotEqual** *typeExpr typeExpr*

**datatype** *typeExpr* = **EType** *EiffelType*  
| **Type** *expr*

**datatype** *expr* = **ConstInt** *int*  
| **RefVar** *varID*  
| **Attr** *objID attribID*  
| **CallR** *callRoutine*  
| **Create** *EiffelType routine argument*  
| **Old** *expr*  
| **Bool** *boolExpr*  
| **Void**

**datatype** *callRoutine* = **Call** *expr routine argument*

**datatype** *argument* = **Argument** *expr*

The CIL instructions that will be translated in this project are the following:

**datatype** *Instruction* = **ldloc** *VarName*  
| **ldc** *Value*  
| **stloc** *VarName*  
| **iladd**  
| **ilsub**  
| **ilmul**  
| **ildiv**  
| **ilrem**  
| **ilceq**  
| **ilcgt**  
| **ilclt**  
| **iland**  
| **ilor**  
| **ilneg**  
| **br** *Label*

```
| brtrue Label
| brfalse Label
| nop
| ret
| ldfld IFldld
| stfld IFldld
| newobj TName
| castc TName
| callvirt Type MethodID
| throw
```

### *Intended results*

The project will implement a translator that embeds proofs from the Eiffel PTC into Isabelle. This project will conclude the whole PCC architecture. Using the Eiffel PTC, proofs can be translated from Eiffel to CIL. Then, this project will embed the proof into Isabelle. Finally, the proof can be checked using the proof checker developed by Nordio, et al [4].

### *Optional Part*

An optional part of the project is to develop further the source proof - bytecode proof interfacing in EiffelStudio. Example: it would be nice if corresponding lines of the source- and the bytecode proof would both get highlighted in the GUI, since it can be quite tedious if this linking has to be searched manually.

## **2. BACKGROUND MATERIAL**

### *Reading list*

- Nordio, M. and Müller, P. and Meyer, B.: **Proof-Transforming Compilation of Eiffel Programs**
- M. Nordio and P. Müller and B. Meyer: **Formalizing Proof-Transforming Compilation of Eiffel Programs**
- P. Müller and M. Nordio: **Proof-Transforming Compilation of Programs with Abrupt Termination**. 6th Workshop on Specification and Verification of Component-Based Systems (SAVCBS), 2007.
- P. Müller and M. Nordio: **Proof-Transforming Compilation of Programs with Abrupt Termination**. *Technical Report 565, ETH Zurich*, 2007.

## **3. PROJECT MANAGEMENT**

### *Objectives and priorities*

The goal of the project is being able to produce Isabelle embeded proofs from the PTC. In a first step, simple proofs containing only Boolean and arithmetic operators and CIL instructions such as ld and st will be translated. In a second step, the translation will be extended incrementally.

### *Criteria for success*

The project will succeed if input such as the boolean expressions and instructions described in Section 1 can be embedded into Isabelle and successfully checked by compiler. If the already implemented PTC oughts to have a bug in the translation from Eiffel to CIL, the bug will be fixed so that proofs can be successfully checked by the proof checker. Axioms in the proof checker might be added to achieve this goal.

### *Quality management*

#### **Documentation**

- Master thesis report
- Examples of source proof – generated proof in Isabelle.

#### **Validation steps**

Continuous feedback from the supervisor will guide the development process.

## **4. PLAN WITH MILESTONES**

### *Project steps*

1. Setting up working environment – 3 week
  - Compiling EiffelStudio (from [paco.origo.ethz.ch](http://paco.origo.ethz.ch))
  - Installing Isabelle and run proof checker with different examples
  - Run Eiffel PTC and check results
  - Write own proof in Isabelle
2. Translate proof skeleton – 2 week
3. Translate simple Boolean expressions – 2 week
4. Translate simple instructions – 2 week
5. Extend Boolean expressions – 4 weeks
6. Extend instructions – 4 weeks
7. Optional: Interfacing source proof with bytecode proof - 4 weeks
8. Writing thesis report – 4 weeks

### *Deadline*

March 1<sup>st</sup> 2009

## *Tentative schedule*

| Task   | Duration | Weeks | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--|----------|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|
| Setting up working environment                         | 15 days  | 3     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Translate proof skeleton                               | 10 days  | 2     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Translate simple Boolean expressions                   | 10 days  | 2     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Translate simple instructions                          | 10 days  | 2     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Extend Boolean expressions                             | 20 days  | 4     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Extend instructions                                    | 20 days  | 4     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Optional: Interfacing source proof with bytecode proof | 20 days  | 4     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |
| Writing thesis report                                  | 20 days  | 4     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |   |   |

## 5. APPENDIX

Example of a source/bytecode program and its proof:

```

text{ *
  Source program:
  x=5
  y=x+1

  Bytecode program:
  require
    true
    {true}           00: ldc 5
    {s(0)=5}         01: stloc x
    {x=5}            02: ldc 1
    {x=5 \<and> s(0)=1} 03: ldloc x
    {x=5 \<and> s(1)=1 \<and> s(0)=x} 04: add
    {x=5 \<and> s(0)=1+x} 05: stloc y

  ensure
    x=5 \<and> y=6
* }

constdefs
  (* Variables *)
  x:: VarName
  [simp]: "x \<equiv> 0 "

  y:: VarName
  [simp]: "y \<equiv> 1"

constdefs
  PP00:: Prec
  [simp]: "PP00 \<equiv> (\<lambda> s \<sigma> Z. True)"

  LL00:: InstSpec
  [simp]: "LL00 \<equiv> (PP00, 00, (ldc (intV 5) ) )"

```

```

PP01:: Prec
[simp]:"PP01 \<equiv> (\<lambda> s \<sigma> Z. (
s\<lceil>0\<rfloor> = (intV 5) ) ) "

LL01:: InstSpec
[simp]:"LL01 \<equiv> (PP01, 01, (stloc x ) )"

PP02:: Prec
[simp]:"PP02 \<equiv> (\<lambda> s \<sigma> Z. ( (getV \<sigma> x)
= (intV 5) ) )"

LL02:: InstSpec
[simp]:"LL02 \<equiv> (PP02, 02, ldc (intV 1) )"

PP03:: Prec
[simp]:"PP03 \<equiv> (\<lambda> s \<sigma> Z. ( (getV \<sigma> x)
= (intV 5) \<and> (s\<lceil>0\<rfloor> = (intV 1)) ) ) "

LL03:: InstSpec
[simp]:"LL03 \<equiv> (PP03, 03, (ldloc x ) )"

PP04:: Prec
[simp]:"PP04 \<equiv> (\<lambda> s \<sigma> Z. ( (getV \<sigma> x)
= (intV 5) \<and> (s\<lceil>1\<rfloor> = (intV 1)) \<and> (
s\<lceil>0\<rfloor> = (getV \<sigma> x) ) ) ) "

LL04:: InstSpec
[simp]:"LL04 \<equiv> (PP04, 04, iladd)"

PP05:: Prec
[simp]:"PP05 \<equiv> (\<lambda> s \<sigma> Z. ( (getV \<sigma> x)
= (intV 5) \<and> ( s\<lceil>0\<rfloor> = ( addil (getV \<sigma>
x) (intV 1) ) ) ) )"

LL05:: InstSpec
[simp]:"LL05 \<equiv> (PP05, 05, (stloc y ) )"

constdefs
PreCond2:: Prec
[simp]:"PreCond2 \<equiv> (\<lambda> s \<sigma> Z. True)"

PostCond2:: Prec
[simp]:"PostCond2 \<equiv> (\<lambda> s \<sigma> Z. ( (getV
\<sigma> x) = (intV 5) \<and> (getV \<sigma> y) = (intV 6) ) ) "

PostCondE2:: Prec
[simp]:"PostCondE2 \<equiv> (\<lambda> s \<sigma> Z. False)"

constdefs
bodyFoo2:: CilProof
[simp]:"bodyFoo2 \<equiv> [LL00,LL01,LL02,LL03,LL04,LL05]"

et2:: ExcTable
[simp]:"et2 \<equiv> []"

foo2:: MethodDecl
[simp]:"foo2 \<equiv> (1, PreCond2, bodyFoo2, et2, (PostCond2,
PostCondE2)) "

```

```

my_classBody2:: ClassBody
[simp]:"my_classBody2 \<equiv> [foo2]"

consts
  my_c2:: TName

constdefs
  my_class2:: ClassDeclaration
  [simp]:"my_class2 \<equiv> ( refT my_c2, my_classBody2)"

  my_program2:: CilProgram
  [simp]:"my_program2 \<equiv> [my_class2, my_class]"

theorem small_step_VCGen2: "(VCGenInst LL03 et2 bodyFoo2 my_program2
PostCond2 PostCondE2 ) "
  apply(simp)
  apply(auto)
  apply(case_tac s, auto)
done

theorem safe_program2: "(VCGen my_program2) "
  apply(auto)
  apply(case_tac s, auto)
done

```

## 6. REFERENCES

- [1] Chair of Software Engineering: *Semester-/Diplomarbeiten*; Online at: <http://se.inf.ethz.ch/projects/index.html>.
- [2] Bertrand Meyer: *Object-Oriented Software Construction, 2nd edition*, Prentice Hall, 1997.
- [3] Isabelle: <http://www.cl.cam.ac.uk/research/hvg/Isabelle/>
- [4] Nordio, M. and Müller, P. and Meyer, B.: Proof-Transforming Compilation of Eiffel Programs. *TOOLS-EUROPE*, 2008.
- [5] M. Nordio and P. Müller and B. Meyer: Formalizing Proof-Transforming Compilation of Eiffel Programs. *Technical Report 587, ETH Zurich*, 2008.
- [6] P. Müller and M. Nordio: Proof-Transforming Compilation of Programs with Abrupt Termination. 6th Workshop on Specification and Verification of Component-Based Systems (SAVCBS), 2007.
- [7] Michel Guex. Implementing a Proof-Transforming Compiler from Eiffel to CIL. Semester Thesis, ETH Zurich. 2006.
- [8] Hasan Karahan. Proof-Transforming Compilation of Eiffel Contracts. Diploma Thesis, ETH Zurich. 2008.
- [9] Manuel Hess. Integrating Proof-Transforming Compilation into EiffelStudio. Master Thesis, ETH Zurich. 2008.