Course Unit Compilers

Masters in Informatics and Computing Engineering (MIEIC)

Department of Informatics Engineering

University of Porto/FEUP

2º Semester – 2020/2021

OO-based Low-Level Intermediate representation (OLLIR) Tool

Compiler backend tool

v0.2, April 6, 2021

Table of Contents

Abstract: This document describes the OO-based Low-Level Intermediate representation (OLLIR [3]) tool to be used in the compiler project. The OLLIR tool is able to parse OLLIR code and to represent such code in a Java structure that includes a control-flow graph (CFG) for each method in the input OLLIR. This tool includes the first stages of the compiler backend responsible to generate JVM code in the *jasmin* code format [2] from OLLIR input code.

---

1 Structure of the OLLIR Tool

Class ***Ollir*** is used as an example with a main method that calls the stages implemented by the OLLIR tool. This provides a main example (see Figure 1) that can be used in the main method of the backend of the compiler.

```
OllirParser parser;

        if (args.length == 0) {
            System.out.println("OLLIR 0.1:  Reading from standard input . . .");
            parser = new OllirParser(System.in);
        } else if (args.length == 1) {
            System.out.println("OLLIR 0.1:  Reading from file " + args[0] + " . . .");
            try {
                parser = new OllirParser(new java.io.FileInputStream(args[0]));
            } catch (java.io.FileNotFoundException e) {
```

```
                System.out.println("OLLIR 0.1:  File " + args[0] + " not found.");
                return;
            }
        } else {
            System.out.println("OLLIR 0.1:  Usage is one of:");
            System.out.println("         java org.specs.comp.ollir.Ollir < inputfile");
            System.out.println("OR");
            System.out.println("         java org.specs.comp.ollir.Ollir inputfile");
            return;
        }

        try {
            parser.ClassUnit();
            System.out.println("OLLIR 0.1:  OLLIR code parsed successfully.");
        } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.out.println("OLLIR 0.1:  Encountered errors during parse.");
        }

        // get Class
        ClassUnit myClass = parser.getMyClass();
        try {
            myClass.checkMethodLabels();
            myClass.buildCFGs();
            myClass.outputCFGs();
            myClass.buildVarTables();
            myClass.show();
        } catch (OllirErrorException e) {
            System.out.println(e.getMessage());
        }
```

Figure 1. Code example of a main method with the calls regarding the OLLIR parsing, building of CFGs, output of CFGs, build of Table with Variables for each method and a print to the console of the main information loaded form the input OLLIR.

Table 1, Table 2, Table 3 and Table 4 show the enum and classes used by the OLLIR tool.

Table 1. Enums used.

Enum Name	Possible Values	Brief description
AccessModifiers	PUBLIC, PRIVATE, PROTECTED, DEFAULT	The access modifiers that can be used.
NonAccessModifiers	FINAL, STATIC, ABSTRACT, NONE	The types of non-access modifiers.
CallType	invokevirtual, invokeinterface, invokespecial, invokestatic, NEW, arraylength, ldc	The kind of function call for CALL instructions
ElementType	INT32, BOOLEAN, ARRAYREF, OBJECTREF, CLASS, THIS, STRING, VOID	The type of each element used in an OLLIR instruction.
InstructionType	ASSIGN, CALL, GOTO, BRANCH, RETURN, PUTFIELD, GETFIELD, UNARYOPER, BINARYOPER, NOPER	The type of instruction.
OperandType	INT32, BOOLEAN, ARRAYREF, OBJECTREF, THIS, STRING, VOID	The type of each operand in an OLLIR instruction.
OperationType	ADD, SUB, MUL, DIV, SHR, SHL, SHRR, XOR, AND, OR, LTH, GTH, EQ, NEQ, LTE, GTE, ADDI32, SUBI32, MULI32, DIVI32, SHRI32, SHLI32, SHRRI32, XORI32, ANDI32, ORI32, LTHI32, GTHI32, EQI32, NEQI32, LTEI32, GTEI32, ANDB, ORB, NOTB, NOT	The type of operation.

Enum Name	Possible Values	Brief description
NodeType	INSTRUCTION, BEGIN, END	Type of the node in the control-flow-graph (CFG) of each method.
VarScope	LOCAL, PARAMETER, FIELD	The scope of the variables considering three scopes in OLLIR: field (class variable), method local variables, method parameters.

Table 2. Main classes representing the structure of the input OLLIR.

Class Name	Inherited classes	Brief description
ClassUnit	NA	Class representing all the information loaded from the OLLIR input
Field	NA	Class representing the fields of an OLLIR class.
Method	NA	Class represents each method in the input OLLIR class.
Node	NA	Class the represents a node in the CFG.
Operation	NA	Class representing the operation in an OLLIR instruction.

Table 3. Classes for the code in each input OLLIR method.

Class Name	Inherited classes	Brief description
Node		Class the represents a node in the CFG.
Node → Instruction	→ AssignInstruction	Class representing an OLLIR instruction. This extends the Node class, which is used for the CFG. Class representing assignments.
	→ UnaryOpInstruction	Class representing the OLLIR instructions with unary operations.
	→ UnaryOpInstruction → BinaryOpInstruction	Class representing all the instructions with 2 operands
	→ CallInstruction	Class representing the CALL instructions
	→ CondBranchInstruction	Class representing the OLLIR conditional instructions.
	→ GetFieldInstruction	lass representing the field instructions: getfield, getstatic.
	→ GetFieldInstruction → PutFieldInstruction	Class representing the field instructions: putfield, putstatic.
	→ GotoInstruction	Class representing the goto instructions. Those instructions include a label to which the flow must jump.
	→ ReturnInstruction	Class representing the OLLIR return instructions.
	→ SingleOpInstruction	Class representing the instructions with a single element.
	→ UnaryOperation	Class representing the OLLIR instructions with unary operations.

Table 4. Other classes used.

Class Name	Inherited classes	Brief description
Type		Class representing the type of variables, assignments, returns, literals.
	→ ClassType	The Type is a Class
Operation	NA	Class representing the operation in an OLLIR instruction.
Descriptor	NA	Class used to store the information regarding each variable of a method. It is used in the symbol table of the variables for each method
Element		Class representing the elements used in the OLLIR instructions.
	→ LiteralElement	Class represents the OLLIR elements that are literals.
	→ Operand	Class representing the elements of OLLIR instructions that are not literals.
	Operand → ArrayOperand	The class that represents each operand of type array.

2 Example

We include here an example of an OLLIR code (see Figure 2), the output of the show method (see Figure 3), `myClass.show();` in Figure 1, and of the CFG of method "sum" (see Figure 4) output to a dot[1] file by `myClass.outputCFGs();` in Figure 1.

```
myClass {

  .construct myClass().V {
    invokespecial(this, "<init>").V;
  }

  .method public sum(A.array.i32).i32 {
    sum.i32 :=.i32 0.i32;
    i.i32 :=.i32 0.i32;

    Loop:
      t1.i32 :=.i32 arraylength($1.A.array.i32).i32;
      if (i.i32 >=.i32 t1.i32) goto End;
      t2.i32 :=.i32 $1.A[i.i32].i32;
      sum.i32 :=.i32 sum.i32 +.i32 t2.i32;
      i.i32 :=.i32 i.i32 +.i32 1.i32;
      goto Loop;
    End:
      ret.i32 sum.i32;
  }
}
```

Figure 2. OLLIR example (in file "ex1.lir").

```
OLLIR 0.1:  Reading from file ex3.lir . . .
OLLIR 0.1:  OLLIR code parsed successfully.
** Name of the package: null
** Name of the class: myClass
        Access modifier: DEFAULT
        Static class: false
        Final class: false
-------------------------
```

[1] https://graphviz.org/

```
*** Name of the method: sum
        Construct method: false
        Access modifier: PUBLIC
        Static method: false
        Final method: false
        * Parameters:
                ARRAYREF
                ARRAYREF
        * Return: Type: ARRAYREF [0INT32
        * No. Instructions: 12

        * Instructions:
        ASSIGN Operand: t1 INT32 =       CALL Operand: A ARRAYREF
        ASSIGN Operand: C ARRAYREF =     CALL Operand: array ARRAYREF, Operand: t1 INT32
        ASSIGN Operand: i INT32 =        NOPER Literal: 0
        ASSIGN Operand: t1 INT32 =       CALL Operand: A ARRAYREF
        BRANCH Operand: i INT32 GTE Operand: t1 INT32 Label: End
        ASSIGN Operand: t2 INT32 =       NOPER Operand: A INT32
        ASSIGN Operand: t3 INT32 =       NOPER Operand: B INT32
        ASSIGN Operand: t4 INT32 =       BINARYOPER Operand: t2 INT32 ADD Operand: t3 INT32
        ASSIGN Operand: C INT32 =        NOPER Operand: t4 INT32
        ASSIGN Operand: i INT32 =        BINARYOPER Operand: i INT32 ADD Literal: 1
        GOTO Loop
        RETURN Operand: C ARRAYREF

        * Table of variables:
                Var name: t4 scope: LOCAL virtual register: 8
                Var name: A scope: PARAMETER virtual register: 1
                Var name: B scope: PARAMETER virtual register: 1
                Var name: C scope: LOCAL virtual register: 3
                Var name: array scope: LOCAL virtual register: 4
                Var name: i scope: LOCAL virtual register: 5
                Var name: t1 scope: LOCAL virtual register: 2
                Var name: t2 scope: LOCAL virtual register: 6
                Var name: t3 scope: LOCAL virtual register: 7
 ------------------------------------------------
```

Figure 3. Output by the method show for the OLLIR input given in Figure 2.

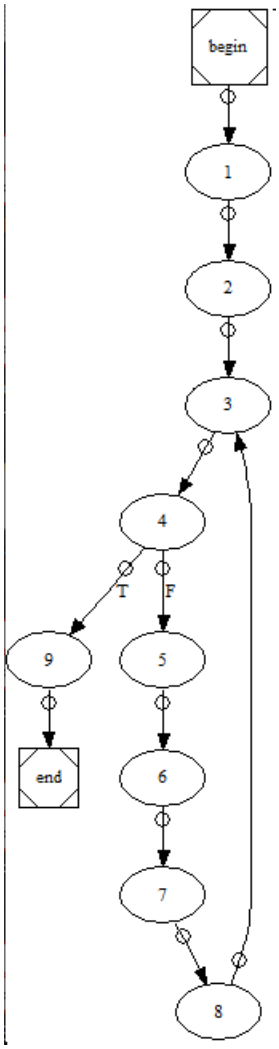

Figure 4. Output of the CFG for method sum of the OLLIR input given in Figure 2. Each node label represents the ordering of the OLLIR instructions in the input file.

References

[1] The Java Virtual Machine Specification, http://java.sun.com/docs/books/jvms/

[2] Jasmin Home Page, http://jasmin.sourceforge.net/

[3] Compiler MIEIC Course, "OO-based Low-Level Intermediate representation (OLLIR)," FEUP, University of Porto, v0.4, March 31, 2021.