Confined Types

with the

OVM Framework

by

Christian Grothoff

(C) 2001 OVM Consortium

Introduction

Infering confined types is a 3 step process:

- load classes (closed world)
- infer constraints
- solve constraint system

The OVM framework addresses the first two steps.

Confined Types and OVM Intersection

In your project your code will touch the following parts of OVM:

- load classes
  - Driver
  - Repository
  - OVMClass
  - NameSpace
  - ReferenceType
- infer constraints
  - Rewriter
  - InstructionSet
  - Instruction
  - Stack
  - StackElement
  - Policy

Classes in OVM

A class in OVM consists of representations on several levels of abstraction:

- Repository
  - Constants, Descriptors, Selectors, Bytecode
- OVMClass
  - Fields, Methods
- ReferenceType
  - Parent, Interfaces, Namespace (context)

The Repository

The Repository is basically a shared set of constants and is build from the constant pools of all loaded classes. It consists of:

- utf8 Strings
- Java Strings
- Descriptors
- Selectors

The classes in ovm.tools.bytecode parse a class and initialize the repository. This process is initiated by the OVMPreLoader.

OVMClass

OVMClass is a representation for the classfile. OVMClasses consist of

- access flags (class modifiers)
- names of the interfaces implemented
- name of the parent class
- OVMFields
  - OVMSelector
    - name of the field
    - type of the field
- OVMMethods
  - OVMSelector
    - name of the method
    - OVMDescriptor
      - arguments
      - return type
  - OVMCodeFragment

ReferenceType

ReferenceType represents a type in its context with:

- parent class
- interfaces
- child classes

ReferenceTypes come in differend brands:

- ClassType for Classes
- InterfaceType for Interfaces
- ArrayType for Arrays

Verification Framework

The Verification Framework sits in the ovm.tools.rewriter package.
The main classes are:

- Rewriter
  - runs the verification
- InstructionSet
  - contains all the instructions
- Stack
  - the abstract state of the StackFrame
- StackElement
  - the type of the local variables and the operand stack

Verification

The verification process works as follows:

- First, an initial state is created.
  - It represents the state of the VM on entry to the method.
- The initial state is put into the worklist.
- while the worklist is not empty:
  - take a state from the worklist
  - check that this state is new
  - execute the instruction that the PC of the state refers to
  - put all states that may follow after the execution into the worklist

State

A VM state is represented by Stack objects. A Stack object consists of:

- a program counter (offset into the bytecode of the method)
- the local variables
- the operand stack (including stack height)

StackElements

The operand stack and the local variables consist of StackElements.
The different StackElements are:

- Primitive (int, float, byte, char, short, boolean)
- WidePrimitive (long, double)
- TypedReference (Objects)
- TypedArrayReference (Arrays)
- ThisPointer (this)
- NullPointer (null)
- JumpTarget (immutable, see: jsr)

All StackElements are defined in the ovm.tools.rewriter.instructions package.

Driver

You are given a Driver class that provides the following services:

- parsing the command line
- loading the classes
- running the verification framework on all methods
- helper methods to query for:
  - return types of public methods
  - types of public fields
  - hierarchy information (extends Thread?)

Your Project

To complete your project, you will have to do the following:

- extend the verification framework to
  - derive anonymous methods
  - generate constraints for confined types
- extend the driver to
  - solve the constraint system
  - print the results

How to derive anonymous methods

- extend the Rewriter class (ConfRewriter)
- change the execute method of the relevant instructions
- keep track of ’bad’ uses of ’this’:
  - "if (se instanceof ThisPointer)"
- record non-anonymous methods (ConfRewriter)
- record calls to method with the ’this’ pointer

How to derive confined types

- implement 2 ConfinmentPolicy methods to
  - keep track of widening
  - keep track of method invocations on private classes
- check for public fields and methods
  - (code provided in Driver)
- check for violations in the hierarchy (Throwable)
  - (code provided in Driver)

check for invocations of non-anonymous methods on confined types

- solve the constraint system

The constraint system

A ::= non-anon(m,c)

T ::= non-conf(c)

C ::= A | C | A => A | A => T | T => T

OVM Classes you do NOT care about:

- MetaObject
- OVMPreLoader
- ClassTypeStub
- Type
- PrimitiveType
- ovm.orep.layout.*
- ovm.mm.*
- ovm.tools.bootstrap.*
- ovm.tools.inspector.*

If you wonder what any of these classes do,

you are on the wrong track!

Tips

- start by changing the Instructions in ConfRewriter
  - add datastructures to the ConfRewriter
- next edit ConfinementPolicy
- print the constraints you gather on the testcases provided
- write code to solve the constraint system
- if you are stuck with OVM related code, ask ovm@cs.purdue.edu

Thanks for your

at

presentation created with

http://www.mew.org/mgp/

May the source be with you.

The OVM Consortium
http://gecko.cs.purdue.edu/ovm/

- Christian Grothoff grothoff@cs.purdue.edu
- James Zheng Liang jianqiz@cs.purdue.edu