Annotating Java Class Files with Array Bounds Check and Null Pointer Check Information

Feng Qian (fqian@sable.mcgill.ca mailto:fqian@sable.mcgill.ca)

March 30, 2021

This note explains how to use Soot annotation options to add array bounds check and null pointer check attributes to a class file and how to use these attributes in a JIT or ahead-of-time compiler.

1 Array References and Object References

Java requires array bounds checks when accessing arrays, and null pointer checks when accessing objects. Array bounds checks are implemented at the virtual machine level by inserting comparison instructions before accessing an array element. Most of operating systems can raise a hardware exception when a bytecode accesses a null pointer, so the nullness check on an object reference is free at most of the time. However, some bytecodes, like the *invokespecial* and `athrow` instructions, do need explicit comparison instructions to detect null pointers. Both of these safety checking mechanisms do cause heavy runtime overhead.

Soot provides static analyses for detecting safe array and object accesses in a method. These analyses mark array and object reference bytecodes as either safe or unsafe. The results of these analyses are encoded into the class file as attributes, which can then be understood by an interpreter or JIT compiler. If a bytecode is marked as safe in its attribute, the associated comparison instructions can be eliminated. This can speed up the execution of Java applications. Our process of encoding class files with attributes is called *annotation*.

Soot can be used as a compiler framework to support any attributes you would like to define; they can then be encoded into the class file. The process of adding new analyses and attributes is documented in "Adding attributes to class files via Soot".

2 Annotation options in Soot

2.1 Description of new options

Soot has new command-line options `-annot-nullpointer` and `-annot-arraybounds` to enable the phases required to emit null pointer check and array bounds check annotations, respectively.

Soot has some phase options to configure the annotation process. These phase options only take effect when annotation is enabled. Note that the array bounds check analysis and null pointer check analysis constitute two different phases, but that the results are combined and stored in the same attribute in the class files.

The null pointer check analysis has the phase name "*jap.npc*". It has one phase option (aside from the default option *enabled*).

-p jap.npc only-array-ref

By default, all bytecodes that need null pointer checks are annotated with the analysis result. When this option is set to true, Soot will annotate only array reference bytecodes with null pointer check information; other bytecodes, such as `getfield` and `putfield`, will not be annotated.

Soot also has phase options for the array bounds check analysis. These options affect three levels of analyses: intraprocedural, class-level, and whole-program. The array bounds check analysis has the phase name "*jap.abc*". If the whole-program analysis is required, an extra phase "*wjap.ra*" for finding rectangular arrays is required. This phase can be also enabled with phase options.

By default, our array bounds check analysis is intraprocedual, since it only examines local variables. This is fast, but conservative. Other options can improve the analysis result; however, it will usually take longer to carry out the analysis, and some options assume that the application is single-threaded.

-p jap.abc with-cse
: The analysis will consider common subexpressions. For example, consider the situation where `r1` is assigned `a*b`; later, `r2` is assigned `a*b`, where both `a` and `b` have not been changed between the two statements. The analysis can conclude that `r2` has the same value as `r1`. Experiments show that this option can improve the result slightly.

-p jap.abc with-arrayref
: With this option enabled, array references can be considered as common subexpressions; however, we are more conservative when writing into an array, because array objects may be aliased. NOTE: We also assume that the application in a single-threaded program or in a synchronized block. That is, an array element may not be changed by other threads between two array references.

-p jap.abc with-fieldref
: The analysis treats field references (static and instance) as common subexpressions. The restrictions from the '`with-arrayref`' option also apply.

-p jap.abc with-classfield
: This option makes the analysis work on the class level. The algorithm analyzes 'final' or 'private' class fields first. It can recognize the fields that hold array objects with constant length. In an application using lots of array fields, this option can improve the analysis results dramatically.

-p jap.abc with-all
: A macro. Instead of typing a long string of phase options, this option will turn on all options of the phase "*jap.abc*".

-p jap.abc with-rectarray, -p wjap.ra with-wholeapp
: These two options are used together to make Soot run the whole-program analysis for rectangular array objects. This analysis is based on the call graph, and it usually takes a long time. If the application uses rectangular arrays, these options can improve the analysis result.

2.2 Examples

Annotate the benchmark in class file mode with both analyses.

```
java soot.Main -annot-nullpointer -annot-arraybounds spec.benchmarks._222_mpegaudio.Main
```

The options for rectangular array should be used in application mode. For example:

```
java soot.Main --app -annot-arraybounds -annot-arraybounds -p wjap.ra with-wholeapp
  -p jap.abc with-all spec.benchmarks._222_mpegaudio.Main
```

The following command only annotates the array reference bytecodes.

```
java soot.Main -annot-arraybounds -annot-arraybounds -jap.npc only-array-ref
  spec.benchmarks._222_mpegaudio.Main
```

3 Using attributes in the Virtual Machine

The array bounds check and null pointer check information is encoded in a single attribute in a class file. The attribute is called `ArrayNullCheckAttribute`. When a VM reads in the class file, it can use the attribute to avoid generating comparison instructions for the safe bounds and nullness checks.

All array reference bytecodes, such as *?aload, ?store* will be annotated with bounds check information. Bytecodes that need null pointer check are listed below:

```
?aload
?astore
getfield
putfield
invokevirtual
invokespecial
invokeinterface
arraylength
monitorenter
monitorexit
athrow
```

The attributes in the class file are organized as a table. If a method has been annotated, it will have an `ArrayNullCheckAttribute` attribute on its `Code_attribute`. The data structure is defined as:

```
array_null_check_attribute
{
    u2 attribute_name_index;
    u4 attribute_length;
    u3 attribute[attribute_length/3];
}
```

The attribute data consist of 3-byte entries. Each entry has the first two bytes indicating the PC of the bytecode it belongs to; the third byte is used to represent annotation information.

```
soot_attr_entry
{
    u2 PC;
    u1 value;
}
```

Entries are sorted by PC in ascending order when written into the class file. The right-most two bits of the '*value*' byte represent upper and lower bounds information. The third bit from right is used for nullness annotation. Other bits are not used and set to zero. The bit value '1' indicates the check is needed, and 0 represents a known-to-be-safe access. In general, only when both lower and upper bounds are safe can the check instructions be eliminated. However, sometimes this depends on the VM implementation.

```
0 0 0 0 0 N U L
    N : nullness check
    U : upper bounds check
    L : lower bounds check
```

For example, the attribute data should be interpreted as:

```
0 0 0 0 0 1 x x    // need null check
0 0 0 0 0 0 x x    // no null check
```

```
// x x represent array bound check.

0 0 0 0 0 0 0 0    // do not need null check or array bounds check
0 0 0 0 0 1 0 0    // need null check, but not array bounds check
```

Other information

The detailed annotation process is described in our technical report. The array bounds check analysis algorithm will show up in another technical report. There is a tutorial describing how to develop other annotation attributes using Soot.

Change log

- October 2, 2000: Initial version.