

**ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ**



Πτυχιακή Εργασία

**Στατική sharing ανάλυση για βελτιστοποίηση
δυναμικής ανίχνευσης data race**

<https://github.com/Bela-Kamilo/static-sharing-analysis-for-optimizing-dynamic-race-detection>

Κυπαρίσσης Αλέξανδρος CSD 4210

Επόπτης καθηγητής: Πολύβιος Πρατικάκης

...

Ηράκλειο Κρήτης 2025

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. Περίληψη	2
2. Εισαγωγή	
2.1 data race	
2.2 points-to ανάλυση	
3. Σχετική Δουλειά	
4. Αναλυτική περιγραφή	
4.1 Jimple IR	
4.2 Αφαίρεση θέσεων μνήμης	
4.3 Points-to Sets	
4.4 Ασφαλής αφαίρεση	
4.5 May	
4.6 Intra-procedural	
4.7 Πρόβλημα ικανοποίησης περιορισμών	
4.8 Κανόνες Παραγωγής	
4.9 Side Effects	
4.10 Choco-solver	
4.11 DaCapo Benchmarks	
5. Επίλογος	
6. Αναφορές	

1. Περίληψη

Δυναμικές, on-the-fly, αναλύσεις προγραμμάτων παρέχουν σημαντικά περισσότερη πληροφορία από ο,τι στατικές προσεγγίσεις με σημαντικά μεγαλύτερο κόστος απόδοσης. Σκοπός αυτής της εργασίας είναι η εύρεση προσβάσεων στο heap του JVM, οι οποίες είναι αδύνατο να προκαλέσουν data race, πριν την εκτέλεση ενός προγράμματος. Βάση αυτής της πληροφορίας, μια δυναμική ανίχνευση data race μπορεί να αγνοήσει με ασφάλεια συγκεκριμένες εντολές μειώνοντας τον χρόνο εκτέλεσής της. Παρουσιάζεται μια στατική ανάλυση, ανεξάρτητη της ροής του προγράμματος, της οποίας η βάση είναι μια points-to ανάλυση.

Abstract

Dynamic, on-the-fly, program analyses yield significantly more information than static approaches at the cost of significant performance cost. The aim of this work is finding thread safe JVM memory accesses, which are sure not to cause data races, before execution time. Using this information, a dynamic race detector can safely ignore certain instructions reducing its execution time. A static, context-insensitive, points-to based analysis is presented

2. Εισαγωγή

<κειμενο>

2.1 Data race

To specification της Java ορίζει το data race ως εξής :

<in quote format :>

When a program contains two conflicting accesses that are not ordered by a happens-before relationship, it is said to contain a *data race*.

Two accesses to (reads of or writes to) the same variable are said to be *conflicting* if at least one of the accesses is a write.¹

2.1 Points-to ανάλυση

Η points-to ανάλυση έχει σκοπο να ανακαλύψει ποιές μεταβλητές της στοίβας του προγράμματος ενδέχεται να αναφέρονται σε ποιά objects.

Κάθε τελεστής new σηματοδοτεί την δημιουργία ενός object.

Εντολές εκχώρησης και κλήσεις μεθόδων μεταβιβάζουν τα objects μεταξύ των μεταβλητών υποδεικνύοντας σε ποιά objects ενδέχεται να αναφέρονται.

Το αποτέλεσμα είναι μία αντιστοίχιση των μεταβλητών και συνόλων από objects.

3. Σχετική δουλειά

<thread sanitizer>

<Andersen's analysis>

4. Αναλυτική περιγραφή

Η είσοδος της ανάλυσης είναι Java bytecode, η επεξεργασία του οποίου γίνεται μέσω του SootUp framework.²

Η είσοδος μετατρέπεται σε μορφή ενδιάμεσης αναπαράστασης (Jimple IR).

<entry method κάθε run() >

Εξετάζεται κάθε εντολή που εν δυνάμει εκτελείται από αυτή την μέθοδο και όποια άλλη μέθοδο καλείται αναδρομικά.

Παράγεται μια αντιστοιχία μεταβλητών και συνόλων από objects στα οποία εν δυνάμει αναφέρονται και μια αντιστοίχιση μεθόδων και προσβάσεων μνήμης.

Οι προσβάσεις μνήμης είναι της μορφής : READ/WRITE object#.field .

<read/write effect sets/set sizes>

4.1 Jimple IR

Η Jimple είναι η ενδιάμεση αναπαράσταση του SootUp.

- stackless
- no nested structures

παράδειγμα hello world :

```
public class HelloWorld extends java.lang.Object
{
    public void <init>()
    {
        HelloWorld r0;
        r0 := @this: HelloWorld;
        specialinvoke r0.<java.lang.Object: void <init>()>();
        return;
    }

    public static void main(java.lang.String[])
    {
        java.lang.String[] r0;
        java.io.PrintStream r1;
        r0 := @parameter0: java.lang.String[];
        r1 = <java.lang.System: java.io.PrintStream out>;
        virtualinvoke r1.<java.io.PrintStream:
        void println(java.lang.String)>("Hello world!");
        return;
    }
}
```

Αναλυτικά η γραμματική της ³:

<i>stmt</i> \rightarrow <i>assignStmt</i> <i>identityStmt</i> <i>gotoStmt</i> <i>ifStmt</i> <i>invokeStmt</i> <i>switchStmt</i> <i>monitorStmt</i> <i>returnStmt</i> <i>throwStmt</i> <i>breakpointStmt</i> <i>nopStmt</i> ;
<i>assignStmt</i> \rightarrow <i>local</i> = <i>rvalue</i> ; <i>field</i> = <i>imm</i> ; <i>local</i> . <i>field</i> = <i>imm</i> ; <i>local</i> [<i>imm</i>] = <i>imm</i> ;
<i>identityStmt</i> \rightarrow <i>local</i> := @ <i>this</i> : <i>type</i> ; <i>local</i> := @ <i>parameter</i> <i>n</i> : <i>type</i> ; <i>local</i> := @ <i>exception</i> ;
<i>gotoStmt</i> \rightarrow goto <i>label</i> ;
<i>ifStmt</i> \rightarrow if <i>conditionExpr</i> goto <i>label</i> ;
<i>invokeStmt</i> \rightarrow invoke <i>invokeExpr</i> ;
<i>switchStmt</i> \rightarrow lookupswitch <i>imm</i> {case <i>value</i> ₁ : goto <i>label</i> ₁ ; ... case <i>value</i> _{<i>n</i>} : goto <i>label</i> _{<i>n</i>} ; default: goto <i>defaultLabel</i> }; tableswitch <i>imm</i> {case <i>low</i> : goto <i>lowLabel</i> ; ... case <i>high</i> : goto <i>highLabel</i> ; default: goto <i>defaultLabel</i> };
<i>monitorStmt</i> \rightarrow entermonitor <i>imm</i> ; exitmonitor <i>imm</i> ;
<i>returnStmt</i> \rightarrow return <i>imm</i> ; return;
<i>throwStmt</i> \rightarrow throw <i>imm</i> ;
<i>breakpointStmt</i> \rightarrow breakpoint;
<i>nopStmt</i> \rightarrow nop;

$imm \longrightarrow \text{local} \mid \text{constant}$
$conditionExpr \longrightarrow imm_1 \text{ condop } imm_2$ $condop \longrightarrow > \mid < \mid = \mid \neq \mid \leq \mid \geq$
$rvalue \longrightarrow concreteRef \mid imm \mid expr$ $concreteRef \longrightarrow \text{field} \mid$ $\text{local} . \text{field} \mid$ $\text{local} [imm]$
$invokeExpr \longrightarrow \text{specialinvoke local.m}(imm_1, \dots, imm_n) \mid$ $\text{interfaceinvoke local.m}(imm_1, \dots, imm_n) \mid$ $\text{virtualinvoke local.m}(imm_1, \dots, imm_n) \mid$ $\text{staticinvoke m}(imm_1, \dots, imm_n)$
$expr \longrightarrow imm_1 \text{ binop } imm_2 \mid$ $(\text{type}) \text{ imm} \mid$ $imm \text{ instanceof type} \mid$ $invokeExpr \mid$ $\text{new refType} \mid$ $\text{newarray (type) } [imm] \mid$ $\text{newmultiaarray (type) } [imm_1] \dots [imm_n] []^* \mid$ $\text{length } imm \mid$ $\text{neg } imm$
$binop \longrightarrow + \mid - \mid > \mid < \mid = \mid \neq \mid \leq \mid \geq \mid * \mid / \mid$ $<< \mid >> \mid <<< \mid \% \mid \text{rem} \mid \& \mid \mid \mid$ $\text{cmp} \mid \text{cmpg} \mid \text{cmpl}$

4.2 Αφαίρεση θέσεων μνήμης

Κάθε `new` σηματοδοτείται μοναδικά με έναν ακέραιο αριθμό αναπαριστώντας μια θέση μνήμης ενός object.

```
x= new1 A();    //object 1 created here  
y= new2 A();    //object 2 created here
```

Ένα `new` ενδεχομένως να δημιουργήσει περισσότερα από ένα objects.
Τα objects που δημιουργούνται από την ίδια εντολή αντιμετωπίζονται σαν ένα

```
while(...){  
    x= new3 A(); //object(s) 3 created here  
    ...  
}
```

Οι θέσεις μνήμης μας, αναπαριστώντας objects, έχουν πεδία.
Αναφερόμαστε σε αυτά ως `1.someField`

4.3 Points-to Sets

Η πληροφορία “η μεταβλητή `x` αναφέρεται στο object 1” αναπαριστάται με την αντιστοιχία $x \rightarrow \{1\}$.

Μια αντισοίχιση του παραπάνω παραδείγματος

```
x= new1 A();    //object 1 created here  
y= new2 A();    //object 2 created here
```

είναι η εξής :

```
x → {1}  
y → {2}
```

4.4 Ασφαλής αφαίρεση

<κειμενο>

4.5 May

Μία pointer ανάλυση χαρακτηρίζεται may ή must. Σκοπός της παρούσας ανάλυσης είναι οι αντιστοιχίες

```
someVariable → {1, 3}  
WRITES(someMethod) → {2.someField, 4.someField}
```

να εκφράζουν ότι η μεταβλητή someVariable ενδέχεται να αναφέρεται στο object 1 ή στο object 3 και η μέθοδος someMethod ενδέχεται να γράφει το πεδίο someField στην μνήμη όπου είναι δεσμευμένα τα objects 2 ή 4. Επομένως η ανάλυσή μας είναι may ανάλυση.

4.6 Intra-procedural

<κειμενο>

4.7 Πρόβλημα ικανοποίησης περιορισμών

<κειμενο>

4.8 Κανόνες Παραγωγής

<κειμενο>

4.9 Side Effects

<κειμενο>

4.10 Choco-solver

<κειμενο>

4.11 DaCapo Benchmarks

<κειμενο>

ΑΝΑΦΟΡΕΣ

- [1] Java Language Specification, Java SE 7 Edition
<https://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.4.5>
- [2] SootUp
<https://soot-oss.github.io/SootUp/latest/>
- [3] Soot : a java bytecode optimization framework, Raja Vallée-Rai master thesis