Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Intervallum-alapú absztrakt interpretációs algoritmus fejlesztése invariáns tulajdonságok ellenőrzésére

Bachelor's Thesis

Author
Dávid Román

Advisor
András Vörös
Ákos Hajdú

March 6, 2019

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Román Dávid*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2019. március 6.

Román Dávid
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

Chapter 1

Introduction

Todays softwares are made from millions of lines by hundreds or even thousands of programmers. According to Steve McConnell's book Code Complete on average there are 10-50 errors in 1000 lines of code. So it is inevitable that there will be a lot of mistakes during making these huge softwares. On the other hand we rely on these various parts of our lives, so if the program has bugs it causes different effects. If the outcome of this malfunction dangers great fortunes os human health or even lives than we say it is a safety critical system. We want to make sure that these systems are fault proof. Static analysis is a method to analyze the software without actually executing it, detecting possible vulnerable part of the source code. Some problems such as simple coding errors are easy to find, however we can detect other, more complicated vulnerabilities like possible zero division or other logical errors. However checking the whole software can be impossible within a reasonable time. In this case abstracting can simplify the problem, and make it possible to analyze certain behaviors of the software.

Static Analysis by Abstract Interpretation (SAAI) was introduced by Cousot in [2]. An easy to understand description is available at [1]. Able to analyze certain behaviors of the software, by making an abstraction which focuses on this behavior so it is much simpler than the whole software, but the required conditions can still be tested. There are plenty of abstraction methods such as sign or interval abstraction.

Chapter 2

Background

2.1 program representation

A program most commonly is represented by a source code. Example: an average counter function in c

```
int average(int a, int b){
int avg;
avg=(a+b)/2;
return avg;
}
```

There are many different type of code languages, making a static analyzer for all, would be hard and unnecessarily time-consuming.

2.2 Control Flow Automata(CFA)

CFA can describe the programs as graphs, where edges are annotated with program statements. The Theta framework [4] provides a representation of a CFA formalism.

A CFA is a directed graph with

- variables,
- locations, with dedicated initial, final and error locations,
- edges between locations, labeled with statements over the variables.

1. Assume: check if a condition is true for the variables
2. Assign: assign a concrete value to a variable
3. Havoc: assign a random value to a
4. Skip: no action

This simple C code translates to 5 location:

```
//Loc1
int a=1;
//Loc2
if(a!=1){
```

```
  //errorLOC
}
else {
//Loc3
  printf("%d", a);
}
//Loc4
```

The edges are the program statements. For example from Loc1 to Loc2 an assign statement which set a variable to 1, and from Loc2 to Loc3 is an assume statement (!(a!=1)).

Analysis is usually made for reachability of the error state.

2.3 Motivation

Why do we need abstraction?

2.4 Abstraction in general

2.5 Abstraction analysis algorithm for CFA

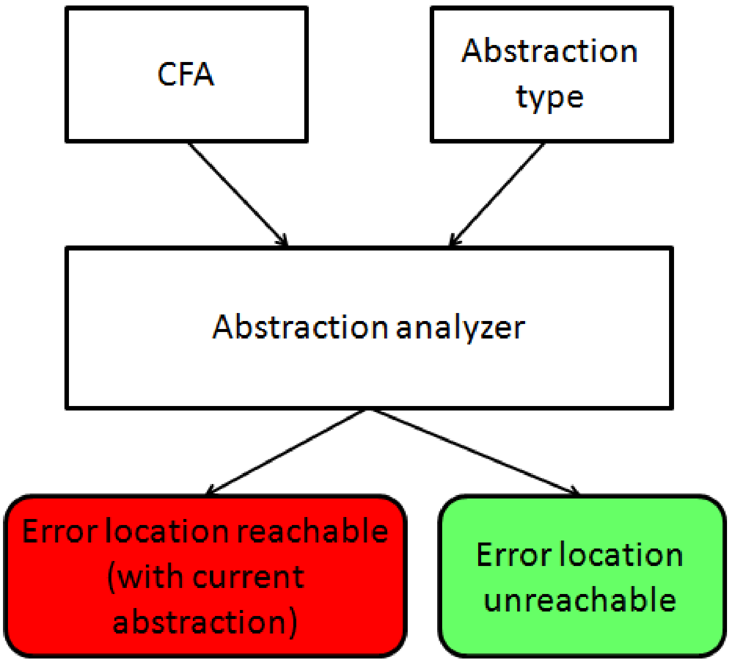

Figure 2.1: Abstraction analysis model

The CFA provides only one error state. This is not a problem since if there is more, then a simple abstraction can make one error state that contains all.

$\forall i\ S_P^i \in$ Error states $S_P^{\prime error} = \cup S_P^i$

Label: representation of the Program. (in section SAAI these were the states of the program S_P^i). Usually it will be a representation of the CFA's variables.

Every abstraction type need to have its own Label type. For example in case of sign abstraction a label type would be a (-), (+) or (-+) assigned to every variable (meaning: it can only be minus, it can only be positive, it can be both).

The possible trajectories are represented in the CFA by iterations of the CFA graph. So the problem of reaching the error states, in CFA means that there is no valid path from the initial location to the error location. Valid means that every edge in the path is a possible step.

For example a possible trajectory in the code from the previous section is Loc1, Loc2, Loc3, Loc4 (this is the only possible trajectory since Loc2 -> errorLoc edge is an impossible step (a is 1)).

The validation test on an edge is only possible if we put labels to the locations and from the label it is possible to decide that an edge is valid or not. Note: the edge can only be invalid if it has an assume statement For example if we use sign abstraction and on LocationA we have a label that $var = (+)$ than we can decide wether edge LocationA -> LocationB is valid or not. For example $var > 0$ is valid, but $var < 0$ is not.

Apply the statement: If an edge is valid, than we put a new label to the target location (the target of the edge) according to the statement on the edge. It has two different cases; if the target does not have a label, that means we have reached it for the first time. In this case from the source's label and the edge's statement, we need to be able, to decide the targets label. This is actually depending on the partition tactic (see Partition in previous chapter). If the target already has a label we need to take it into account. This is depending on the widening tactic (see Widening in previous chapter).

Discovered locations: All the locations that have been reached, and therefore have a label.

Discovered mapping: Every discovered location mapped with its label. When we apply a statement we modify the Discovered mapping (change the label for one location or add a completely new location).

Modifying edge: All the outgoing edges from those locations whose label have been modified from the previous Discovered mapping

Fixpoint: The point where the Discovered mapping can not be changed anymore. So there is no more modifying edges. Note: this is equivalent to: the previous Discovered mapping is the same as the current one (if we applied all the modifying edges from the previous discovered mapping).

Initial step: we put a label on the initial location (it is given in the CFA). Therefore every label type should have an initial label. It represents the program state, where we do not know anything, for example in sign abstraction every variable should be assigned to (+-), since both (+) and (-) can be true.

Iteration: Let there be a set of discovered locations $D(L)$, and a discovered mapping $M(Loc, Label)^n$ and the error location is $ELOC$. If $ELOC \in D(L)$ we can stop the iteration we reached the error location. Otherwise If $M(Loc, Label)^{n-1} == M(Loc, Label)^n$ we can stop there are no more modifying edges we reached a fixpoint therefore the error location is not reachable. If $M(Loc, Label)^{n-1}! = M(Loc, Label)^n$ than we get all the modifying edges from $M(Loc, Label)$ and apply all of the statements in the modifying edges. If one location is modified by more than one statement we add these labels together, for example in sign abstraction $var = (-)$ and $var = (+)$ are the two modifying statements, then we put $var = (-+)$. So labels should also support this operation.

If a location is reached and labeled, than its new label can only be less specific. For example in sign abstraction LocA has a $var = (+-)$ label than if there is a statement which assigns $var = (+)$ it can not narrow down LocA's label as in LocA (-) is already possible (in at least one trajectory).

Partitioning and widening can differ according to what type of abstraction are we using.

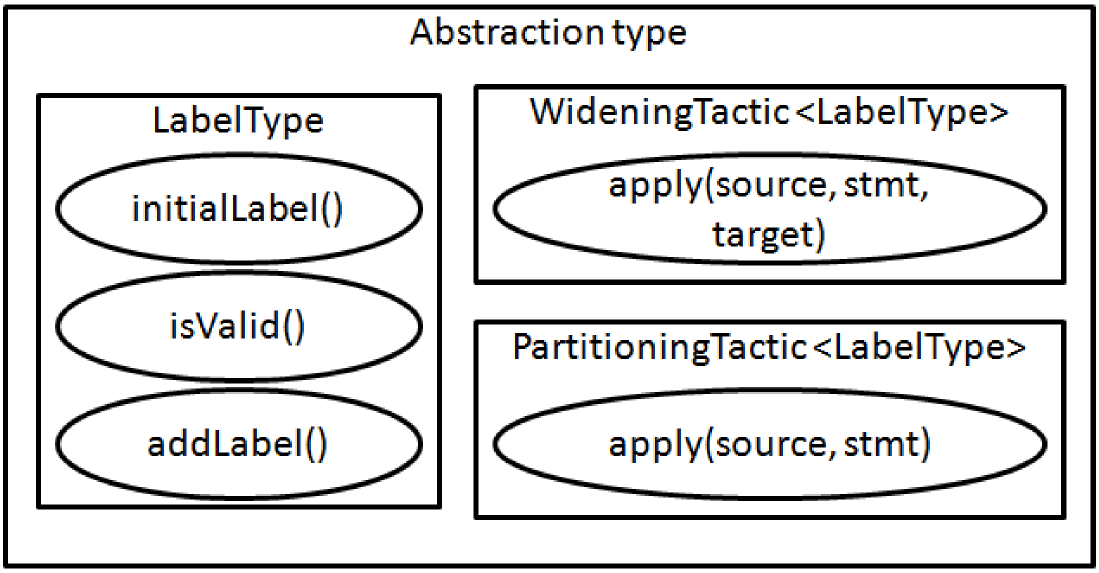

Figure 2.2: The structure of an Abstraction type

Chapter 3

Abstract Interpretation with Intervals

3.1 Library

Definition 1. Bound A Bound is a specified whole number or Infinite (can be positive or negative) $Bound \in \mathbb{Z} \vee Bound \in \{+\infty, -\infty\}$ ▪

Definition 2. max($Bound1$, $Bound2$)=

if $Bound1 == +\infty \vee Bound2 == +\infty$ then $+\infty$

else if $Bound1 == -\infty \vee Bound1 < Bound2$ then $Bound2$

else $Bound1$ ▪

Definition 3. min($Bound1$, $Bound2$)=

if $Bound1 == -\infty \vee Bound2 == -\infty$ then $-\infty$

else if $Bound1 == +\infty \vee Bound1 > Bound2$ then $Bound2$

else $Bound1$ ▪

Definition 4. $Bound + K \in \mathbb{Z})$=

if $Bound == +\infty \vee Bound == -\infty$ then $Bound1$

else ($Bound \in \mathbb{Z}$) then $Bound + K$ ▪

Definition 5. $|Bound|$=

if min($Bound$, 0) $== 0$ then $Bound$

else $Bound * (-1)$ ▪

Or

Definition 6. $|Bound|$=

if $Bound \in \mathbb{Z}$ then $|Bound|$ is the regular $absolutefunction$ in $\mathbb{Z}$

else $|Bound| = +\infty$ ▪

Definition 7. Interval An interval is specified with two Bounds: the lower Bound(LB) and the higher Bound(HB).

ex.: $(2; +\infty)$, $(3; 1)$ ▪

Definition 8. Interval is valid

if $\min(LB,\, HB) == LB \neq +\infty \wedge$ $\max(LB,\, HB) == HB \neq -\infty$

Note: empty interval (Ei) $\equiv$ invalid interval

Note2: intervals $(+\infty, +\infty)$, $(-\infty, -\infty)$ are also empty intervals

ex.: $(2; +\infty)$ and $(0; 0)$ is valid, but $(3; 1)$ is not valid $\equiv$ invalid ▪

Definition 9. section of two intervals $Interval1 \cap Interval1$=

if $Interval1, Interval2$ is valid then

LB=max($Interval1.LB$, $Interval2.LB$)

RB=min($Interval1.HB$, $Interval2.HB$)

else Ei

ex.: $(2, 8) \cap (1, 3) = (2, 3)$, $(2, 8) \cap Ei = Ei$ ▪

Definition 10. union of two intervals $Interval1 \cup Interval1$=

if $Interval1, Interval2$ is valid then

LB=min($Interval1.LB$, $Interval2.LB$)

RB=max($Interval1.HB$, $Interval2.HB$)

else if $Interval1$ is valid then $Interval1$

else if$Interval2$ is valid then $Interval2$

else Ei

ex.: $(2, 8) \cup (1, 3) = (1, 8)$, $(2, 8) \cup Ei = (2, 8)$ ▪

Definition 11. subtraction of two intervals (no partition) $Interval1 \setminus Interval2$=

if $Interval1$, $Interval2$ is valid then

if min($Interval1.LB$, $Interval2.LB$)==$Interval2.LB$ $\wedge$ max($Interval1.HB$, $Interval2.HB$)==$Interval2.HB$ then Ei

else

$Interval2.LB$=$Interval2.LB - 1$

$Interval2.HB$=$Interval2.HB + 1$

if min($Interval1.LB$, $Interval2.LB$)==$Interval1.LB$ $\wedge$ max($Interval1.HB$, $Interval2.HB$)==$Interval1.HB$ $\wedge$ $Interval2.LB \neq -\infty \wedge Interval2.HB \neq +\infty$ then $Interval1$

else if min($Interval1.LB$, $Interval2.LB$)==$Interval1.LB$ $\wedge$ max($Interval1.HB$, $Interval2.HB$)==$Interval1.HB$ $\wedge$ $Interval2.LB == -\infty \wedge Interval2.HB \neq +\infty$ then

LB=$Interval2.HB$

HB=$Interval1.HB$

else if min($Interval1.LB$, $Interval2.LB$)==$Interval1.LB$ $\wedge$ max($Interval1.HB$, $Interval2.HB$)==$Interval1.HB$ $\wedge$ $Interval2.LB \neq -\infty \wedge Interval2.HB == +\infty$ then

LB=$Interval1.LB$

HB=$Interval2.LB$

else if min($Interval1.LB$, $Interval2.HB$)==$Interval2.HB$ $\vee$ max($Interval1.HB$, $Interval2.LB$)==$Interval2.LB$ then $Interval1$

else if min($Interval1.LB$, $Interval2.LB$)==$Interval2.LB$ $\wedge$ max($Interval1.HB$, $Interval2.HB$)==$Interval1.HB$ then

LB=$Interval2.HB$

HB=$Interval1.HB$

else if min($Interval1.LB$, $Interval2.LB$)==$Interval1.LB$ $\wedge$ max($Interval1.HB$, $Interval2.HB$)==$Interval2.HB$ then

LB=$Interval1.LB$

HB=$Interval2.LB$

if $Interval2$ is invalid then $Interval1$

else Ei

for visual representation (see figure 4.1)

ex.: $(4,6) \setminus (1,8) = Ei$, $(2,8) \setminus (4,6) = (2,8)$, $(5,8) \setminus (1,4) = (5,8)$, $(1,4) \setminus (5,8) = (1,4)$, $(1,6) \setminus (3,8) = (1,2)$, $(3,8) \setminus (1,6) = (7,8)$, $Ei \setminus (1,6) = Ei$ ▪

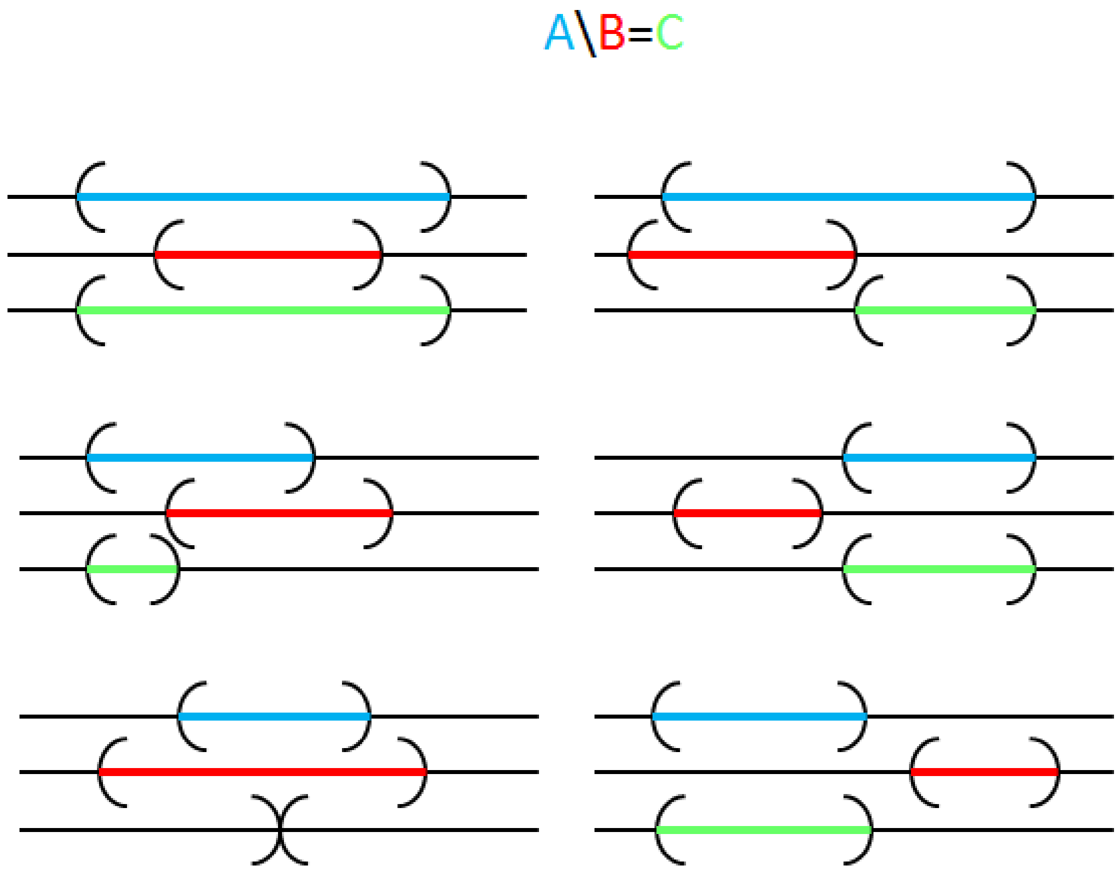

Figure 3.1: Interval $A \setminus$ Interval B possible outcomes

Definition 12. Initial interval (Ii)=

An Interval where

LB=$-\infty$

RB=$+\infty$

Initial interval $\equiv \Omega$ where Ω is the Full set of possible values

$(-\infty$,$+\infty)$

Definition 13. complementer of an interval$\overline{Interval}$=

$Ii \setminus Interval$

ex.: $\overline{(2, 8)} = (-\infty, +\infty)$ (Ii), $\overline{(-\infty, 8)} = (9, +\infty)$, $\overline{Ei} = (Ii)$

Definition 14. inside $k \in Interval$

let k be $\in \mathbb{Z}$ and interval Int be (k, k). then if $Int \cup Interval == Int$ we say k is inside $Interval$

ex.: $7 \in (2, 7)$

3.2 Interval representation

Definition 15. Interval representation is a label for interval abstraction. It maps an interval to every variable. The possible values are inside the given intervals for every variable.

Theorem 1. Let there be an Interval representation, where we map an Initial interval (Ii) to every variable. This interval representation is a good initial label for the abstraction analysis.

Definition 16. Let there be an Interval representation. If every variable is mapped with a valid interval then we say the Interval Representation is .

Theorem 2. Let there be two Interval representation $Ier1$ and $Ier2$ to the same program so it has the same variables. To every variable we map

$$Ier1.IntervalforVar \cup Ier2.IntervalforVar$$

This is a good *addLabel*() function for the abstract analysis.

proof: Let us say that this is not a good *addLabel*() function. That means that exists a variable value that is allowed by one of the Interval representation, but is not allowed in

$$Ier1.IntervalforVar \cup Ier2.IntervalforVar$$

however this contradicts with the definition of the union of two intervals. So this should be a good *addLabel*() function.

3.3 No Partitioning Tactic<Interval representation>

Definition 17. No Partitioning Tactic<Interval representation> is a Partitioning tactic for abstract analysis using Interval Representation as abstraction label.

Every Partitioning Tactic should define how to set the target's label from a source's label and an edge's statement.

Definition 18. Let there be an Interval $Interval$ and a statement $stmt$. Then we can apply $stmt$ to $Interval$ and it results in a new Interval. ($Interval.apply(stmt)$) ▪

Let there be a source's Interval representation $IerS$ and a statement $stmt$. No Partitioning Tactic<Interval representation> maps every variable to $IesS.IntervalforVar.apply(stmt)$.

Theorem 3. Let there be an Interval $Interval$ for variable var and a statement $stmt$. If $stmt$ has no effect on var then $Interval.apply(stmt) = Interval$ ▪

proof: every possible values in the source location will be possible in the target location, and if a value is not allowed in the source location it will not be allowed in the target location since nothing has changed regarding the variable

Theorem 4. Let there be an Interval $Interval$ for variable var and a Skip statement $skipstmt$.It has no effect on the variable. ▪

proof: it is directly comes from the definition of Skip statement. see CFA section

Theorem 5. Let there be an Interval $Interval$ for variable var and a Havoc statement $havstmt$. If $havstmt$ sets var then $Interval.apply(havstmt) = Ii$. Otherwise it has no effect on the variable. ▪

proof: if the statement sets the variable it means it can have any values. So we must represent the variable with $\Omega \equiv Ii$. Setting another variable does not have any effect on the current variable.

3.4 Applying Assign statement

Definition 19. Let there be an Assign statement $assignstmt$. Then the interval that represents all the possible values allowed by the assignment is $assignstmt.transform$. ▪

Theorem 6. Let there be an Interval $Interval$ for variable var and an Assign statement $assignstmt$. If $assignstmt$ sets var then $Interval.apply(assignstmt) = assignstmt.transform$. Otherwise it has no effect on the variable. ▪

proof: if the statement sets the variable it means it will have the value given by the assignment. So we must represent the variable with the Interval that represents any values allowed by the assignment which is $assignstmt.transform$. Setting another variable does not have any effect on the current variable.

An assignment is made of Expressions. Using the Theta Framework [4] and considering only the integer type expressions the possibilities are as follows

- Divide ($IntDivExpr$)
- Add ($IntAddExpr$)
- Subtract ($IntSubExpr$)

- Reference ($refExpr$) for variables.
- Literal ($IntLitExpr$) for literal expressions like 4 or 0

Theorem 7. Let k be a literal expression then the Interval that represents the possible values is (k, k) ∎

proof: The only possible value is k

Theorem 8. Let $Interval$ be a representation for possible values for variable var. Then a reference expression referencing var can be represented with $interval$. ∎

proof: The possible values are the same as it was before.

Theorem 9. Let $EXpr1$, $EXpr2$, ... ,$EXprn$ be expressions represented by $Interval1$, $Interval2$, ..., $Intervaln$. then the Interval representation for $Add(EXpr1, EXpr2, ..., EXprn)$ is

if $Interval1.LB, Interval2.Lb, ..., Intervaln.LB == -\infty$ then $result.LB = -\infty$ else $result.LB = \sum Intervali.LB$

if $Interval1.HB, Interval2.Hb, ..., Intervaln.HB == +\infty$ then $result.LB = +\infty$ else $result.LB = \sum Intervali.HB$ ∎

proof: the highest possible value is that every expression has the highest value. If all of them is finite then the biggest value is the sum of these, otherwise it is infinite. The lowest possible value is similar.

Theorem 10. Let $EXpr1$, $EXpr2$ be expressions represented by $Interval1$, $Interval2$. Then the Interval representation for $Sub(EXpr1, EXpr2)$ is $result$ where:

$result.LB = -\infty$ //initially

$result.HB = +\infty$ //initially

if $Interval1.HB \neq +\infty \wedge Interval2.LB \neq -\infty$ then

$result.HB = Interval1.HB - Interval2.LB$ $(Interval2.LB, Interval1.HB \in \mathbb{Z})$

if $Interval1.LB \neq -\infty \wedge Interval2.HB \neq \infty$ then

$result.LB = Interval1.LB - Interval2.HB$ $(Interval1.LB, Interval2.HB \in \mathbb{Z})$

proof: Starting as every values will be possible. We can decide the maximum value by subtracting the lowest possible value from the highest possible value. Let the values be a and b where we want to calculate $a - b$ and $a, b \in \mathbb{Z}$ then $a - b > c - b$ if $a > c$ and $a - b > a - c$ if $b < c$. The lowest possible value is similar.

Theorem 11. If we put more possible values to the assignment result we do not lose any possible values ∎

proof: If a value is possible and we put other possibilities as well, it is trivial that the original value will still be possible.

Of course we might have some values that is incorrect, however as said in the SAAI chapter it is tolerable to say that something is reachable even though it is not.

Theorem 12. Let $EXpr1$, $EXpr2$ be expressions represented by $Interval1$, $Interval2$. Then the Interval representation for $Div(EXpr1, EXpr2)$ is

if $Interval1.LB \neq -\infty \wedge Interval1.HB \neq +\infty$ then

$A =$max($|Interval1.LB|$, $|Interval1.HB|$)

if(0 is inside $Interval2$) then $B = 1$

else $B =$min($|Interval1.LB|$, $|Interval1.HB|$)

$Div(EXpr1, EXpr2).HB = A/B$

$Div(EXpr1, EXpr2).LB = -A/B$

else

$Div(EXpr1, EXpr2) = Ii$

proof: We simplify the problem by omitting the signs of the Bounds. This helps in the problem of sign changes (like (+)/(-)=(-)). We do not lose any possible values since we search for the highest possible absolute value and the interval will be $(-highest, highest)$. Now if $Interval1$ biggest absolute value is infinite then the result will be infinite as well. $(\infty/a = \infty)$ If it is finite then let as consider a and b where $a, b \in \mathbb{Z}^+$.Then $a/b > c/b$ if $a > c$ and $a/b > a/c$ if $b < c$. So in $Interval1$ we search for the highest possible absolute value in $Interval2$ for the lowest possible value. If *interval*2 is just positive or negative then the the lowest absolute value is on the bound, otherwise it contains 0. Zero division is not allowed, however our abstraction sometimes put 0 into the possibilities even though it is not possible. (For instance after division we always put 0 in the possible values, however it is only possible if the dividend is zero) So if we omit the 0 value then the next smallest absolute value is 1.

Theorem 13. Let $EXpr1$, $EXpr2$, ... ,$EXprn$ be expressions represented by $Interval1$, $Interval2$, ..., $Intervaln$. then the Interval representation for $Mul(EXpr1, EXpr2, ..., EXprn)$ is

if $Interval1.LB, Interval2.Lb, ..., Intervaln.LB == -\infty \vee +\infty$ then

$Mul(EXpr1, EXpr2, ..., EXprn) = Ii$

else

$Mul(EXpr1, EXpr2, ..., EXprn) = \Pi\max(|Intervali.LB|, |Intervali.HB|)$ ▪

proof: We simplify the problem by omitting the signs of the Bounds. This helps in the problem of sign changes (like (+)/(-)=(-)). We do not lose any possible values, because we search for the highest possible absolute value and the interval will be $(-highest, highest)$, so we only add more values. The biggest possible absolute value is when every multiplier has its highest possible value. If any multiplier is ∞ then the result is of course Ii

3.5 Applying Assume statement

Definition 20. Let there be an Assume statement *assumestmt* and a variable *var*. Then the interval that represents the values where the assumption is feasible for *var* is *assumestmt.transform*. ▪

Theorem 14. Let there be an Interval $Interval$ for variable var and an Assume statement $assumestmt$. If $assumestmt$ has a condition to var then $Interval.apply(assumestmt) = assumestmt.transform$. Otherwise it has no effect on the variable. ▪

proof: If the statement has condition for the variable it means it will have the value allowed by the assumption. So we must represent the variable with the Interval that represents any possible values in the assumption which is $assumestmt.transform$.

Theorem 15. An Assume statement can only narrow down the possible values. ▪

proof: If a value is not allowed in the source location, then it will not be possible in the target location since we, do not change the variable

The consequence of this theorem is this next theorem:

Theorem 16. Let there be an Interval $Interval$ for variable var and an Assume statement $assumestmt$. We do not lose any possible values if $Interval.apply(assumestmt) = Interval$ ▪

Definition 21. A *condition* is an interval, which represents the possible values for a variable (can be calculated from any Expression used in the assignment). ▪

Definition 22. We say an assumption is trivial for a variable var if it is in a form of: $var\{==, \neq, \geq, >, \leq, <\}condition$ where *condition* is an interval (it can be calculated from any Expression used in the assignment, but shall not depend on var) ▪

Definition 23. Let variable var be represented by $interval$. An incorrect value is inside $interval$, but var is not allowed to have this value. ▪

Definition 24. An incorrect condition is a *condition*, which is an interval that may have incorrect values. ▪

Theorem 17. A trivial condition is then and only then incorrect, if the *condition* is calculated using reference, multiply or divide expression ▪

proof: then:

Reference expression can be represented by an interval for a variable, which allows incorrect values.

Multiply, and divide expression both uses over exaggeration for the possible values.

only then: all the other possible assignments allow only the possible values.

Definition 25. We say an Assume statement is applicable for a variable if the statement consists of AND, OR or NOT functions of a trivial assumption for the variable. It is not applicable though if the Assume statement has incorrect condition. ▪

Let there be an Interval $Interval$ for variable var and an Assume statement $assumestmt$. if $assumestmt$ is not applicable $assumestmt.transform = Interval$. We can do this because of Theorem 17

Theorem 18. Let there be a trivial assumption for variable var represented by $IntervalVar$ where the assumption's *condition* is correct and represented by $IntervalCondition$

then

$var \geq condition = (\max(IntervalCondition.LB,\ IntervalVar.LB), IntervalVar.HB)$

ex. $(3,7) \geq (1,5) = (3,7)$, $(3,7) \geq (5,5) = (5,7)$

$var \leq condition = (IntervalVar.LB,\ \min(IntervalCondition.HB,\ IntervalVar.HB))$

ex. $(3,7) \leq (1,5) = (3,5)$, $(3,7) \leq (1,1) = (3,1) \equiv Ei$

$var > condition = (\max(IntervalCondition.LB + 1,\ IntervalVar.LB), IntervalVar.HB)$

ex. $(3,7) > (3,5) = (4,7)$, $(3,7) > (2,5) = (3,7)$

$var < condition = (IntervalVar.LB,\ \min(IntervalCondition.HB - 1,\ IntervalVar.HB))$

ex. $(3,7) < (1,5) = (3,4)$, $(3,7) < (3,3) = (3,2) \equiv Ei$

$var \neq condition = var \setminus condition$

ex. $(3,7) \neq (1,5) = (6,7)$, $(6,7) \neq (1,5) = (6,7)$

$var == condition = var \cup condition$

ex. $(3,7) == (1,5) = (1,5)$, $(6,7) == (1,5) = Ei$

Note: $\neq$ is the only one that can make incorrect possible values. For example $(3,7) \neq (4,5) = (3,7)$ however 4 and 5 are incorrect. Still we only allow more possibilities.

proof: $<, >$: Let $var > condition$ be $(a,b) > (c,d)$ and valid then $var > condition = (max(a, c+1), b)$

if $a > c+1$ then

$\forall i \in (a,b), \exists j \in (c,d)$ where $i > j$ so it is true and

$\forall i \notin (a, +\infty), \nexists j \in (a,b)$ where $i > j$

$\forall i \notin (-\infty, b), \nexists i < b$ no incorrect possible value is added

else $\forall i \in (c+1, b), \exists j \in (c,d)$ where $i > j$ so it is true and

$\forall i \notin (c+1, +\infty), \nexists j \in (c,b)$ where $i > j$

$\forall i \notin (-\infty, b), \nexists i < b$ no incorrect possible value is added

The other equation can be similarly proved.

Definition 26. If an Assume statement consists of AND functions of trivial assumption and the trivial assumptions result intervals are $Interval1, Interval2, ..., Intervaln$, then $assumestmt.transform = Interval1 \cap Interval2 \cap ... \cap Intervaln$ ▪

Theorem 19. The previously defined result interval is correct (has all the possible values) ▪

proof: The assumption is true only if every operant is true. An operant is true if the value is inside the interval that represents the operand so the section of these intervals is the possible values. Let us say there is a value where the result should be true, but it is not in the section. Then it is only not in the section, because one interval does not allow it. That operant results in false to this value thus the overall result will be false. So the value must be in the section.

Note: The operants does allow all the possible values (only $\neq$ can allow incorrectly values, but still allows the correct ones)

Definition 27. If an Assume statement consists of OR functions of trivial assumption and the trivial assumptions result intervals are $Interval1, Interval2, ..., Intervaln$, then $assumestmt.transform = Interval1 \cup Interval2 \cup ... \cup Intervaln$ ▪

Theorem 20. The previously defined result interval is correct (has all the possible values) ▪

proof: The assumption is false only if every operant is false. An operant is false if the value is not inside the interval that represents the operand. Let us say there is a value where the result should be true, but it is not in the union. It is only possible if the value is not inside of any operants interval. So all operants results in false for the value thus the overall result will be false. So the value must be in the union.

Note: The operants does allow all the possible values (only $\neq$ can allow incorrectly values, but still allows the correct ones)

Definition 28. If an Assume statement consists of a NOT function and is similar $Not(\mathit{condition})$ and the *condition* was calculated with allowing no incorrect values and $IntervalCondition$ is the interval representation for *condition* and we search for the new interval of a variable represented previously by $IntervalVar$ then $Not(\mathit{condition}) = IntervalVar \setminus \mathit{condition}$ ▪

Theorem 21. The previously defined result interval is correct (has all the possible values) ▪

proof: The assumption is true only if the condition is false.The Condition is false if the value is not inside the interval that represents the condition. Let us say there is a value where the result should be true, but it is not in $IntervalVar \setminus \mathit{condition}$. It is only possible if the value is inside of *condition*. Note: it must be inside $IntervalVar$. So the condition results in true for the value thus the overall result will be false. Note: if the *condition* interval has values that should be resulted in false, than these values overall result should be true, therefore *condition* should not allow incorrect values. So the value must be in $Ii \setminus \mathit{condition}$.

Note: This can also allow incorrect values, but at least allows all the possible ones.

Theorem 22. if we allow *condition* to be calculated by not only the assignment statements (ADD, MUL, etc). but conditional statements as well ($\geq$, AND, etc).Then *condition* is then and only then correct, when *condition* is calculated using no reference, multiply, divide, $\neq$ or NOT expression ▪

proof: We already discussed reference, multiply and divide. $\neq$ or NOT is similar to it since these can allow incorrect values.

3.6 Possible partitions

In some cases we allowed incorrect values for an interval. A trivial example is $a \neq 0$. in this case we set the interval for a to Ii thus having 0 as an incorrect value. One partitioning tactic could be to cat the result intervals to more pieces so we can have gaps as well.

For the previous example we can represent the possible values by two intervals $(-\infty, -1)$ and $(1, \infty)$.

This would make it possible to make $\neq$ a usable function in correct *condition*s.

In this case the label for a location could be mapping every variable to a set of intervals (not just one interval)

3.7 No Widening Tactic<Interval representation>

Definition 29. No Widening Tactic<Interval representation> is a Widening tactic for abstract analysis using Interval Representation as abstraction label. ▪

Every Widening Tactic should define how to set the target's label from a source's label, an edge's statement and the target's previous label.

Theorem 23. Let the target's new Interval be $IntervalNew$ and the previous be $IntervalOld$, then $\forall i \in IntervalOld$ it is true that $i \in IntervalNew$. ▪

proof: "If a location is reached and labeled, than its new label can only be less specific." see previous chapter

Let there be a source's Interval representation $IerS$, targets previous interval representation $IerT$ and a statement $stmt$. No Widening Tactic<Interval representation> maps every variable to $IerT \cup IesS.IntervalforVar.apply(stmt)$.

Theorem 24. The above mentioned tactic results in no loss of possible values ▪

proof: Let us say value a should be possible in the target location, but $a \notin IerT \cup IesS.IntervalforVar.apply(stmt)$. This means $a \notin IesS.IntervalforVar.apply(stmt) \wedge a \notin Ier$. $IesS.IntervalforVar.apply(stmt)$ allows all the possible new values (see previous sections). So a can not be a new value so it has to be an old one, but we said $a \notin Ier$. So every possible value is still possible

3.8 Other possible Widening Tactic

Let $interval = (0, 0)$ be the representation for a variable in a certain location. Assume there is an edge, which increments the variable by one. After applying the statement of this edge the new interval will be $interval = (0, 1)$. Assuming that this edge' source always changing (for example the source location is the same as the target) then $interval.HB$ can increase infinitely.

One possible solution for this problem is a widening tactic which actually has some widening approach.

Let there be a location, which is already labeled, with an interval representation. Let $intervalOld$ be an interval mapped for variable a. After the label is "refreshed" (this location was the target in a modifying edge.) let $intervalNew$ be an interval mapped for variable a. If $\exists i$ where $i \in intervalNew \wedge i \notin intervalOld$ We can say that the label for this location "wider".

To eliminate the possibility to widen the label infinitely, after one widening we change the label to prevent further widening. In the interval it can be for example the previously mentioned $(0, 0), (0, 1), (0, 2), ...$ can be prevented by instead of mapping tha variable with $(0, 1)$ we immediately map it with $(0, +\infty)$.

To prevent allowing too many incorrect values, we have to make it possible, to narrow the previously widened label.

For instance in the previous example, there can be an assumption that the variable must be smaller than 7. In this case we can narrow the widened interval $(0, +\infty)$ to $(0, 7)$.

3.9 Loss of information during interval abstraction

Abstraction always means that some information will inevitably get lost. Using Interval representation as labels for the location also has information losses.

During No Partition tactic we allowed that the intervals can represent incorrect values as long as it represents all the correct (possible) values. This already is some lost information.

Another even bigger concern that we lose some major connection between variables. Since every variable is maintained independently from the others. Let there be two variables $var1, var2$. There could be a correlation such that in a location $var1 = 1$ if $var2 = 1$ and $var1 = 0$ otherwise. The above described interval representation would label this: $var1 \in (0, 1) var2 \in (-\infty, +\infty)$. So we lost the information that $var1$ is only 1 if $var2 = 1$. Fortunately this problem only widens the possible values.

3.10 interval abstraction with color classes

3.11 Detailed example run on a CFA

Chapter 4

Implementation

4.1 Architecture

4.2 Abstraction Tool Usage

Chapter 5

Evaluation

5.1 Reachability analysis on real life examples

5.2 Conclusion

References

[1] Patrick Cousot: An informal overview of abstract interpretation

[2] Cousot, P. and R. Cousot, Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points

[3] Eric Goubaulta, St´ephane Le Rouxb, Jeremy Lecontec, Leo Libertib, and Fabrizio Marinellid: Static Analysis by Abstract Interpretation: A Mathematical Programming Approach

[4] Theta framework