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

April 18, 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. április 18.

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

Our every day life is becoming more and more affected by softwares. When we travel our cars, planes and trains are all operating through thousands or even million lines of codes. Of course nowadays human supervision is still required, however their control over the machines decreases year by year. Softwares are not only taking control in our traveling, but other various parts of our lives, such as in health care, education, agriculture and the list is growing. The result of this take over is that our life becomes easier as softwares can solve problems more effectively.

However, these softwares are made by programmers who are humans as well. 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, which can consist of a couple million lines of code written by thousands of programmers.

So it is plausible that there will be bugs in our system and they are causing different problems. If the outcome of the malfunctioning software dangers great fortunes, human health or even lives than we say it is a safety critical system. It is important to make sure that these systems are error free.

There are plenty of methods for verifying our software. Static analysis is one of them. It checks the code without actually executing it, detecting possible vulnerable parts. 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.

Formal verification is a mathematical method which can not only detect possible errors, but it can prove the correctness of the program. Abstract interpretation is a formal verification method that provides efficient abstraction and iteration strategy for defining invariants in the system.

Interval abstraction is an efficient abstraction method that represents the possible program states with intervals.

In our work we develop library for interval abstraction that can be used by any abstract interpretation algorithm. We also implement some of the possible algorithms. We try to make it so it can be extended later with other abstraction types, and other algorithms.

In order to be able to verify and evaluate our work we implement our solution in Theta, an open source verification framework (developed in our university). We can compare the performance of the different algorithms on multiple types of programs, including industrial programmable logic controller (PLC) codes from CERN, and several types of programs from the Competition on Software Verification (SV-Comp).

Chapter 2

Background

2.1 program representation

A program most commonly is represented by a source code. Example: Euclid's algorithm in c

```
int main(string args[])
{
  int arg1, arg2; //search for the highest common divisor of the 2 argument
  arg1 = args[0];
  arg2 = args[1];
  if(arg1>1000000 || arg1<=0 || arg2>1000000 || arg2<=0)
  {
    exit(1); //if the arguments are not supported we exit
  }
  for(arg1!=arg2)
  {
    if(arg1>arg2){
      arg1=arg1-arg2;
    }
    else{
      arg2=arg2-arg1;
    }
  }
  assertTrue(arg1==arg2);
  return arg1;
}
```

There are many different type of code languages, making an abstract interpretation for all, would be hard and unnecessarily time-consuming. we need a common representation for programs on which we only need to make one abstract analyzer.

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.

Statements can be:

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

The code in the 2.1 section translates to the CFA in figure 2.1

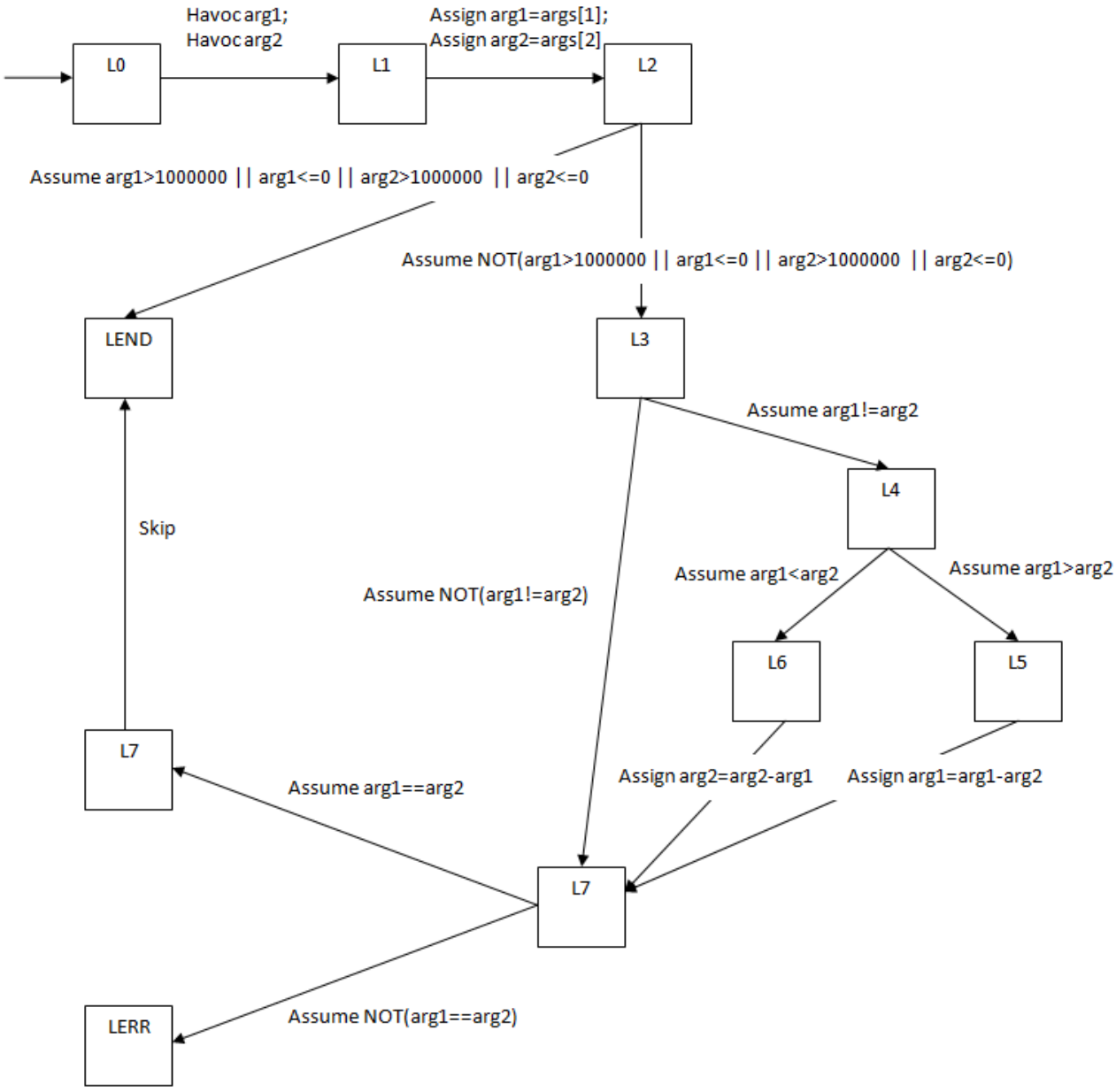

Figure 2.1: CFA representation of Euclid's algorithm

Analysis is usually made for reachability of the error state. As seen in the example, assertions in the code can be used to create error states in the cfa. So in the end of the

Euclid's algorithm if $arg1 == arg2$ assertion fails we go to an error state. if there are more than one assertions in a code we can define one error state, and if the assertion fails we go to this error state.

So making sure that our code works properly now can be decided by the reachability of the error state. If we can prove it mathematically that the error state can not be reached than we proved that all of our assertions were satisfied, and therefore our program on the observed invariant is error free.

2.3 Motivation

There are many formalisms for reachability analysis such as Bounded Modell Checking, which can provide a good counter example, a route to the error state. However the runtime -which depends on a good SMT solver- on really big cfa-s can be way too long. Also it can not prove that the error state is unreachable since it can only check bounded length of counter examples.

In order to make sure that there is no route to the error state we need to check all of them. Checking all the possible routes however can easily mean we have to examine infinite possibilities. For example the routes can differ, because of a variable's starting value -infinite possibilities for that- which already means it can have infinite different outcomes. To narrow down these possibilities abstraction can be really efficient. For example depending on our abstraction type we can make the similar routes into one group and instead of the infinite routes we will have a finite number of groups. One group contains those routes which are working in the same way on the abstraction, but it worked differently on the original.

The easiest example if we have a code:

```
int main(string args[])
{
  int arg = args[0];
  if(arg<=0){
    //do something
  }
  if(arg>0){
    //do other things
  }
}
```

arg can have infinite possible values so the program should have infinite routes, however if we use interval abstraction we can set two different groups that are working the same way, and this two contains all the possible values. $GroupOne : arg = (-\infty; 0); GroupTwo : arg = (1; \infty)$

2.4 Abstraction in general

The goal of the abstraction is to make it possible to compute *in reasonable time* that a certain error location is reachable or not. In theory there are three obstacles that make it impossible.

1. There can be infinite amounts of location
2. There can be a route (from the initial location), which is infinite (never reaches an end location)

3. There can be infinite amount of different routes.

Number 1 problem in cfa-s will not occur since locations are in between program statements and there can only be finite amount in one code.

Number 2 problem will be solved by fixpoints. See definition 1

Number 3 abstraction on the possible variables makes the similarly working routes the same, therefore narrowing the possibilities to a finite group of routes. See section 2.3

2.5 Abstraction analysis algorithm for CFA

The goal of the abstraction interpretation algorithm as shown in figure 2.2 is to state whether the error location is reachable or not. For this we need the CFA representation of the program and the abstraction type. More specifically a full library for that abstraction type, but we will expand on that later.

In the industrial world our goal is to prevent any erroneous behavior of our safty critical system. So detecting that an error exists is more important than to prove that the error not exists. Since the damage is much more costly if we fail in the first one. Imagine, if we say that there is no problem in our software that runs the subway system, but indeed there was, it can cause trains to collide which can cause serious -even life threatening- accidents. However, if we say that there is a possible error, the programmer can check, if that is really possible, and worse scenario is that he spent a lot of time examining the problem and found out that it is not possible, the damage is not that big.

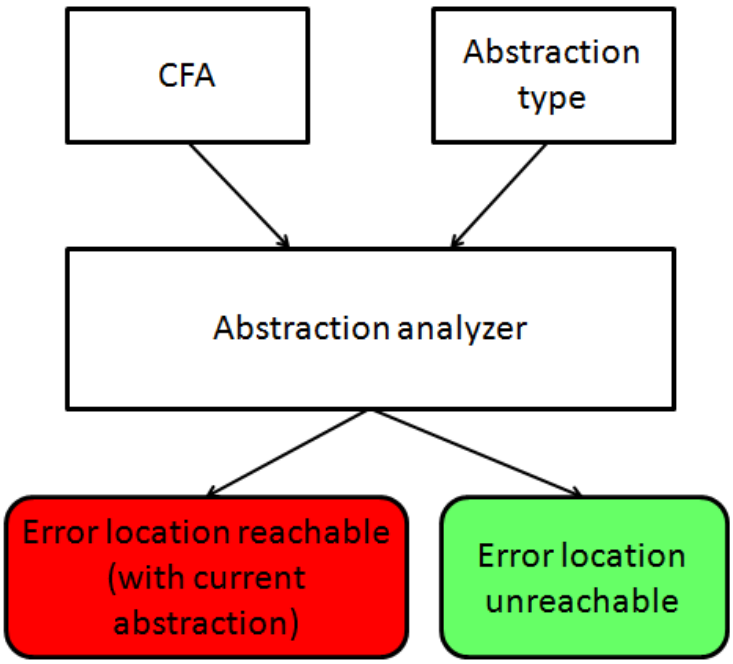

Figure 2.2: Abstraction analysis model

The question is to find a valid route from the initial location to the error location. Valid means that every step of this route were possible. It depend on the variables that the location has. As we already discussed it is efficient to do some type of abstraction on

these variables. In our work we will focus on interval abstraction, but there can be of course many other such as sign abstraction.

Abstraction interpretation on CFA-s means that we try to discover all the reachable locations starting from the initial location of course. We will put a label on every location that we have reached, and if no more label can be put -or changed- we reached a fixpoint (see definition 1).

The purpose of the label is not only to mark the reached location, but also is to be able to tell whether an out edge from the location can be possible or not. That of course depends on the variables in the program, so the label is some kind of representation of the possible variable values at the location that have been marked with it. In abstract interpretation this is of course some abstract form of the values. in sign abstraction it can be the possible signs of the variable in the location.

So one thing that an abstraction type should provide is a label for the variables, from which we can decide an assumption out edge is possible, and this label should be able to follow the changes of assign and havoc out edges. Now let us say that it is given.

So the first step is to put the initial label on the initial location. (let us say that an initial label is given for any abstraction type). Then let us say that the reached locations (marked with label) set is $ReachedLocations = initialLocation$. The next step is to see if we reached an error location already. If we did we can stop, because an error location is reachable. Otherwise we do the next iteration.

In the next iteration we get all of the out edges of the $ReachdLocations$. And if the label allows it, than we put a new label onto the target location. There can be two different scenario. First the target location has no label yet,than this is a new location, we put the source location's label modified by the edge (assumption, assignment or havoc of course given by the abstraction type). Second the target location already has a label. In this case the change can widen or narrow the possibilities on that location. Narrowing it should not be possible though since if the values on this location already can exist it should still exist later. However widening can occur a lot of consecutive times for example in a cycle. To make it more reasonable we can use widening tactics.

We use widening tactic if a certain location's possible values widens. For example if a cycle grows a variable by one in every cycle finding the fixpoint can be time consuming. So the main idea is to widen the possible values more than we actually should and therefore find the limitation earlier. There can be more possible tactics, but it is important to make sure that in the end we only allow the possible values.

So the iteration ends if we can reach the error location, or if we found a fixpoint. If we found a fixpoint it means that the error location is unreachable. This is proven, if the abstraction label functions were implemented correctly, but it does not depend on which abstraction type was used. On the other hand if we reached the error location, we can only say that the error can be reached with the current abstraction. It is still possible that the error can not be reached, just we lost too many information during abstraction.

As we stated we need an abstraction type for the abstract interpretation. figure 2.3 shows the summary of what we except from an abstraction type to be able to do.

- initial label: be able to mark an initial location (here every values should be possible)
- isValid: beang able to tell if after the assumption out edge the result can be valid (there are still possible values)

- addLabel: being able to change a given location's label with assignment, assumption or havoc out edge
- wideningtactic: The widening tactic should be specified (it depends on the label type)
- partitioningtactic: This applies the assignment, assumption or havoc statements to the label

Partitioning has not been mentioned, because in our work we did not implement any partitioning tactics. However our library gives the possibility to implement one. We will show some possible partitioning for interval abstraction.

Definition 1. fixpoint A fixpoint is when no further refinement can be made on the discovered reachable locations and no more location is reachable. In CFA this will mean that none of the outedges in any of the reached location can reach a new location nor can change the label of the reached locations. ▪

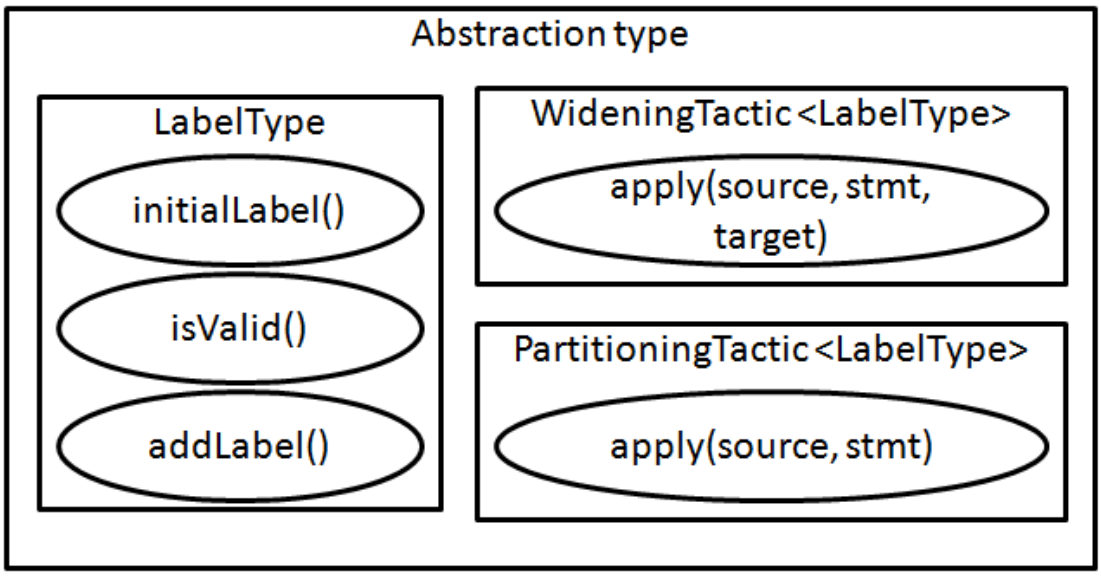

Figure 2.3: The structure of an Abstraction type

Chapter 3

Abstract Interpretation with Intervals

In this chapter we introduce our library for the interval abstraction. First we show the basic elements of intervals and the possible operations on them. Then we introduce Interval representation, which is a proper implementation of the interval abstraction type for abstraction interpretation. We prove the correctness of this representation. Show two possible widening tactics. Discuss some weaknesses of the abstraction and possible solutions. In section 3.11 we present how the algorithm works on an example.

3.1 Intervals

An interval consists of two bounds an $upperBound$ and a $lowerBound$, where bounds are whole numbers supplied with the opportunity to be infinite (both positive or negative) we will refer t that as ∞.

$Interval = (lowerBound; upperBound)$

The interval represents the whole numbers, between the two bounds (including the bound itself). So for example an interval $A = (0; 4)$ then it represents these numbers: $0; 1; 2; 3; 4$

We will say an interval is valid if $lowerBound \leq upperBound$. Otherwise we will say it is invalid. Empty intervals can be represented by invalid intervals.

Some general intervals are named as the following:

- $Ei = (1; -1)$ (empty interval)
- $Ii = (-\infty; \infty)$ (initial interval)

3.1.1 operations

3.1.1.1 section

section of two intervals: $Interval1(LB; UB) \cap Interval1(LB; UP) = Result(LB; UB)$

if $Interval1, Interval2$ is valid then

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

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

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

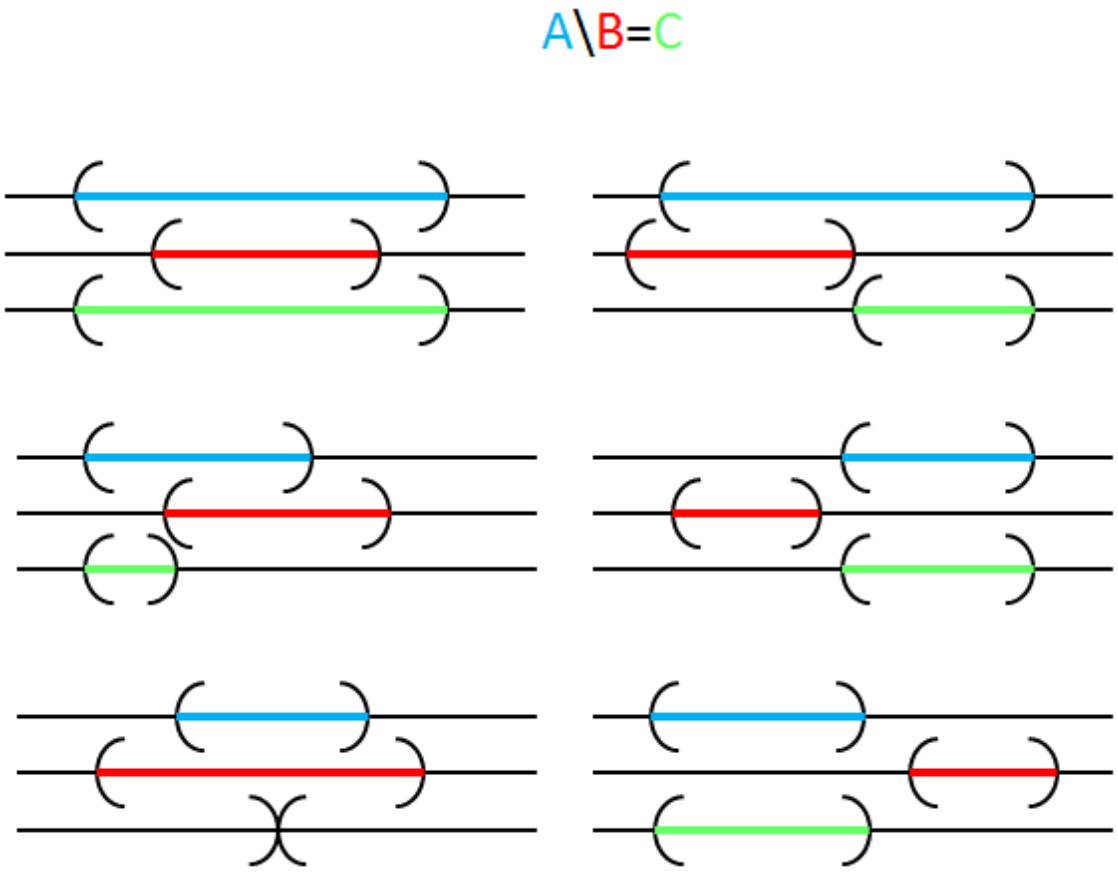

Figure 3.1: Interval A \ Interval B possible outcomes

Definition 5. 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 6. 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 7. 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 8. 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 9. 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 10. 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 11. 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 12. 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 $interval2$ 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 13. 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 14. A *condition* is an interval, which represents the possible values for a variable (can be calculated from any Expression used in the assignment). ▪

Definition 15. 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 16. Let variable var be represented by $interval$. An incorrect value is inside $interval$, but var is not allowed to have this value. ▪

Definition 17. 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 18. 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 19. 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 20. 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 21. If an Assume statement consists of a NOT function and is similar $Not(condition)$ 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 $Not(condition) = IntervalVar \setminus condition$ ▪

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 $IntervalVar \setminus condition$. 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 $Ii \setminus condition$.

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 22. 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