Loop Bound Analysis

Pritom Rajkhowa* and Peisen Yao† and Fangzhen Lin‡
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Emails: *prajkhowa@cse.ust.hk,†pyao@cse.ust.hk ‡flin@cse.ust.hk

***Abstract*—Determining the tight upper bounds on the iteration number of a given program can be used in some critical practical applications. Some prominent applications of the tight upper bounds analysis are program complexity analysis and worst-case execution time (WCET) analysis. This paper describes a system which addresses the problem of automatically inferring iteration bounds of imperative program loops. It makes use of a translation from these programs to first-order logic with quantifiers on natural numbers. We have implemented our method in a prototype tool NEWNAME and evaluated it on benchmarks from the literature.**

***Index Terms*—Automatic Program Verification, First-Order Logic, Mathematical Induction, Recurrences, SMT, Arithmetic**

I. INTRODUCTION

Static analysis of a program's quantitative behavior plays an important role in code optimization and verification. This information can be used to automatically prove certain program properties such as safety property, termination, and time complexity of the program. Specially memory-constrained environments such as embedded software systems must satisfy resource-constraints. To ensure the reliability of such programs, it important to prove that these programs terminate on any input within some time and memory limit [1]. In autonomous systems and application of cloud computing [2], [3], it is critical to find and monitor static energy usage information. Worst-case time bounds can help create constant-time implementations that prevent side-channel attacks [4], [5]. These examples address one of the fundamental questions that quantitative resource information can provide useful feedback for developers.

However, time complexity analysis is useful for any systems working with large data. Consider, for example, sorting algorithms: some of them are practically useless for very large arrays because they could spend hours sorting them, while other finish the computation within seconds. Most of the current static analysis tools in this area compute only termination or an asymptotic complexity of a program. But one can see that there is a significant difference between time complexities n^2 and $1000 \cdot n^2$, while the asymptotic complexity is the same. We present an algorithm for computing more precise time bounds. The need for the precision can be seen in the embedded systems: hardware which has to perform n operations within a certain time limit is more expensive than the hardware which has to perform just n operations.

Research interest in the field of loop bound analysis is quite recent. The most and development have occurred in the last decade. Previous works such as tools like SPEED [6], KoAT [7], PUBS [8], Rank [9], and LOOPUS [10] take a imperative program P and always derive a sound bounds for numerical. There are another category of tools which are based on the method of amortized analysis and type systems for functional programs [11], [12], [13]. The major drawback of these tools is that they can only associate potential with individual program variables or data structures. There is another categories of tools, such as CAMPY [14] and C^4B [15] , which provide user interaction to manual proofs of resource bounds. ✎ [yao: It is more important to summarize the technical dimensions and limitations, instead of the "techniques they use"]

In this work, we introduce a new framework for automatically deriving resource bounds on C programs. Another additional feature this framework provides is that when a tool fails to find a resource bound for the input program, then it provides sound user interaction during bound derivation. We have named the framework as NEWNAME which take a program P from the programmer and translate it to a set of axioms of first-order logic with quantifiers on natural numbers proposed recently by Lin. Then it uses a translated set of axioms derived the resource bound of the input program. When it failed to derive any bound, then it expected bound value B from the programmer. NEWNAME produces either of the following output

- prove that input bound value B satisfied on all inputs by the program P using translated axioms.
- produce a counterexample if B satisfied for some input.

Compared to more classical approaches based on ranking functions or amortized reasoning, our tool inherits the benefits of not explicitly generate any invariant. However, state-of-the-art classical approaches are often unable or inefficient to derive suitable non-linear invariants to find suitable resource bounds. For example, inferring non-linear invariants is a long-standing challenge for conventional safety property verifiers as well [16], [17]. The crux of our technique is to derive non-linear bound resource bounds by constructing a proof that runs of a program path satisfy a given that. NEWNAMEś ability to reason about non-linear behavior the fact that it uses decision procedure for the combination of theories of linear arithmetic and uninterpreted functions. We use auxiliary uninterpreted

functions to abstract nonlinear functions to build an initial abstraction, then the theorem prover lazily refines an approximation of the theory of non-linear. If it finds a model, it tests the model as a model of the path formula under the standard model of non-linear arithmetic. This place our approach in an advantageous position as compared to other state-of-the-art which failed to find the resource bounds the program with the non-linear body. Another core technical contribution of NEWNAME is that our technique does not require special or separate analysis for compositional resource analysis, like exiting approaches [15], because of the flexibility provided by our translation. The translated axioms give the relationship between the input and output values of the program variables which provide us the pliability to reason about any property of the program.

In particular, NEWNAME is integrated with strong recurrence solver which is capable of finding closed forms—effectively of a certain class of P-finite and C-finite recurrences relations as well as certain classes of conditional recurrences, a simple case of mutual recurrences and multivariable recurrence relations. NEWNAME can is able to infer resource bounds on C programs with mutually-recursive functions and nested loops.

However, we want to highlight the major weakness of existing approaches with the help of the following examples of C code snippet

P_1	P_2

`int x , C;`
`while(x < C)`
`{`
`  x = x + 1;`
`}`
	`int x ,y,C;`
	`while(x + y < C)`
	`{`
	`  x = x + 1;`
	`  y = y + 1;`
	`}`

The potential-based techniques and amortized analysis base tool C^4B can easily find the bound for the program P_1 where C^4B return the bound $1.00|[0, C]|$ for the program P_1. But all the tools including C^4B failed to find the bound for the program P_2. NEWNAME can find the bound of both programs $|[0, C - x]|$ and $|[0, (C - x - y)/2]|$ for P_1 and P_2 respectively. To illustrate how NEWNAME works, consider the program P_2. Lin's [18] translation of the above program P generates the following a set of axioms $\Pi_P^{\vec{X}}$ after some simple simplifications where $\vec{X} = \{x, y, C\}$

$$
\begin{aligned}
&C_1 = C, x_1 = x_3(n), y_1 = y_3(n),\\
&x_3(0) = x, y_3(0) = y,\\
&x_3(n+1) = x(n) + 1, y_3(n+1) = y_3(n) + 1,\\
&\neg(x_3(N) + y_3(N) < C),\\
&\forall n.n < N \rightarrow (x_3(n) + y_3(n) < C)
\end{aligned}
$$

where x_1, y_1 and C_1 denote the output values of a, b, z, x and y, respectively, $x_6(n)$ and $y_6(n)$ the values of x and y during the nth iteration of the loop, respectively. Also N is a natural number constant, and the last two axioms say that it is exactly the number of iterations the loop executes before exiting. Our recurrence solving tool recSolve(RS) can find the closed-form solutions of $x_3(n)$ and $y_3(n)$, which yields the closed-form solution $x_3(n) = n - x$ and $y_3(n) = n - y$ respectively. Those can be used to simplify the recurrence for $y_3(n)$ into following set of axoims after eliminates recurrence relation for $x_3()$, and $y_3()$

$$
\begin{aligned}
&x_1 = N + x, y_1 = N + y, C_1 = C,\\
&(2 * N + x + y \geq C)\\
&\forall n.n < N \rightarrow (2 * n + x + y < C)
\end{aligned}
$$

The system tried to derive using algorithm XXX derived $N = (C - x - y)/2$, then it tried to prove using SMT solver. If it is able to successfully prove that, then it successfully derived bound $|[0, (C - x - y)/2]|$.

We have recently constructed a fully automated program verification system called NEWNAME [19], [20], for programs with integer assignments and arrays. NEWNAME is based on the translation Lin [18] which translates the programs to first-order logic with quantifiers over natural numbers. Translation of the loop body in the program generates a number of axioms based on recursive relationships. Simplify the translated set of axioms by trying to compute closed-form solutions of recursive relationships. Simplification process help to reduced quantified axioms which help it reducing the complexity in theorem proving. The translation of multi-path programs with loops results in conditional recurrences. Our proposed recurrence solver (RS) which is capable of finding closed-form solutions of conditional recurrences is integrated with NEWNAME which clearly shows its effectiveness in performance on state-of-art benchmark programs. The theorem proving part uses Z3 directly when the translated axioms have no inductive definitions. Otherwise, our system searches for a proof by mathematical induction using Z3 as the base theorem prover.

The idea of using closed forms of recurrence relations to approximate loops has appeared in a number of previous work. The techniques proposed by Rodríguez-Carbonell and Kapur [21] and Kovács[22] are based on solving recurrence relations for discovering invariant polynomial equations. On the other hands, works [23], [24], [25] are based on recurrences analysis which focuses on over-approximate analysis of general loops. Our approach gets the best of both worlds. In contrast, our recurrences precisely aim to analyze the accurate semantics of general loops.

To summarize, this paper makes the following key contributions:

- We develop the first automatic amortized analysis for C programs. It is naturally compositional, tracks size changes of variables to derive global bounds, can handle mutually-recursive functions, generates resource abstractions for functions, derives proof certificates, and handles resources that may become available during execution.
- It can detect logarithmic bounds.
- It distinguishes different branches inside loops and computes bounds for each of them separately.

- We prove the soundness of the analysis.
- We implemented our resource bound analysis in the publicly available tool NEWNAME .
- We present experiments with NEWNAME on a benchmark of C^4B which is more than 2900 lines of C code. A detailed comparison shows that our prototype is the only tool that can derive global bounds for larger C programs while being as powerful as existing tools when deriving linear local bounds for tricky loop and recursion patterns.

II. TRANSLATION

Our translator consider programs in the following language:

```
E ::= array(E,...,E) |
operator(E,...,E)
B ::= E = E |
boolean-op(B,...,B)
P ::= array(E,...,E) = E |
if B then P else P |
P; P |
while B do P
```

Given a program P, and a language $\vec{X}$, our system generates a set of first-order axioms denoted by $\Pi_P^{\vec{X}}$ that captures the changes of P on $\vec{X}$. Here, a language means a set of functions and predicate symbols. For $\Pi_P^{\vec{X}}$ to be correct, $\vec{X}$ needs to include all program variables in P as well as any functions and predicates that can be changed by P. The axioms in the set $\Pi_P^{\vec{X}}$ are generated inductively on the structure of P. The algorithm is described in detail in [18] and implementation is explained in [19]. The inductive cases of translations are given in the table provided in the supplementary information[1]. We have extended our translation programs with arrays; the extension is described in detail in [20].

The translation for the sequence, conditonal, and while loops remain the same as that of the deterministic programs presented in [18]. We have updated those rules by integrating the simplification process, which are briefly described as follows:

Definition 2.1: When `P` is `if B then P1 else P2` then $\Pi_P^{\vec{X}}$ is the set of following axioms:

$B \rightarrow \varphi$, for each $\varphi \in \Pi_{P1}^{\vec{X}}$,

$\neg B \rightarrow \varphi$, for each $\varphi \in \Pi_{P1}^{\vec{X}}$,

We assume here that for each boolean expression B, there is a corresponding formula B in our first-order language.

Translator uses a systematic technique for the simplifying condition of each axoim $\varphi \in \Pi_P^{\vec{X}}$ of the following form where $\Pi_P^{\vec{X}}$ is the set of axioms generated after the translation of the conditional statement:

$$\forall \vec{x}.X_1(\vec{x}) \quad = \quad ite(B, \hat{E_1}, \hat{E_2}) \tag{1}$$

where E_1, E_2 are the translation of the expressions. It uses two different rules which are presented as follows:

[1] https://github.com/VerifierIntegerAssignment/VIAP_ARRAY/blob/master/Document/Inductive_Translation.pdf

- When B is boolean expression, and $E_1 = E_2$, then φ is simplified to $X_1(\vec{x}) = E_1$.
- When B is boolean expression, and evaluate B is true, then φ is simplified to $X_1(\vec{x}) = E_1$.

Definition 2.2: When `P` is `while B do P1` then $\Pi_P^{\vec{X}}$ is the set of following axioms: $\varphi(n)$, for each $\varphi \in \Pi_{P1}^{\vec{X}}$,

$X^i(\vec{x}) = X^i(\vec{x}, 0)$, for each $X^i \in \vec{X}$

$smallest(N, n, \neg B(n))$,

$X_1^i(\vec{x}) = X^i(\vec{x}, N)$, for each $X^i \in X$ where n is a new natural number variable not already in φ, and N a new constant not already used in $\Pi_{P1}^{\vec{X}}$. For each formula or term α, $\alpha(n)$ is defined inductively as follows: it is obtained from α by performing the following recursive substitutions:

- for each $X^i \in \vec{X}$, replace all occurrences of $X_1^i(e_1, ..., e_k)$ by $X^i(e_1(n), ..., e_k(n), n+1)$, and
- for each program variable X in α, replace all occurrences of $X(e_1, ..., e_k)$ by $X(e_1(n), ..., e_k(n), n)$. Notice that this replacement is for every program variable X, including those not in $\vec{X}$.

$smallest(N, n, \neg B(n))$ is a shorthand for the following formula

$$\neg B(N), \tag{2}$$
$$\forall n.n < N \rightarrow B(n) \tag{3}$$

Lets consider $\vec{\sigma}$ represents the set of inductively defined axioms during the translation. If all the condition axioms $\vec{\sigma}$ are of the following form for some $h \geq 1$

$$\begin{aligned} X^i(e_1(n), ..., e_k(n), n+1) \quad = \\ ite(\theta_1, f_1(X^i(e_1(n), ..., e_k(n)), n), \\ ite(\theta_2, f_2(X^i(e_1(n), ..., e_k(n)), n)..., \\ ite(\theta_h, f_h(X^i(e_1(n), ..., e_k(n)), n) \\ , f_{h+1}(X^i(e_1(n), ..., e_k(n)), n))), \\ \text{for } 1 \leq i \leq h \wedge X^i \in \vec{X}, \end{aligned} \tag{4}$$

where $\theta_1, \theta_2, ..., \theta_h$ are boolean expressions, and $f_1(x, y), f_2(x, y), ..., f_{h+1}(x, y)$ are polynomial functions of x and y. The translator uses a systematic technique to reconstruct $\Pi_P^{\vec{X}}$ as following:

- if all the conditions θ_i, $1 \leq i \leq h$, in it are independent of the recurrence variable n, the translator splits the set of axioms $\Pi_P^{\vec{X}}$ to a sequence of set of axioms $\Pi_{P1}^{\vec{X}}$, $\Pi_{P2}^{\vec{X}}$,...,$\Pi_{Ph}^{\vec{X}}$, $\Pi_{Ph+1}^{\vec{X}}$ corresponding to conditions $\theta_1, \theta_2, ..., \theta_h, \theta_{h+1}$ respectively where $\theta_{h+1} = (\neg\theta_1 \wedge \neg\theta_2 \wedge ..... \wedge \neg\theta_h)$. For any Π_i such that $1 \leq i \leq h$. $\Pi_{P1}^{\vec{X}}$ consist of the following set of axioms along with all non-conditional inductively defined axioms of $\Pi_P^{\vec{X}}$ which corresponds to boolean expression θ_i

  $X^i(\vec{x}, 0) = X^i(\vec{x})$, for each $X^i \in \vec{X}$

  $smallest(N, n, \neg(B(n) \wedge \theta_i))$,

$X^i(e_1(n),...,e_k(n),n+1) = f_i(X^i(e_1(n),...,e_k(n)),n),$

$X_1^i(\vec{x}) = X^i(\vec{x},N),$ for each $X^i \in X$

Then reconstruct $\Pi_P^{\vec{X}}$ from $\Pi_{P1}^{\vec{X}}$, $\Pi_{P2}^{\vec{X}}$,...,$\Pi_{Ph}^{\vec{X}}$, ,$\Pi_{Ph+1}^{\vec{X}}$ following the translation rule definition presented in **??**.

$\varphi(\vec{X_1}/\vec{Y_1}) \to \varphi$, for each $\varphi \in \Pi_{P1}^{\vec{X}}$,

$\varphi(\vec{X}/\vec{Y_1}) \to \varphi$, for each $\varphi \in \Pi_{P2}^{\vec{X}}$

$\vdots$

$\varphi(\vec{X_1}/\vec{Y_h}) \to \varphi$, for each $\varphi \in \Pi_{Ph}^{\vec{X}}$,

$\varphi(\vec{X}/\vec{Y_h}) \to \varphi$, for each $\varphi \in \Pi_{Ph+1}^{\vec{X}}$

where any $\vec{Y_i} = (Y_i^1,...,Y_i^k)$ is a tuple of new function symbols such that each Y_i^j is of the same arity as X^j in $\vec{X}$ such that $1 \le j \le k$, $\varphi(\vec{X_1}/\vec{Y_i})$ is the result of replacing in φ each occurrence of X_1^j by Y_i^j, and similarly for $\varphi(X/\vec{Y_i})$.

- if the condition θ_i, $1 \le i \le h$, contains only arithmetic comparisons $>$ or $<$, polynomial functions of n, and does not mention $X(n)$. Then translator tries to compute the ranges of θ_i as follows. For each $1 \le i \le h$, let

$$\phi_i = \neg\theta_1 \wedge \cdots \wedge \neg\theta_{i-1} \wedge \theta_i.$$

Thus ϕ_i is the condition for $X(n+1)$ to take the value $f_i(X(n),n)$. Our system then tries to compute h constants $C_1,...C_h$ such that

$$0 = C_0 < C_1 < \cdots < C_h,$$

and for each $i \le h+1$,

$$\forall n. C_{i-1} \le n < C_i \to \phi_{\pi(i)}(n),$$
$$\forall n. n \ge C_i \to \neg\phi_{\pi(i)}(n).$$

where π is a permutation on $\{1,...,n\}$. The computation of these constants and the associated permutation π is done using an SMT solver: it first computes k for which $\phi_k(0)$ holds, and then asks the SMT solver to find a model of

$$\forall n. 0 \le n < C \to \phi_k(n), \forall n. n \ge C \to \neg\phi_k(n).$$

If it returns a model, then set $C_1 = C$ in the model, and let $\pi(1) = k$, and the process continues until all C_i are computed. If this fails at any point, then the translator aborts this simplification step and return $\Pi_P^{\vec{X}}$.
Once the system succeeds in computing these constants, the translator reconstruct $\Pi_P^{\vec{X}}$ to a sequence of set of axioms $\Pi_{P1}^{\vec{X}}$, $\Pi_{P2}^{\vec{X}}$,...,$\Pi_{Ph}^{\vec{X}}$, ,$\Pi_{Ph+1}^{\vec{X}}$ corresponding to constants $C_1, C_2, ..., C_h, C_{h+1}$ respectively. For any Π_i such that $1 \le i \le h$. $\Pi_{P1}^{\vec{X}}$ consist of the following set of axioms along with all non-conditional inductively defined axioms of $\Pi_P^{\vec{X}}$ which corresponds to constant C_i

$X^i(\vec{x},0) = X^i(\vec{x})$, for each $X^i \in \vec{X}$

$smallest(N,n,\neg(B(n) \wedge n < C_i))$,

$X^i(e_1(n),...,e_k(n),n+1) = f_i(X^i(e_1(n),...,e_k(n)),n),$

$X_1^i(\vec{x}) = X^i(\vec{x},N),$ for each $X^i \in X$

Then reconstruct $\Pi_P^{\vec{X}}$ from $\Pi_{P1}^{\vec{X}}$, $\Pi_{P2}^{\vec{X}}$,...,$\Pi_{Ph}^{\vec{X}}$, ,$\Pi_{Ph+1}^{\vec{X}}$ following the translation rule definition presented in **??** as presented in previous case.

III. Motivating Example Illustrating our Technique

In this section, We illustrate the main ideas of our approach using the above simple example. In III-B, we present an iterative implementation of Naive algorithm for pattern searching. In III-B1, we presentation translated set axioms generated from the input program by NewNameİn III-B2, we give explanation of how NewName automatically derived expected bound.

A. An Iterative Implementation Zohar Manna's Version of Integer division algorithm

In this example, we illustrate how NewName find loop bound where all other states of art tools failed to do. In this example, we present an iterative implementation of Zohar Manna's version of integer division algorithm [26]. In III-A1, we present translated set axioms generated from the input program by translator module of NewName . In this example, we give an explanation of how our system automatically derives the loop bound of the program.

The C code snippet P with variables $\vec{X} = \{a,b,x,y,z\}$. The two integer variables a and b are assigned with two non-deterministic function call. The algorithm for dividing a number a by another number b. When the algorithm terminates, it coming up with a quotient x and a remainder y under the assumption of $a \ge 0$ and $b > 0$.

```
int a, b, x, y, z;
a = nondetint();
b = nondetint();
x = 0; y = 0; z = a;
assume(a >= 0);
assume(b > 0);
while(z != 0) {
    if (y + 1 == b)
    {
        x = x + 1;
        y = 0;
        z = z - 1;
    }
    else
    {
        y = y + 1;
        z = z - 1;
    }
}
```

1) Translation of Program: The translation module of NEWNAME implements Lin's [18] translation and generates the following set of axioms $\Pi_P^{\vec{X}}$ for program P, where $\vec{X} = \{a, b, x, y, z\}$.

$$
\begin{aligned}
&a_1 = nondetint_2, y_1 = y_6(N), b_1 = nondetint_3,\\
&x_1 = x_6(N), z_1 = z_6(N),\\
&y_6(0) = 0, x_6(0) = 0, z_6(0) = nondetint_2,\\
&\forall n.y_6(n+1) = ite((y_6(n)+1) = nondetint_3\\
&\qquad\qquad ,0, y_6(n)+1),\\
&\forall n.x_6(n+1) = ite((y_6(n)+1) = nondetint_3,\\
&\qquad\qquad x_6(n)+1, x_6(n)),\\
&\forall n.z_6(n+1) = z_6(n) - 1,\\
&\neg(z_6(N) \neq 0),\\
&\forall n.n < N \rightarrow z_6(n) \neq 0
\end{aligned}
$$

where a_1, b_1, z_1, x_1 and y_1 denote the output values of a, b, z, x and y, respectively, $x_6(n)$, $y_6(n)$ and $z_6(n)$ the values of x, y and z, respectively during the nth iteration of the loop. The conditional expression $ite(c, e_1, e_2)$ has value e_1 if c holds and e_2 otherwise. Also N is a natural number constant, and the last two axioms say that it is exactly the number of iterations the loop executes before exiting.

The translation of assumption results as follows:

$$nondetint_2 \geq 0 \tag{5}$$

$$nondetint_3 > 0 \tag{6}$$

After computing the closed-form solutions for $x_6()$ and $y_6()$ by RS, it substituting them in $\Pi_P^{\vec{X}}$ and then NEWNAME eliminates them, and updated $\Pi_P^{\vec{X}}$ which is represented by the following axioms:

$$
\begin{aligned}
&a_1 = nondetint_2,\\
&z_1 = (nondetint_2 - N),\\
&b_1 = nondetint_3,\\
&y_1 = ite(0 \leq N \wedge N < nondetint_3, N,\\
&ite(N = nondetint_3, 0, N - nondetint_3)),\\
&x_1 = ite(0 \leq N \wedge N < nondetint_3, 0, 1),\\
&\neg((nondetint_2 - N) \neq 0),\\
&\forall n.n < N \rightarrow (nondetint_2 - n) \neq 0
\end{aligned}
$$

NEWNAME compute the bound $\mathcal{B}_{outer}^{\vec{X}} = N$ by deriving $N_2 = n - m + 1$ for the following set of equations

$$
\begin{aligned}
&\neg((nondetint_2 - N) \neq 0),\\
&\forall n.n < N \rightarrow (nondetint_2 - n) \neq 0
\end{aligned}
$$

B. An Iterative Implementation of Pattern Searching

We illustrate the main ideas of our approach using the another simple example. The C code snippet P with variables $\vec{X} = \{n, m, p, t, r, i, j\}$ where p, t and r are two inputs integer array variables of size m, n and n respectively such that $n > m$. The code snippet P tries to find the occurs of pattern p in main text array t. If p match any substring in t, then it stores the starting index of the matched substring of t in r.

```
1.  int n, m, i, j, k=0;
2.  int p[m], t[n], r[n];
3.  for (i = 0; i <= n - m; i++) {
4.      for (j = 0; j < m; j++)
5.          if (t[i + j] != p[j])
6.              break;
7.      if (j == m){ r[k]=i; k=k+1;}
}
```

The number of comparisons in the worst case is $m*(n-m+1)$. Here we will demonstrate how NEWNAME automatically derive that. This example to provide an intuitive description of three important aspects of our approach. Firstly, how the system effectively derive bound without explicitly generating loop invariants. Secondly, how the system effectively handles the program with nested loops. Lastly, how our system handles the program with a unstructured keyword such as break, goto, etc

1) Translation of Program: Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following with resprect to variables $\vec{X} = \{A, B, C, i, j, k, n, d2array\}$:

$$
\begin{aligned}
&1.p_1 = p, m_1 = m, t_1 = t, n_1 = n, r1 = r\\
&2.\forall x_1, x_2.d1array_1(x_1, x_2) = d1array(x_1, x_2)\\
&3.i_1 = i_{14}(N_2)\\
&4.j_1 = j_{14}(N_2)\\
&5.k_1 = k_{14}({}_N2)\\
&6.break\_1\_flag_1 = break\_1\_flag_{14}(N_2)\\
&7.\forall n_1, n_2.j_5(n_1+1, n_2) =\\
&\quad ite((ite((d1array(t, (i_{14}(n_2) + j_5(n_1, n_2))) \neq\\
&\qquad d1array(p, j_5(n_1, n_2))), 1, 0) = 0),\\
&\qquad (j_5(n_1, n_2) + 1), j_5(n_1, n_2))\\
&8.\forall n_1, n_2.break\_1\_flag5((n_1+1), n_2) =\\
&\quad ite((d1array(t, (i_{14}(n_2) + j_5(n_1, n_2))) \neq\\
&\qquad d1array(p, j_5(n_1, n_2))), 1, 0)\\
&9.\forall n_2.j_5(0, n_2) = 0\\
&10.\forall n_2.break\_1\_flag5(0, n_2) = break\_1\_flag_{14}(n_2)\\
&11.\forall n_2.((j_5(N_1(n_2), n_2) \geq m) \vee\\
&\qquad (break\_1\_flag5(N_1(n_2), n_2) \neq 0))\\
&12.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow ((j5(n_1, n_2) < m) \wedge\\
&\qquad (break\_1\_flag5(n_1, n_2) = 0))\\
&13.\forall n_2.i_{14}(n_2+1) =\\
&\qquad ite((j_5(N_1(n_2), n_2) \neq m),\\
&\qquad\quad (i_{14}(n_2)+1), i_{14}(n_2))
\end{aligned}
$$

$$14.\forall n_2.j_{14}(n_2+1) = j_5(N_1(n_2), n_2)$$
$$15.\forall n_2.k_{14}(n_2+1) = ite((j_5(N_1(n_2), n_2) = m), (k_{14}(n_2)+1), k_{14}(n_2))$$
$$16.d1array_{14}(x_1, x_2, (n_2+1)) = ite((j_5(N_1(n_2), n_2) = m), ite(((x_1 = r) \wedge (x_2 = k_{14}(n_2))), (n_2+0), d1array_{14}(x_1, x_2, n_2)), d1array_{14}(x_1, x_2, n_2))$$
$$17.\forall n_2.break\_1\_flag_{14}((n_2+1)) = break\_1\_flag_5(N_1(n_2), n_2)$$
$$18.k_{14}(0) = 0$$
$$19.j_{14}(0) = j$$
$$20.break\_1\_flag_{14}(0) = 0$$
$$21.(N_2 > (n-m))$$
$$22.(n_2 < N_2) \rightarrow (n_2 <= (n-m))$$

2) Bound Analysis: To compute the bound $\mathcal{B}^{\vec{X}}_{Inner}$ using case analysis as system both the equations (11) and (12), which corresponds to while loop condotion, involves function(s) generated from program body, the following set of axoims are used

$$7.\forall n_1, n_2.j_5(n_1+1, n_2) = ite((ite((d1array(t, (i_{14}(n_2) + j_5(n_1, n_2))) \neq d1array(p, j_5(n_1, n_2))), 1, 0) = 0), (j_5(n_1, n_2)+1), j_5(n_1, n_2))$$
$$8.\forall n_1, n_2.break\_1\_flag5((n_1+1), n_2) = ite((d1array(t, (i_{14}(n_2) + j_5(n_1, n_2))) \neq d1array(p, j_5(n_1, n_2))), 1, 0)$$
$$9.\forall n_2.j_5(0, n_2) = 0$$
$$10.\forall n_2.break\_1\_flag5(0, n_2) = break\_1\_flag_{14}(n_2)$$
$$11.\forall n_2.((j_5(N_1(n_2), n_2) \geq m) \vee (break\_1\_flag5(N_1(n_2), n_2) \neq 0))$$
$$12.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow ((j5(n_1, n_2) < m) \wedge (break\_1\_flag5(n_1, n_2) = 0))$$

After analyzing equations, it derives the following cases.
Case 1 When the following condition is holds

$$\forall n_1, n_2.(d1array(t, (i_{14}(n_2) + j_5(n_1, n_2))) \neq d1array(p, j_5(n_1, n_2))) \neq False$$

. Now simplify the equation with respect to the assumption.

$$7.\forall n_1, n_2.j_5(n_1+1, n_2) = (j_5(n_1, n_2)+1)$$
$$8.\forall n_1, n_2.break\_1\_flag5((n_1+1), n_2) = 0$$
$$9.\forall n_2.j_5(0, n_2) = 0$$
$$10.\forall n_2.break\_1\_flag5(0, n_2) = break\_1\_flag_{14}(n_2)$$

Now solve the above equations using RS with respect to their corresponding initial values which results $j_5(n_1, n_2) = n_1$ and $break\_1\_flag5(n_1, n_2) = 0$. After substituting the solution and get rid of equations (7), (8), (9) and (10) with resulted set of axioms are as follows:

$$11.\forall n_2.(N_1(n_2) \geq m) \vee (0 \neq 0)$$
$$12.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < m) \wedge (0 = 0)$$

The above set of equations are simplified by getting rid of redundant condition results following a set of equations

$$11.\forall n_2.(N_1(n_2) \geq m)$$
$$12.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < m)$$

From the above equations, NEWNAME derives $N_1(n_2) = m$ which is the bound $\mathcal{B}^{\vec{X}}_{Inner(case_1)} = m$.
Case 2 When the following condition is holds

$$\forall n_1, n_2.(n_1 < N_1(n_2) \implies (d1array(t, (i_{14}(n_2) + j_5(n_1, n_2))) \neq d1array(p, j_5(n_1, n_2))) \neq False) \wedge ((d1array(t, (i_{14}(n_2) + j_5(N_1(n_2), n_2))) \neq d1array(p, j_5(N_1(n_2), n_2))) == False)$$

Now simplify the equation with respect to the assumption.

$$7.\forall n_1, n_2.j_5(n_1+1, n_2) = ite(0 < n_1 < N_1(n_2), j_5(n_1, n_2)+1, j_5(n_1, n_2))$$
$$8.\forall n_1, n_2.break\_1\_flag5((n_1+1), n_2) = ite(0 < n_1 < N_1(n_2), 0, 1)$$
$$9.\forall n_2.j_5(0, n_2) = 0$$
$$10.\forall n_2.break\_1\_flag5(0, n_2) = break\_1\_flag_{14}(n_2)$$

Now solve the above equations using RS with respect to their corresponding initial values which results $j_5(n_1, n_2) = ite(0 < n_1 < N_1(n_2), n_1, N_1(n_2))$ and $break\_1\_flag5(n_1, n_2) = ite(0 < n_1 < N_1(n_2), 0, 1)$. After substituting the solution and get rid of equations (7), (8), (9) and (10) with resulted set of axioms are as follows:

$$11.\forall n_2.(N_1(n_2) \geq m) \vee (1 \neq 0)$$
$$12.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < m) \wedge (0 = 0)$$

The above set of equations are simplified by getting rid of redundant condition results following a set of equations

$$11.\forall n_2.(N_1(n_2) \geq m)$$
$$12.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < m)$$

From the above equations, NEWNAME derives $N_1(n_2) = m$ which is the bound $\mathcal{B}^{\vec{X}}_{Inner(case_2)} = m$. Now $\mathcal{B}^{\vec{X}}_{Inner} = max(\mathcal{B}^{\vec{X}}_{Inner(case_1)}, \mathcal{B}^{\vec{X}}_{Inner(case_2)}) = max(m, m) = m$

Similary, NEWNAME compute the bound $\mathcal{B}^{\vec{X}}_{outer} = n - m + 1$ by deriving $N_2 = n - m + 1$ for the following set of equations

$$21.(N_2 > (n - m))$$
$$22.(n_2 < N_2) \rightarrow (n_2 <= (n - m))$$

IV. EVALUATION

NEWNAME is implemented as a stand-alone application, which uses Sympy [27] computer algebra system and Z3 [28] SMT solver. The current implementation of NEWNAME executes using a single thread. The tool along with source code is publicly available in the following URL:

https://github.com/VerifierIntegerAssignment/LoopBoundTool

NEWNAME is integrated with our recurrence solver framework, RS, to solve a recurrence. All of the experiments are conducted using Intel® Core™ i3 1.8 GHz processor running with 4GB of memory, Ubuntu 16.04 LTS. The source code and the full experiments are available on

https://github.com/pritomrajkhowa/

A. Benchmarks

Our evaluation subjects include a set of C programs gathered 60 challenging loop and recursion patterns from previous publications [29], [6], [15]. These programs can be divided into three categories

- 30 programs are from a benchmark suite of C^4B [15] which is used compared capability of C^4B with state-of-art bounds generated tools KoAT [7], Rank [9], LOOPUS [10], SPEED [6] and PUBS [8] in its experiment
- 20 programs are from the open-source code. Most of the state-of-art bounds generated tools cannot handle these programs. We have chosen these programs to demonstrate the gap exists in current bound generation tools.
- Another 10 programs are from the open-source code. The main purpose of selection of these program to demonstrate how our tool derive non-linear bound of the programs.

B. Experimental setup

We performed an empirical evaluation in order to answer the following research questions:

- **RQ1** How effectiveness can NEWNAME derived loop bound of programs compared to the state-of-the-art tools?
- **RQ2** How our system NEWNAME tried to fill some of the gap which exits in the currently available state-of-the-art tools?

C. Results

*1) **RQ1**:*

*2) **RQ2**:*

V. RELATED WORK

A. Worst-Case Execution Time(WCET)

WCET of the program is the maximum or worst possible execution time it takes to execute it [30]. WCET problems are typically defined for real-time programs and systems which need to satisfy stringent constraints for all iterations and recursion. The analysis of upper bounds on the execution is important to demonstrate that such real-time satisfaction of these constraints on resources such as the cache and branch speculation. It is not always possible to obtain upper bounds on execution times for programs as it is an undecidable problem. Research on WCET analysis has been an actively researched area for two decades. Some important early works [31], [32] determine WCET of the program source code without considering the timing effects of the underlying micro-architecture. The framework for parametric WCET analysis presented in [33] derive iteration bounds from symbolic expressions by Instantiating the symbolic expressions with specific inputs. Then it determines the WCET of programs. Another interesting work in this direction is [34] where proposed approach automatically identifies induction variables and recurrence relations of loops using abstract interpretation [35]. Closed-form finding mechanism is templates based where precomputed closed form templates are used to find the closed form solution. Then iteration bounds are derived. The approach presented in [36] is based on data flow analysis and interval based abstract interpretation to find iteration bounds. All these approaches for WCET analysis can handle loop with loops with the relatively simple flow and arithmetic. For more complex loops iteration bounds are supplied manually in the form of auxiliary program annotations. Fully automated approach [37] overcome drawbacks of the aforementioned approaches where user guidance is important. Although [37] can infer iteration bounds for special classes of loops with non-trivial arithmetic and flow. Unlike [37], our proposed approach can handle more complex program such as programs with multipath control flow. In our work, we tried addresses a problem related to predicting the performance of a program. We tried to achieve that by considering the cost for each instruction of program and primarily determining accurate bound of iterations and recursion.

B. Invariant Generation and Cost Analysis

Loop invariants describe logical properties of the loop that holds for every iteration of the loop. Loop Invariant usually used to infer bounds on program resources. The approach proposed in tool SPEED [6] instruments counters into the program at different program locations as artificial variables. Then it composed those counter using a proof-rule-based algorithm and computes their upper bounds by using abstract interpretation based techniques to derive linear invariant. This approach is further extended to derive more complex loop bounds in in [29]. In this work, abstract interpretation is used to compute disjunctive invariants and it uses proof-rules using max, sum, and product operations on bound patterns. As a

result, non-linear symbolic bounds of multi-path loops are obtained by deploying product operations on bound patterns in conjunction with SMT reasoning in the theory of linear arithmetic and arrays. [38] also another approach which is based on abstract interpretation based invariant generation in conjunction with so-called cost relations. Cost relations extend recurrence relations and can express recurrence relations with non-deterministic behavior which arise from multi-path loops. Iteration bounds of loops are inferred by constructing evaluation trees of cost relations and computing bounds on the height of the trees. For doing so, linear invariants and ranking functions for each tree node are inferred. Unlike the aforementioned techniques, we do not use abstract interpretation but deploy a recurrence solving approach to generate bounds on simple loops. Contrarily to [6], [29], [38], our method is limited to multi-path loops that can be translated into simple loops by SMT queries over arithmetic.

Another important approach which also uses counter is [37]. But it contrasts SPEED [6], it uses recurrence equations and introduces a counter for each loop path.

C. Recurrence solving and Cost Analysis

Recurrence solving is also used in [39], [40] [3, 9]. The work presented in [9] derives loop bounds by solving arbitrary C-finite recurrences and deploying quantifier elimination over integers and real closed fields. To this end, [40] uses some algebraic algorithms as black-boxes built upon the computer algebra system Mathematica [23]. Contrarily to [40], we only solve C-finite recurrences of order 1, but, unlike [40], we do not rely on computer algebra systems and handle more complex multi-path loops. Symbolic loop bounds in [39] are inferred over arbitrarily nested loops with polynomial dependencies among loop iteration variables. To this end, C-finite and hypergeometric recurrence solving are used. Unlike [39], we only handle C-finite recurrences of order 1. Contrarily to [39], we, however, design flow refinement techniques to make our approach scalable to the WCET analysis of programs.

References

[1] J. Regehr, A. Reid, and K. Webb, "Eliminating stack overflow by abstract interpretation," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 4, pp. 751–778, Nov. 2005. [Online]. Available: http://doi.acm.org/10.1145/1113830.1113833

[2] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855861

[3] M. Cohen, H. S. Zhu, E. E. Senem, and Y. D. Liu, "Energy types," *SIGPLAN Not.*, vol. 47, no. 10, pp. 831–850, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2398857.2384676

[4] G. Barthe, G. Betarte, J. Campo, C. Luna, and D. Pichardie, "System-level non-interference for constant-time cryptography," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 1267–1279. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660283

[5] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1–17. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04138-9_1

[6] S. Gulwani, K. K. Mehra, and T. Chilimbi, "Speed: Precise and efficient static estimation of program computational complexity," in *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '09. New York, NY, USA: ACM, 2009, pp. 127–139. [Online]. Available: http://doi.acm.org/10.1145/1480881.1480898

[7] M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl, "Analyzing runtime and size complexity of integer programs," *ACM Trans. Program. Lang. Syst.*, vol. 38, no. 4, pp. 13:1–13:50, Aug. 2016. [Online]. Available: http://doi.acm.org/10.1145/2866575

[8] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini, "Cost analysis of object-oriented bytecode programs," *Theor. Comput. Sci.*, vol. 413, no. 1, pp. 142–159, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2011.07.009

[9] C. Alias, A. Darte, P. Feautrier, and L. Gonnord, "Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs," in *Proceedings of the 17th International Conference on Static Analysis*, ser. SAS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 117–133. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882094.1882102

[10] M. Sinn, F. Zuleger, and H. Veith, "A simple and scalable static analysis for bound analysis and amortized complexity analysis," in *Computer Aided Verification*, A. Biere and R. Bloem, Eds. Cham: Springer International Publishing, 2014, pp. 745–761.

[11] J. Hoffmann, K. Aehlig, and M. Hofmann, "Multivariate amortized resource analysis," *ACM Trans. Program. Lang. Syst.*, vol. 34, no. 3, pp. 14:1–14:62, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2362389.2362393

[12] M. Hofmann and S. Jost, "Static prediction of heap space usage for first-order functional programs," in *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '03. New York, NY, USA: ACM, 2003, pp. 185–197. [Online]. Available: http://doi.acm.org/10.1145/604131.604148

[13] ——, "Type-based amortised heap-space analysis," in *Proceedings of the 15th European Conference on Programming Languages and Systems*, ser. ESOP'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 22–37. [Online]. Available: http://dx.doi.org/10.1007/11693024_3

[14] A. Srikanth, B. Sahin, and W. R. Harris, "Complexity verification using guided theorem enumeration," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL 2017. New York, NY, USA: ACM, 2017, pp. 639–652. [Online]. Available: http://doi.acm.org/10.1145/3009837.3009864

[15] Q. Carbonneaux, J. Hoffmann, and Z. Shao, "Compositional certified resource bounds," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15. New York, NY, USA: ACM, 2015, pp. 467–478. [Online]. Available: http://doi.acm.org/10.1145/2737924.2737955

[16] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani, "Automatic predicate abstraction of c programs," in *ACM SIGPLAN Notices*, vol. 36, no. 5. ACM, 2001, pp. 203–213.

[17] K. L. McMillan, "Lazy abstraction with interpolants," in *International Conference on Computer Aided Verification*. Springer, 2006, pp. 123–136.

[18] F. Lin, "A formalization of programs in first-order logic with a discrete linear order," *Artificial Intelligence*, vol. 235, pp. 1 – 25, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437021630011X

[19] P. Rajkhowa and F. Lin, "VIAP - automated system for verifying integer assignment programs with loops," in *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017*, T. Jebelean, V. Negru, D. Petcu, D. Zaharie, T. Ida, and S. M. Watt, Eds. IEEE Computer Society, 2017, pp. 137–144. [Online]. Available: https://doi.org/10.1109/SYNASC.2017.00032

[20] ——, "Extending viap to handle array programs," in *10th Working Conference on Verified Software: Theories, Tools, and Experiments, VSTTE 2018, Oxford, UK, July 18-19*, 2018. [Online]. Available: https://github.com/VerifierIntegerAssignment/sv-comp/blob/master/extending-viap-array.pdf

[21] E. Rodríguez-Carbonell and D. Kapur, "Automatic generation of polynomial loop invariants: Algebraic foundations," in *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*. ACM, 2004, pp. 266–273.

[22] L. Kovacs and T. Jebelean, "Automated generation of loop invariants by recurrence solving in theorema," in *Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC04)*, 2004.

[23] A. Farzan and Z. Kincaid, "Compositional recurrence analysis," in *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design*. FMCAD Inc, 2015, pp. 57–64.

[24] Z. Kincaid, J. Breck, A. F. Boroujeni, and T. Reps, "Compositional recurrence analysis revisited," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2017, pp. 248–262.

[25] Z. Kincaid, J. Cyphert, J. Breck, and T. Reps, "Non-linear reasoning for invariant synthesis," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, p. 54, 2017.

[26] Z. Manna, *Introduction to Mathematical Theory of Computation*. New York, NY, USA: McGraw-Hill, Inc., 1974.

[27] D. Joyner, O. Čertík, A. Meurer, and B. E. Granger, "Open source computer algebra systems: Sympy," *ACM Communications in Computer Algebra*, vol. 45, no. 3/4, pp. 225–234, 2012.

[28] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[29] S. Gulwani and F. Zuleger, "The reachability-bound problem," in *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '10. New York, NY, USA: ACM, 2010, pp. 292–304. [Online]. Available: http://doi.acm.org/10.1145/1806596.1806630

[30] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1347375.1347389

[31] P. Puschner and C. Koza, "Calculating the maximum, execution time of real-time programs," *Real-Time Syst.*, vol. 1, no. 2, pp. 159–176, Sep. 1989. [Online]. Available: http://dx.doi.org/10.1007/BF00571421

[32] N. Zhang, A. Burns, and M. Nicholson, "Pipelined processors and worst case execution times," *Real-Time Systems*, vol. 5, no. 4, pp. 319–343, Oct 1993. [Online]. Available: https://doi.org/10.1007/BF01088834

[33] B. Lisper, "Fully automatic, parametric worst-case execution time analysis," in *Proceedings of the 3rd International Workshop on Worst-Case Execution Time Analysis, WCET 2003 - a Satellite Event to ECRTS 2003, Polytechnic Institute of Porto, Portugal, July 1, 2003*, J. Gustafsson, Ed., vol. MDH-MRTC-116/2003-1-SE. Department of Computer Science and Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden, 2003, pp. 99–102.

[34] M. d. Michiel, A. Bonenfant, H. Cassé, and P. Sainrat, "Static loop bound analysis of c programs based on flow analysis and abstract interpretation," in *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2008, pp. 161–166.

[35] Z. Ammarguellat and W. L. Harrison, III, "Automatic recognition of induction variables and recurrence relations by abstract interpretation," in *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*, ser. PLDI '90. New York, NY, USA: ACM, 1990, pp. 283–295. [Online]. Available: http://doi.acm.org/10.1145/93542.93583

[36] A. Prantl, M. Schordan, and J. Knoop, "Tubound – a conceptually new tool for worst-case execution time analysis."

[37] J. Knoop, L. Kovács, and J. Zwirchmayr, "Symbolic loop bound computation for wcet analysis," in *Proceedings of the 8th International Conference on Perspectives of System Informatics*, ser. PSI'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 227–242. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29709-0_20

[38] E. Albert, P. Arenas, S. Genaim, and G. Puebla, "Closed-form upper bounds in static cost analysis," *J. Autom. Reason.*, vol. 46, no. 2, pp. 161–203, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10817-010-9174-1

[39] R. Blanc, T. A. Henzinger, T. Hottelier, and L. Kovács, "Abc: Algebraic bound computation for loops," in *Logic for Programming, Artificial Intelligence, and Reasoning*, E. M. Clarke and A. Voronkov, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 103–118.

[40] T. A. Henzinger, T. Hottelier, and L. Kovács, "Valigator: A verification tool with bound and invariant generation," in *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, ser. LPAR '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 333–342. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89439-1_24

APPENDIX

A. SPEED 1

The following example is taken from benchmark of SPEED [6].

```
1.  int n; x=0; y=0;
2.  while(1){
3.       if (x < n) {
4.            y=y+1;
5.            x=x+1;
6.       } else if (y > 0)
7.          y=y-1;
8.       else
9.          break;
10.    }
11.  }
```

Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following:

$$
\begin{aligned}
&1.n_1 = n, y_1 = y_7(N_1), x_1 = x_7(N_1)\\
&2.break\_1\_flag1 = break\_1\_flag7({}_N1)\\
&3.\forall n_1, y_7(n_1+1) = ite(x_7(n_1) < n),\\
&\quad ite((y_7(n_1) > 0), (y_7(n_1) - 1), y_7(n_1)))\\
&4.\forall n_1, x_7((n_1+1)) = ite((x_7(n_1) < n),\\
&\quad (x_7(n_1) + 1), x_7(n_1))\\
&5.\forall n_1, break\_1\_flag_7((n_1+1)) = ite((x_7(n_1) < n),\\
&\quad 0, ite((y_7(n_1) > 0), 0, 1))\\
&6.y_7(0) = 0, x_7(0) = 0, break\_1\_flag_7(0) = 0\\
&7.((1 <= 0) \vee (break\_1\_flag7(N_1) \neq 0))\\
&8.\forall n_1.(n_1 < N_1) \rightarrow ((1 > 0)\\
&\qquad \wedge (break\_1\_flag_7(n_1) == 0))
\end{aligned}
$$

By analyzing the equations (7) and (8), we have found that loop terminates only when break excuted. We need to check all possible exution path of this loop end up excuting break. For that we will use case analysis to we find the closed form solution of the conditional recurrences of x_7 and y_7

1) Case 1: When $x_7(0) < n$ holdes, RS will find the following closed form solution for the conditional recurrences of x_7 and y_7

$$
\begin{aligned}
x_7(n_1) &= ite(0 < n_1 \leq C_1, n_1, ite(n_1 \leq C_2, C_1, C_1))\\
y_7(n_1) &= ite(0 < n_1 \leq C_1, 0 - n_1, ite(n_1 \leq C_2,\\
&\qquad n - C_1, C_2 - C_1))
\end{aligned}
$$

We can derived $C_1 = n$ and $C_2 = 2 * n$ by solving following additional axoims

$$\forall n_1.0 \leq n_1 < C_1 \rightarrow n_1 < n$$
$$\neg(C_1 < n)$$
$$\forall n_1.C_1 \leq n_1 < C_2 \rightarrow \neg(C_1 < n) \wedge n_1 - C_1 > 0$$
$$\neg(C_1 < n) \wedge \neg(C_2 - C_1 > 0)$$

From the above equations, We can derives $N_1 = 2*n$ which is the bound $\mathcal{B}^{\vec{X}}_{(case_1)} = 2 * n$

2) Case 2: When holdes $\neg(x_7(0) < n) \wedge \neg(y_7(0) > 0)$, RS will find the following closed form solution for the conditional recurrences of x_7 and y_7 along with additional axoims

$$x_7(n_1) = 0, y_7(n_1) = 0$$

From the above equations, We can derives $N_1 = 1$ which is the bound $\mathcal{B}^{\vec{X}}_{(case_2)} = 1$

Another case $\neg(x_7(0) < n) \wedge (y_7(0) > 0)$ is invalid. Now $\mathcal{B}^{\vec{X}} = max(\mathcal{B}^{\vec{X}}_{(case_1)}, \mathcal{B}^{\vec{X}}_{(case_2)}) = max(1, 2 * n) = 2 * n$

B. SPEED 2

The following example is taken from benchmark of SPEED [6].

```
1. int n, m, va=n, vb=0;
2. while (va > 0) {
3.      if (vb < m) {
4.          vb=vb+1; va=va-1;
5.      } else {
6.          vb=vb-1; vb=0;
7. }
8.}
```

Our translator would be translated to a set of axioms $\Pi^{\vec{X}}_P$ like the following:

$$1.n_1 = n, m_1 = m, va_1 = va_5(N_1), vb_1 = vb_5(N_1)$$
$$3.\forall n_1, va_5((n_1 + 1)) = ite((vb_5(n_1) < m), (va_5(n_1) - 1), va_5(n_1))$$
$$3.\forall n_1, vb_5((n_1 + 1)) = ite((vb_5(n_1) < m), (vb_5(n_1) + 1), 0)$$
$$6.va_5(0) = n, vb_5(0) = 0$$
$$7.\neg(va_5(N_1) > 0)$$
$$8.\forall n_1.(n_1 < N_1) \rightarrow (va_5(n_1) > 0)$$

Then RS find the closed form solution of the conditional recurrences of va_5 and vb_5

$$va_5(n_1) = ite(0 < n_1 \leq C_1, n - n_1, 0)$$
$$vb_5(n_1) = ite(0 < n_1 \leq C_1, n_1, C_1)$$

We can derived $C_1 = m$ by solving following additional axoims

$$\forall n_1.0 \leq n_1 < C_1 \rightarrow n_1 < m$$
$$\neg(C_1 < m)$$

After subsituting $va_5(n_1)$ and $vb_5(n_1)$ and getting rid of $va_5(n_1 + 1)$ and $vb_5(n_1 + 1)$ results the following equations:

$$1.n_1 = n, m_1 = m$$
$$2.va_1 = ite(0 < N_1 \leq m, n - N_1, 0)$$
$$3.vb_1 = ite(0 < N_1 \leq m, N_1, m)$$
$$4.\neg(ite(0 < N_1 \leq m, n - N_1, 0) > 0)$$
$$5.\forall n_1.(n_1 < N_1) \rightarrow (ite(0 < n_1 \leq m, n - n_1, 0) > 0)$$

Now we can derived that $\mathcal{B}^{\vec{X}} = n$ from equation (4) and (5).

C. SPEED 3

The following example is taken from benchmark of SPEED [6].

```
1. int n, m, i = n;
2. while (i > 0) {
3.      if (i < m) {
4.          i = i - 1;
5.      } else {
6.          i = i - m;
7. }
8.}
```

Our translator would be translated to a set of axioms $\Pi^{\vec{X}}_P$ like the following:

$$1.n_1 = n, m_1 = m, i_1 = i_3(N_1)$$
$$2.\forall n_1, i_3((n_1 + 1)) = ite((i_3(n_1) < m), (i_3(n_1) - 1), (i_3(n_1) - m))$$
$$3.i_3(0) = n$$
$$4.\neg(i_3(N_1) > 0)$$
$$5.\forall n_1.(n_1 < N_1) \rightarrow (i_3(n_1) > 0)$$

Then RS find the closed form solution of the conditional recurrence of i_3 using case analysis as follows

1) Case 1: When $i_3(0) < m$ holdes, RS will find the following closed form solution

$$i_3(n_1) = n - n_1$$

After subsituting $i_3(n_1)$ and getting rid of $i_3(n_1+1)$ results the following equations:

$$1.n_1 = n, m_1 = m, i_1 = n - N_1$$
$$2.\neg(n - N_1 > 0)$$
$$3.\forall n_1.(n_1 < N_1) \rightarrow (n - n_1 > 0)$$

Now we can derived that $\mathcal{B}^{\vec{X}}_{case1} = n$ from equation (2) and (3).

2) Case 2: When $neg(i_3(0) < m)$ holdes, RS will find the following closed form solution

$$i_3(n_1) = n - m * n_1$$

After subsituting $i_3(n_1)$ and getting rid of $i_3(n_1+1)$ results the following equations:

$$1.n_1 = n, m_1 = m, i_1 = n - m * N_1$$
$$2.\neg(n - m * N_1 > 0)$$
$$3.\forall n_1.(n_1 < N_1) \rightarrow (n - m * n_1 > 0)$$

Now we can derived that $\mathcal{B}^{\vec{X}}_{case2} = n/m$ from equation (2) and (3).

We can derived that $\mathcal{B}^{\vec{X}} = max(n, n/m) = n$.

D. SPEED 4

The following example is taken from benchmark of SPEED [6].

```
1.  int n, m, dir, i;
2.  assume(0<m);
3.  assume(m<n);
4.  i = m;
5.  while (0 < i && i < n) {
6.        if (dir == 1) i++;
7.        else i--;
8.  }
```

Our translator would be translated to a set of axioms $\Pi^{\vec{X}}_P$ like the following:

$$1.m1 = m, dir1 = dir, n1 = n, i_1 = i_2(N_1)$$
$$2.\forall n_1.i_2((n_1 + 1)) = ite((dir == 1),$$
$$(i_2(n_1) + 1), i_2(n_1) - 1)$$
$$3.i_2(0) = m$$
$$4.\neg(0 < i_2(N_1)) \wedge (i_2(N_1) < n)$$
$$5.\forall n_1.(n_1 < N_1) \rightarrow ((0 < i_2(n_1)) \wedge (i_2(n_1) < n))$$

Then RS find the closed form solution of the conditional recurrence of i_3 using case analysis under the assumptions $0 < m$ and $m < n$ as follows

1) Case 1: When $dir == 1$ holdes, RS will find the following closed form solution

$$i_2(n_1) = m + n_1$$

After subsituting $i_3(n_1)$ and getting rid of $i_3(n_1+1)$ results the following equations:

$$1.n_1 = n, m_1 = m, i_1 = m + N_1$$
$$2.\neg(0 < m + N_1) \wedge (m + N_1 < n)$$
$$3.\forall n_1.(n_1 < N_1) \rightarrow ((0 < m + n_1) \wedge (m + n_1 < n))$$

Now we can derived that $\mathcal{B}^{\vec{X}}_{case1} = n - m$ from equation (2) and (3).

2) Case 2: When $neg(dir == 1)$ holdes, RS will find the following closed form solution

$$i_3(n_1) = m - n_1$$

After subsituting $i_3(n_1)$ and getting rid of $i_3(n_1+1)$ results the following equations:

$$1.n_1 = n, m_1 = m, i_1 = m - N_1$$
$$2.\neg(0 < m - N_1) \wedge (m - N_1 < n)$$
$$3.\forall n_1.(n_1 < N_1) \rightarrow ((0 < m - n_1) \wedge (m - n_1 < n))$$

Now we can derived that $\mathcal{B}^{\vec{X}}_{case2} = m$ from equation (2) and (3).

We can derived that $\mathcal{B}^{\vec{X}} = max(m, n - m)$.

E. SPEED 5

The following example is taken from benchmark of SPEED [6].

```
1.  int n, i=0, j;
2.  while (i < n)
3.  {
4.      j = i + 1;
5.      while (j < n)
6.      {
7.          if (nondet_int() > 0){
8.                j = j - 1; n = n - 1;
9.           }
10.           j = j + 1;
11.     }
12.     i = i + 1;
13.  }
```

Our translator would be translated to a set of axioms $\Pi^{\vec{X}}_P$ like the following:

$$1.i_1 = (N_2 + 0), j_1 = j_7(N_2), n_1 = n_7(N_2)$$
$$2.\forall n_1, n_2.j_4((n_1 + 1), n_2) = (ite((nondet\_int_2(n_1, n_2) > 0),$$
$$(j_4(n_1, n_2) - 1), j_4(n_1, n_2)) + 1)$$
$$3.\forall n_1, n_2.n_4((n_1 + 1), n_2) = ite((nondet\_int_2(n_1, n_2) > 0),$$
$$(n_4(n_1, n_2) - 1), n_4(n_1, n_2))$$
$$4.\forall n_2.j_4(0, n_2) = ((n_2 + 0) + 1)$$
$$5.\forall n_2.n_4(0, n_2) = n_7(n_2)$$
$$6.\forall n_2.\neg(j_4(N_1(n_2), n_2) < n_4(N_1(n_2), n_2))$$
$$7.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (j_4(n_1, n_2) < n_4(n_1, n_2))$$
$$8.\forall n_2.j_7((n_2 + 1)) = j_4(N_1(n_2), n_2)$$
$$9.\forall n_2.n_7((n_2 + 1)) = n_4(N_1(n_2), n_2)$$
$$10.j_7(0) = j, n_7(0) = n$$
$$11.\neg((N_2 + 0) < n_7(N_2))$$
$$12.\forall n_2.(n_2 < N_2) \rightarrow ((n_2 + 0) < n_7(n_2))$$

Then RS find the closed form solution of the conditional recurrence of j_4 and n_4, which is associated with inner loop, using case analysis as follows

1) Case 1: When $\forall n_1, n_2.(nondet\_int_2(n_1, n_2) > 0)$ holdes, RS will find the following closed form solution

$$\forall n_2.j_4(n_1, n_2) = ((n_2 + 0) + 1)$$
$$\forall n_2.n_4(n_1, n_2) = n_7(n_2) - n_1$$

After subsituting $j_4(n_1, n_2)$,$n_4(n_1, n_2)$ and getting rid of $j_4(n_1 + 1, n_2)$,$n_4(n_1 + 1, n_2)$ results the following equations:

$$1.i_1 = (N_2 + 0), j_1 = j_7(N_2), n_1 = n_7(N_2)$$
$$2.\forall n_2.\neg(N_1(n_2) < n_7(n_2) - N_1(n_2))$$
$$3.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < n_7(n_2) - n_1)$$
$$4.\forall n_2.j_7((n_2 + 1)) = ((n_2 + 0) + 1)$$
$$5.\forall n_2.n_7((n_2 + 1)) = n_7(n_2) - N_1(n_2)$$
$$6.j_7(0) = j, n_7(0) = n$$
$$7.\neg((N_2 + 0) < n_7(N_2))$$
$$8.\forall n_2.(n_2 < N_2) \rightarrow ((n_2 + 0) < n_7(n_2))$$

Then RS find the closed form solution of the conditional recurrence of j_4 and n_4, which is associated with inner loop, using case analysis as follows

$$n_7(n_2) = n - \sum_{i=1}^{n_2} N_1(i)$$

After subsituting $n_7(n_2)$ and getting rid of $n_7(n_2+1)$ results the following equations:

$$1.i_1 = (N_2 + 0), j_1 = ite(N_2 == 0, j, (N_2)$$
$$2.n_1 = n - \sum_{i=1}^{N_2} N_1(i)$$
$$3.\forall n_2.\neg(N_1(n_2) < n - \sum_{i=1}^{n_2} N_1(i) - N_1(n_2))$$
$$4.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < n - \sum_{i=1}^{n_2} N_1(i) - n_1)$$
$$5.\neg((N_2 + 0) < n - \sum_{i=1}^{N_2} N_1(i))$$
$$6.\forall n_2.(n_2 < N_2) \rightarrow ((n_2 + 0) < n - \sum_{i=1}^{n_2} N_1(i))$$

2) Case 2: When $\forall n_1, n_2.\neg(nondet\_int_2(n_1, n_2) > 0)$ holdes, RS will find the following closed form solution

$$\forall n_2.j_4(n_1, n_2) = ((n_2 + 0) + 1) + n_1$$
$$\forall n_2.n_4(n_1, n_2) = n_7(n_2) + n_1$$

After subsituting $j_4(n_1, n_2)$,$n_4(n_1, n_2)$ and getting rid of $j_4(n_1 + 1, n_2)$,$n_4(n_1 + 1, n_2)$ results the following equations:

$$1.i_1 = (N_2 + 0), j_1 = j_7(N_2), n_1 = n_7(N_2)$$
$$2.\forall n_2.\neg(N_1(n_2) < n_7(n_2) + N_1(n_2))$$
$$3.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < n_7(n_2) + n_1)$$
$$4.\forall n_2.j_7((n_2 + 1)) = ((n_2 + 0) + 1) - N_1(n_2)$$
$$5.\forall n_2.n_7((n_2 + 1)) = n_7(n_2) + N_1(n_2)$$
$$6.j_7(0) = j, n_7(0) = n$$
$$7.\neg((N_2 + 0) < n_7(N_2))$$
$$8.\forall n_2.(n_2 < N_2) \rightarrow ((n_2 + 0) < n_7(n_2))$$

Then RS find the closed form solution of the conditional recurrence of j_4 and n_4, which is associated with inner loop, using case analysis as follows

$$n_7(n_2) = n + \sum_{i=1}^{n_2} N_1(i)$$

After subsituting $n_7(n_2)$ and getting rid of $n_7(n_2+1)$ results the following equations:

$$1.i_1 = (N_2 + 0), j_1 = ite(N_2 == 0, j, (N_2)$$
$$2.n_1 = n + \sum_{i=1}^{N_2} N_1(i)$$
$$3.\forall n_2.\neg(N_1(n_2) < n - \sum_{i=1}^{n_2} N_1(i) + N_1(n_2))$$
$$4.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow (n_1 < n - \sum_{i=1}^{n_2} N_1(i) + n_1)$$
$$5.\neg((N_2 + 0) < n + \sum_{i=1}^{N_2} N_1(i))$$
$$6.\forall n_2.(n_2 < N_2) \rightarrow ((n_2 + 0) < n + \sum_{i=1}^{n_2} N_1(i))$$

F. SPEED 6

The following example is taken from benchmark of SPEED [6].

```
1.  int n;
2.  while (n>0)
3.  {
4.      n=n-1;
5.      while (n>0)
6.      {
7.          if (__VERIFIER_nondet_int())
                    break;
8.          n=n-1;
9.      }
10.  }
```

Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following:

$1.break\_1\_flag_1 = break\_1\_flag_7(N_2), n_1 = n_7(N_2)$
$2.\forall n_1, n_2.n_5((n_1+1), n_2) =$
$ite((ite((nondet\_int_2(n_1, n_2) > 0), 1, 0) == 0),$
$(n_5(n_1, n_2) - 1), n_5(n_1, n_2))$
$3.\forall n_1, n_2.break\_1\_flag_5((n_1+1), n_2) =$
$ite((nondet_int_2(n_1, n_2) > 0), 1, 0)$
$4.\forall n_2.break\_1\_flag_5(0, n_2) = break\_1\_flag_7(n_2)$
$5.\forall n_2.n_5(0, n_2) = (n_7(n_2) - 1)$
$6.\forall n_2.\neg((n_5(< N_1(n_2), n_2) > 0)$
$\wedge(break\_1\_flag_5(< N_1(n_2), n_2) == 0))$
$7.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow ((n_5(n_1, n_2) > 0)$
$\wedge(break\_1\_flag_5(n_1, n_2) == 0))$
$8.\forall n_2.break\_1\_flag_7((n_2+1)) = break\_1\_flag_5(N_1(n_2), n_2)$
$9.\forall n_2.n_7((n_2+1)) = n_5(N_1(n_2), n_2)$
$10.break\_1\_flag_7(0) = 0, n_7(0) = n$
$11.\neg(n_7(N_2) > 0)$
$12.\forall n_2.(n_2 < N_2) \rightarrow (n_7(n_2) > 0)$

By analyzing the equations (6) and (7), we have found that loop terminates only when break excuted. We need to check all possible exution path of this loop end up excuting break. For that we will use case analysis to we find the closed form solution of the conditional recurrences of n_5 and n_7

1) Case 1: When break never excute which means $\forall n_1, n_2.\neg(nondet\_int_2(n_1, n_2) > 0)$, then we tried to find the closed form solution of the following recurrence

$$n_5((n_1+1), n_2) = (n_5(n_1, n_2) - 1)$$

Then RS find the closed form solution of the recurrence of n_5, which is associated with inner loop, using case analysis. After subsituting $n_5(n_1, n_2) = (n_7(n_2) - 1) - n_1$ and getting rid of $n_5((n_1+1), n_2)$ results the following equations:

$1.break\_1\_flag_1 = break\_1\_flag_7(N_2), n_1 = n_7(N_2)$
$2.\forall n_2.\neg(((n_7(n_2) - 1) - N_1(n_2) > 0) \wedge (0 == 0))$
$3.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow ((n_7(n_2) - 1) - N_1(n_2) > 0)$
$\wedge(0 == 0))$
$4.\forall n_2.break\_1\_flag_7((n_2+1)) = 0$
$5.\forall n_2.n_7((n_2+1)) = (n_7(n_2) - 1) - N_1(n_2)$
$6.break\_1\_flag_7(0) = 0, n_7(0) = n$
$7.\neg(n_7(N_2) > 0)$
$8.\forall n_2.(n_2 < N_2) \rightarrow (n_7(n_2) > 0)$

Then RS find the closed form solution of the recurrence of n_7, which is associated with inner loop, using case analysis. After subsituting $n_7(n_2) = n - \sum_{i=1}^{n_2} N_1(i) - n_2$ and getting rid of $n_7(n_2)$ results the following equations:

$1.break\_1\_flag_1 = break\_1\_flag_7(N_2), n_1 = n_7(N_2)$
$2.\forall n_2.\neg(((n - \sum_{i=1}^{n_2} N_1(i) - n_2 - 1)$
$-N_1(n_2) > 0) \wedge (0 == 0))$
$3.\forall n_1, n_2.(n_1 < N_1(n_2)) \rightarrow ((n - \sum_{i=1}^{n_2} N_1(i) - n_2 - 1)$
$-N_1(n_2) > 0) \wedge (0 == 0))$
$4.\neg(n - \sum_{i=1}^{N_2} N_1(i) - N_2 > 0)$
$5.\forall n_2.(n_2 < N_2) \rightarrow (n - \sum_{i=1}^{n_2} N_1(i) - n_2 > 0)$

G. SPEED 7

The following example is taken from benchmark of SPEED [6].

```
1.  int x,y,n,m;
2.  while (n > x)
3.  {
4.     if (m > y) {y = y + 1;}
5.     else { x = x + 1; }
6.  }
```

Our translator would be translated to a set of axioms $\Pi_P^{\vec{X}}$ like the following:

$1.m_1 = m, n_1 = n, y_1 = y_3(N_1), x_1 = x_3(N_1)$
$2.\forall n_1.y_3((n_1+1)) = ite((m > y_3(n_1)),$
$(y_3(n_1) + 1), y_3(n_1))$
$3.\forall n_1.x_3((n_1+1)) = ite((m > y_3(n_1)),$
$x_3(n_1), (x_3(n_1) + 1))$
$10.y_3(0) = y, x_3(0) = x$
$11.\neg(n > x_3(N_1))$
$12.\forall n_1.(n_1 < N_1) \rightarrow (n > x_3(n_1))$

Then RS find the closed form solution of the conditional recurrence of x_3 and y_3 using case analysis as follows

1) Case 1: When $(m > y_3(0)$ holdes, RS will find the following closed form solution

$$y_3(n_1) = ite(0 \leq n_1 \leq C_1, y + n_1, y + C_1)$$
$$x_3(n_1) = ite(0 \leq n_1 \leq C_1, x, x + n_1)$$

We can derived $C_1 = m$ by solving following additional axoims

$$\forall n_1.0 \leq n_1 < C_1 \rightarrow m > n_1$$
$$\neg(m > C_1)$$

After subsituting $x_3(n_1)$ and $y_3(n_1)$ and getting rid of $x_3(n_1+1)$ and $y_3(n_1+1)$ results the following equations:

$$
\begin{aligned}
&1.m_1 = m, n_1 = n,\\
&2.y_1 = ite(0 \leq N_1 \leq m, y + N_1, y + m)\\
&3.x_1 = ite(0 \leq N_1 \leq m, x, x + N_1)\\
&4.\neg(n > ite(0 \leq n_1 \leq m, x, x + N_1))\\
&5.\forall n_1.(n_1 < N_1) \rightarrow (n > ite(0 \leq n_1 \leq m, x, x + n_1))
\end{aligned}
$$

Now we can derived that $\mathcal{B}^{\vec{X}}_{Case_1} = n - x$ from equation (4) and (5).

2) Case 2: When $\neg(m > y_3(0)$ holdes, RS will find the following closed form solution

$$
\begin{aligned}
y_3(n_1) &= y\\
x_3(n_1) &= x + n_1
\end{aligned}
$$

After subsituting $x_3(n_1)$ and $y_3(n_1)$ and getting rid of $x_3(n_1 + 1)$ and $y_3(n_1 + 1)$ results the following equations:

$$
\begin{aligned}
&1.m_1 = m, n_1 = n,\\
&2.y_1 = y\\
&3.x_1 = x + N_1\\
&4.\neg(n > x + N_1)\\
&5.\forall n_1.(n_1 < N_1) \rightarrow n > x + n_1
\end{aligned}
$$

Now we can derived that $\mathcal{B}^{\vec{X}}_{Case_2} = n - x$ from equation (4) and (5).