A Recurrence Solver And Its Uses in Program Verification

Pritom Rajkhowa and Fangzhen Lin

Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong,
{prajkhowa,flin}@cse.ust.hk

Abstract. This paper describes a system for computing closed form solution of recurrences often encountered in computer program verification. It uses SymPy's rsolve() function as a base solver, and extends it to handle conditional recurrences. It also includes a solution for a simple case of mutual recurrences. To show the effectiveness of this recurrence solver, we added it to our automated program verification system VIAP. As a result, the new verification system can now verify many programs that were previously out of reach for VIAP. As an application, we use the new system as a program equivalence checker, and tried it on the benchmarks from two recent specialized equivalence checker REVE. Our experiments show that on REVE benchmarks, our system performs as well as these two state-of-art systems. On other benchmarks from ICRA and NLA, our system outperforms REVE.

Keywords: Automatic Program Verification, First-Order Logic, Mathematical Induction, Recurrences, SMT, Arithmetic

1 Introduction

A recurrence is a recursive definition of a function in terms of itself. This includes inductively defined functions such as

$$f(n+1) = 2f(n) + 1,\ f(0) = 0$$

mutually recursive functions such as

$$f(n+1) = f(n) + g(n) + 1,\ g(n+1) = f(n) + g(n) + 2,$$

$$f(0) = 0,\ g(0) = 0$$

or what we call conditional recurrences such as

$$f(n) = \begin{cases} 0, & \text{if } n = 0 \\ f(n-1) + 2, & \text{if } x > y \\ f(n-1) + 3, & \text{if } x \leq y \end{cases}$$

where x and y are two constants.

Reasoning about recurrences, including computing their closed-form solutions if they exist, is an important problem in discrete mathematics and computer science. Tools such as Maple [1], Sympy [2] and Mathematica [3] come with built-in functions to compute the closed-form solutions of certain recurrences. Petkovsek *et. al* [4] provided a comprehensive account of the types of recurrences that have known closed-form solutions and the algorithms for computing them. Much of this had been motivated by work in theoretical computer science, especially analysis of algorithms.

In software engineering, recurrences have long been used to prove properties about recursive programs and programs with loops, especially for deriving loop invariants, see e.g. [5,6,7,8,9].

Our motivation for this work also came from formal program verification. We have recently constructed a fully automated program verification system called VIAP [10,11], for programs with integer assignments and arrays. VIAP is based on a translation from programs to first-order logic with quantifiers over natural numbers by Lin [12]. In this translation, loops are translated to recurrences. For example, the following program

```
int x=0,y=0;
while (x<100) {
  if (x < 50) y++ else y--
  x++
}
```

would be translated to a set of axioms like the following:

$$
\begin{aligned}
&x_1 = x_2(N) \wedge y_1 = y_2(N),\\
&x_2(0) = 0,\\
&\forall n.x_2(n+1) = x_2(n) + 1,\\
&y_2(0) = 0,\\
&\forall n.y_2(n+1) = ite(x_2(n) < 50, y_2(n) + 1, y_2(n) - 1),\\
&\neg(x_2(N) < 100),\\
&\forall n.n < N \rightarrow x_2(n) < 100,
\end{aligned}
$$

where x_1 and y_1 denote the output values of x and y, respectively, $x_2(n)$ and $y_2(n)$ the values of x and y during the nth iteration of the loop, respectively. 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.

There are two recurrences in the above axioms. The one for $x_2(n)$ can be solved using the rsolve() function in SymPy, and this yields the closed-form solution $x_2(n) = n$ which can then be used to simplify the recurrence for $y_2(n)$ into

$$y_2(0) = 0,\ y_2(n+1) = ite(n < 50, y_2(n) + 1, y_2(n) - 1),$$

which is a conditional recurrence cannot be dealt by existing tools, to the best of our knowledge.

This motivated us to design a system for computing closed-form solutions of recurrences that can deal with conditional ones like the above one for $y_2(n)$. Our recurrence solver uses rsolve() from SymPy as the base solver for non-conditional recurrences like the above one for $x2(n)$. It also depends on the SMT solver Z3 [13] to perform simplifications and for computing the truth value of a condition. It can solve certain types of conditional recurrences including the one above, for which it returns the following closed-form solution:

$$y_2(n) = ite(0 \leq n < 50, n, 100 - n).$$

Once the recurrence solver is added to our verification system, it can solve many more benchmarks that were previously out of its reach. As an application of our new system, we apply it to the problem of checking the equivalence of two programs, by reducing the problem to that of verifying a safety property of the sequential composition of the two programs, similar to the approach taken in [14,15]. We compare our equivalence checking system to REVE [16] the state-of-art equivalence checker. Our experiments show that on REVE benchmarks, our system performs REVE. On other benchmark problems from ICRA and NLA, our system outperforms REVE. The performance of our verification system with the recurrence solver added clearly shows its effectiveness. We hope it can also be used in other systems. Here, a simple example to provide an intuitive description how approach out perform some state of art autmatic verifer. The C code snippet is from HOLA benchmark used in [17].

```
int x = 0, y = 0;
int i = 0, j = 0;

while(x <= LINT
     && y <= LINT
        && nondet_int())
{
   while(nondet_int())
   {
     if(x==y) { i++;}
     else { j++; }
   }
   if(i>=j){ x++; y++;}
   else { y++;}
}

assert(i>=j);
```

would be translated to a set of axioms like the following:

$$
\begin{aligned}
&LINT_1 = LINT \wedge i_1 = i_6(N_2) \wedge y_1 = y_6(N_2) \wedge \\
&j_1 = 0 \wedge x_1 = x_6(N_2), \\
&\forall n_1, n_2.i_2(n_1 + 1, n_2) = \\
&\qquad ite(x_6(n_2) == y_6(n_2), (i_2(n_1, n_2) + 1) \\
&\qquad\qquad , i_2(n_1, n_2)), \\
&\forall n_2.i_2(0, n_2) = i_6(n_2), \\
&\forall n_1, n_2.\neg(nondet\_int_2(n_1, n_2) > 0),
\end{aligned}
$$

$$
\begin{aligned}
&\forall n_1, n_2.n_1 < N_1(n_2) \rightarrow (nondet\_int_2(n_1, n_2) > 0), \\
&\forall n_2.i_6(n_2 + 1) = i_2(N_1(n_2), n_2), \\
&\forall n_2.y_6(n_2 + 1) = ite(i_2(N_1(n_2), n_2) \geq 0, y_6(n_2) + 1 \\
&\qquad\qquad , y_6(n_2)) \\
&\forall n_2.x_6(n_2 + 1) = ite(i_2(N_1(n_2), n_2) \geq 0, x_6(n_2) + 1 \\
&\qquad\qquad , x_6(n_2)) \\
&i_6(0) = 0, y_6(0) = 0, x_6(0) = 0 \\
&\neg(x_2(N) < 100), \\
&\forall n_2.n_2 < N_2 \rightarrow ((x_6(n_2) <= LINT) \\
&\qquad\qquad \wedge (y_6(n_2) <= LINT)) \\
&\qquad\qquad\qquad \wedge (nondet\_int_3(n_2) > 0)
\end{aligned}
$$

where $LINT_1$, i_1, j_1, x_1 and y_1 denote the output values of $LINT$, i, j, x and y, respectively, $i_2(n_1, n_2)$ the value of i at n_1th iteration of the inter loop during n_2th iteration of the outer loop. $i_6(n_2)$, $x_6(n_2)$ and $y_6(n_2)$ the values of x and y during the n_2th iteration of the loop, respectively. Also N_2 is a system generated natural number constant denoting the number of iterations that the outer loop runs before exiting; N_1 is a system generated natural number function denoting that for each n_2, $N_1(n_2)$ is the number of iterations that the inner loop runs during the n_2th iteration of the outer loop. There are three recurrences in the above axioms and all are non-conditional. Our desgined system tried to find the closed form solutions of these conditional equations. Our system first tried to simplify equations by successfully proving following the condition of recurrence $i_2(n_2, n_1)$ always hols using SMT solver (Z3)

$$\forall n_2.x_6(n_2) == y_6(n_2)$$

Then simplifed the condition of recurrence $i_2(n_1 + 1, n_2) = i_2(n_1, n_2) + 1$ can be solved using the rsolve() function in SymPy with reprect to initial value $i_2(0, n_2) = i_6(n_2)$ and this yields the closed-form solution $i_2(n_1, n_2) = i_6(n_2)+n_1$ which can then be used to simplify the recurrence for $y_2(n)$ into The one for $x_2(n)$

can be solved using the rsolve() function in SymPy, and this yields the closed-form solution $x_2(n) = n$ which can then be used to simplify the recurrence for $x_6(n_1)$,$y_6(n_1)$ and $i_6(n_1)$ into

$$\begin{aligned}
&\forall n_2.i_6(n_2+1) = i_6(n_2) + N_1(n_2),\\
&\forall n_2.y_6(n_2+1) = ite(i_6(n_2) + N_1(n_2) \geq 0, y_6(n_2) + 1\\
&\qquad\qquad\qquad\qquad , y_6(n_2))\\
&\forall n_2.x_6(n_2+1) = ite(i_2(N_1(n_2), n_2) \geq 0, x_6(n_2) + 1\\
&\qquad\qquad\qquad\qquad , x_6(n_2))\\
&i_6(0) = 0, y_6(0) = 0, x_6(0) = 0
\end{aligned}$$

For the above program the assertion to be proved is

$$i_6(N_2) >= 0.$$

Our system then try to prove it by proving the following more general one

$$\forall n_2.i_6(n_2) >= 0$$

by induction on n_2, which was successful. Tools like ICRA[17] VeriAbs [18], UAutomizer [19], Seahorn [20], and CPAChecker [21] failed prove the assertion.

2 Recurrence Solver (RS)

Our recurrence solver RS takes a set of recurrences and other constraints, returns a set of closed-form solutions it found for some of the recurrences and the remaining recurrences and constraints simplified using the computed closed-form solutions. We will not give a formal definition of recurrences and constraints here. Informally recurrences are just axioms in a formal language like first-order logic. This informal description should be enough to understand the ideas and algorithms used in our solver.

Algorithm (1a) below outlines our recurrence solver. It makes use of three sub-solvers and two helper functions:

- A base non-conditional recurrence solver ($NCRS$, Algorithm (1c)) that calls rsolve() in SymPy to try to compute closed-form solutions of non-conditional recurrences: given a set Π of recurrences and other constraints, $NCRS$ (Π) returns two sets C, Δ, and a new Π, where C is the set of closed-form solutions computed, Δ the set of non-conditional recurrences that it tried but failed to find a closed-form solution, and Π the set of remaining recurrences.
- A mutual recurrence solver (MRS, Algorithm (1b)) for computing the closed-form solution of a specific type of mutual recurrences: given a set Π of recurrences, MRS (Π) returns three sets C, Δ, and a new Π, where C is the set of closed-form solutions computed, Δ the set of mutual recurrences that it tried but failed to find a closed-form solution, and Π the set of remaining recurrences.

– A conditional recurrence solver (CRS, Algorithm (1d)) for computing the closed-form solution of several classes of conditional recurrences: given a set Π of recurrences, CRS (Π) returns three sets C, Δ, and a new Π, where C is the set of closed-form solutions computed, Δ the set of conditional recurrences that it tried but failed to find a closed-form solution, and Π the set of remaining recurrences.
– A simplifier (SC) simplify conditional recurrences: given a set Π of recurrences and $SC(\Pi)$ returns a new simplified Π. SC uses SMT solver to simplify a condition.

2.1 The Non-Conditional Recurrence Solver ($NCRS$)

This sub-solver basically called SymPy's rsolve() function to compute two types of non-conditional recurrences. The pseudo code is given as Algorithm (1c) which makes use of the following functions:

– $selectNC(\Pi)$ - it returns a recurrence $\langle \sigma, \sigma_0, n \rangle$ from Π, where σ is the recurence, σ_0 the initial condition, and n the variable involved in the recurrence. The selected σ must be non-conditional of the form of either

$$X(n+1) = f(X(n), n),$$

where $f(x, y)$ is a polynomial function of x and y
or

$$X(n+1) = X(n) + f(n) + A_1 F_1(n) + \cdots + A_k F_k(n),$$

where $f(n)$ is a polynomial function in n, A_i's are constants, and F_i's are function symbols.
– $rsolve(\langle \sigma, \sigma_0, n \rangle)$ calls SymPy's recurrence solving function after converting from our language to SmpPy's. SymPy [2] can find the closed-form solution of recurrence equation of type C-finite and Gosper-summable recurrence.
– SR simplifies both Π and Δ using the set computed closed-form solution C. This function is described in more details later.

2.2 The Mutual Recurrence Solver (MRS)

The type of mutually recursive functions that our system can solve is defined by the following selection function:

$$selectMR(\Pi) = \langle \boldsymbol{\sigma}, \boldsymbol{\sigma_0}, n \rangle,$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma_0}$ are sets of equations in Π of the form: for some $h > 1$, $\boldsymbol{\sigma}$ consists of

$$X_i(n+1) = A * (X_1(n) +$$

Input: Π – a set of recurrences and constraints
Output: C – the set of closed-form solutions computed; Π – the remaining set of recurrences and constraints simplified using C; Δ – the set of selected but unsolved recurrence equations

$C \leftarrow \emptyset$;
$\Delta \leftarrow \emptyset$;
while *true* **do**
  $C_1, \Delta_1, \Pi \leftarrow NCRS(\Pi)$;
  $C_2, \Delta_2, \Pi \leftarrow MRS(\Pi)$;
  $C_3, \Delta_3, \Pi \leftarrow CRS(\Pi)$;
  $C' \leftarrow C_1 \cup C_2 \cup C_3$;
  $\Delta' \leftarrow \Delta_1 \cup \Delta_2 \cup \Delta_3$;
  if $C' == \emptyset$ *and* $\Delta' == \emptyset$ **then**
    return $C,\ \Pi \cup \Delta$;
  else
    $C \leftarrow C \cup C'$;
    $\Delta \leftarrow \Delta \cup \Delta'$;
  end
end

(a) Recurrence Solver (RS)

Input: Π – a set of recurrences and constraints
Output: C – the set of closed-form solutions computed; Δ – the set of selected but unsolved recurrence equations; Π – the remaining set of recurrences and constraints simplified using C;

$C \leftarrow \emptyset$;
$\Delta \leftarrow \emptyset$;
$\langle \sigma, \sigma_0, n \rangle = selectNC(\Pi)$;
while $\langle \sigma, \sigma_0, n \rangle \neq \emptyset$ **do**
  $f_\sigma \leftarrow rsolve(\sigma, \sigma_0, n)$;
  if $f_\sigma \neq \emptyset$ **then**
    $C \leftarrow C \cup f_\sigma$;
    $\Pi \leftarrow \Pi - \{\sigma, \sigma_0\}$;
    $\Pi, \Delta \leftarrow SR(f_\sigma, \Pi, \Delta)$;
  else
    $\Delta \leftarrow \Delta \cup \{\sigma, \sigma_0\}$;
  end
  $\langle \sigma, \sigma_0, n \rangle = selectNC(\Pi)$;
end
return $C,\ \Delta,\ \Pi$

(c) The Non-Conditional Recurrence Solver ($NCRS$)

Input: Π – a set of recurrences and other constraints
Output: C – the set of closed-form solutions computed; Δ – the set of selected but unsolved recurrence equations; Π – the remaining set of recurrences and constraints simplified using C

$\mathcal{C} \leftarrow \emptyset$;
$\Delta \leftarrow \emptyset$;
$\langle \boldsymbol{\sigma}, \boldsymbol{\sigma_0}, n \rangle = selectMR(\Pi)$;
while $\langle \boldsymbol{\sigma}, \boldsymbol{\sigma_0}, n \rangle \neq \emptyset$ **do**
  $\boldsymbol{f_\sigma} \leftarrow \mathcal{M}(\boldsymbol{\sigma}, \boldsymbol{\sigma_0}, n)$;
  if $\boldsymbol{f_\sigma} \neq \emptyset$ **then**
    $C \leftarrow C \cup \boldsymbol{f_\sigma}$;
    $\Pi \leftarrow \Pi - \boldsymbol{\sigma} - \boldsymbol{\sigma_0}$;
    $\Pi, \Delta \leftarrow SR(\boldsymbol{f_\sigma}, \Pi, \Delta)$;
  else
    $\Delta \leftarrow \Delta \cup \boldsymbol{\sigma} \cup \boldsymbol{\sigma_0}$;
  end
  $\langle \boldsymbol{\sigma}, \boldsymbol{\sigma_0}, n \rangle = selectMR(\Pi)$;
end
return C, Δ, Π;

(b) Mutual Recurrence Solver (MRS)

Input: Π – a set of recurrences and other constraints.
Output: C – the set of closed-form solutions computed; Δ – the set of selected but unsolved recurrence equations; Π – the remaining set of recurrences and constraints simplified using C, as well as possibly some new constraints added.

$\mathcal{C} \leftarrow \emptyset$;
$\Delta \leftarrow \emptyset$;
$\langle \sigma, \sigma_0, n \rangle = selectCR(\Pi)$;
while $\langle \sigma, \sigma_0, n \rangle \neq \emptyset$ **do**
  $f_\sigma, \boldsymbol{\Phi} \leftarrow \mathcal{C}(\sigma, \sigma_0, n)$;
  if $f_\sigma \neq \emptyset$ **then**
    $C \leftarrow C \cup f_\sigma$;
    $\Pi \leftarrow \Pi \cup \boldsymbol{\Phi}$;
    $\Pi \leftarrow \Pi - \{\sigma, \sigma_0\}$;
    $\Pi, \Delta \leftarrow SR(f_\sigma, \Pi, \Delta)$;
  else
    $\Delta \leftarrow \Delta \cup \{\sigma, \sigma_0\}$;
  end
  $\langle \sigma, \sigma_0, n \rangle = selectCR(\Pi)$;
end
return C, Δ, Π;

(d) Conditional Recurrence Solver (CRS)

$$... + X_h(n)) + C_i, \quad \text{for } 1 \leq i \leq h, \tag{1}$$

where A and C_i are constants, and $\boldsymbol{\sigma_0}$ consists of

$$X_i(0) = E_i, \quad \text{for } 1 \leq i \leq h, \tag{2}$$

where each E_i is a constant expression.

Algorithm (1b) below outlines our solver for these mutually recursive functions. The function $\mathcal{M}$ in the algorithm works as follows: given the input recurrences (1), it generates the following closed-form solutions for $X_1, ..., X_h$:

$$\begin{aligned} X_i(n) = ite(&n = 0, E_i, \\ &ite(n = 1, A(E_1 + .. + E_h) + C_i, \\ &\quad A^n h^n (E1 + ... + E_h) \\ &\qquad +(C_1 + .. + C_h)\frac{A(1 - A^n h^n)}{1 - A * h} + C_i)) \end{aligned} \tag{3}$$

The proof of correctness of the proposed closed-form solution is presented in Proposition (1).

Proposition 1. *Equation (3) represents closed-form solution of the set of mutually recurrence equations (1).*

Proof. We tried to prove that equation (3) represents closed-form solution of the set of mutually recurrence equations (1) using induction.

- **Base Case:** The base case when $n = 0$ and $n = 1$ can be proved easily.
- **Induction Hypothesis:** Assume that the following equations holds when $n = k$ such that $k > 1$ for all $i \in [1, h]$

$$X_i(k) = (A^k * h^k * (E1 + ... + E_h))$$

$$+((C_1 + .. + C_h) * \frac{A(1 - A^k * h^k)}{1 - A * h}) + C_i \tag{4}$$

- **Inductive Step:** We prove that the equations also hold when $n = k + 1$ using the Induction Hypothesis above. From the equations (1) we can have following:

$$X_i(k + 1) = A * (X_1(k) + ... + X_h(k)) + C_i$$

After substituting corresponding equations from (4) Induction Hypothesis.

$$\begin{aligned} &= A * (((A^k * h^k * (E1 + ... + E_h)) \\ &\quad +((C_1 + .. + C_h) * \frac{A(1 - A^k * h^k)}{1 - A * h}) + C_1) + ... \\ &\qquad +((A^k * h^k * (E1 + ... + E_h)) \end{aligned}$$

$$
\begin{aligned}
&+((C_1 + .. + C_h) * \frac{A(1 - A^k * h^k)}{1 - A * h}) + C_h)) + C_i \\
&= A * ((A^k * h^{k+1} * (E1 + ... + E_h)) \\
&\quad +((C_1 + .. + C_h) * \frac{A * h * (1 - A^k * h^k)}{1 - A * h}) + (C_1 + .. + C_h) + C_i) \\
&= A * ((A^k * h^{k+1} * (E1 + ... + E_h)) \\
&\quad +((C_1 + .. + C_h) * (\frac{A * h * (1 - A^k * h^k)}{1 - A * h} + 1))) + C_i \\
&= A * ((A^k * h^{k+1} * (E1 + ... + E_h)) \\
&\quad +((C_1 + .. + C_h) * (\frac{(1 - A^{k+1} * h^{k+1})}{1 - A * h}))) + C_i \\
&= (A^{k+1} * h^{k+1} * (E1 + ... + E_h)) \\
&\quad +((C_1 + .. + C_h) * (\frac{A * (1 - A^{k+1} * h^{k+1})}{1 - A * h})) + C_i
\end{aligned}
$$

Hence, for all value of $n \geq 0$ the closed-form solutions of the equations 1 holds.

It is possible to compute closed-form solutions of other mutually recursive equations. For example, consider the following:

$$
\begin{aligned}
f(n+1) &= 2f(n) + 3g(n), \quad f(0) = 1, \\
g(n+1) &= 5f(n) + 6g(n), \quad g(0) = 2.
\end{aligned}
$$

The recurrences can be manipulated to yield

$$
f(n+2) = 8f(n+1) + 3f(n), \;\; f(1) = 8, \;\; f(0) = 1.
$$

These second order recurrences can be solved using SymPy [2] as it has implemented algorithms from Abramov, Bronstein and Petkovsek [22] [23]. We leave this as a future work.

2.3 Conditional Recurrence (CRS)

We now consider our conditional recurrence solver $CRS(\Pi)$. A conditional recurrence here is one that is returned by the following seletion function:

$$
selectCR(\Pi) = \langle \sigma, X(0) = E, n \rangle,
$$

where E is a constant expression and σ is a conditional equation in Π of the following form:

$$
\begin{aligned}
X(n+1) = ite(\theta_1, f_1(X(n), n), ite(\theta_2, f_2(X(n), n)..., \\
ite(\theta_h, f_h(X(n), n), f_{h+1}(X(n), n)))
\end{aligned} \tag{5}
$$

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.

Algorithm (1d) below outlines our solver $CRS(\Pi)$ for conditional recurences. It relies on the function $\mathcal{C}(\sigma, \sigma_0, n)$ which returns f_σ where f_σ is either a closed-form solution for $X(n)$ or when it fails to find the closed solution. $\mathcal{C}(\sigma, \sigma_0, n)$ considers following five cases of σ.

Type 1 A conditional recurrence σ of the form (5) is called of type 1 if all the conditions θ_i, $1 \leq i \leq h$, in it are independent of the recurrence variable n. Given such a type 1 conditional recurrence σ, $\mathcal{C}(\sigma, X(0) = E, n)$ works as follows:

For each $i \in [1, h+1]$, introduce a new function $g_i(n)$ and call $rsolve(g_i(n+1) = f_i(g_i(n), n), g_i(0) = E, n)$. If one of them does not return a closed-form solution, then return $\emptyset$ for $\mathcal{C}(\sigma, X(0) = E, n)$. Otherwise, suppose $g_i(n) = E_i(n)$ is the returned closed-form solution for g_i, then return f_σ, where f_σ is

$$X(n) = ite(\theta_1, E_1(n), ite(\theta_2, E_2(n), \\ ...ite(\theta_h, E_h(n), E_{h+1}(n))..) \tag{6}$$

The proof of correctness of this closed-form solution is presented in Proposition (2).

Proposition 2. *Equation (6) represents closed-form solution of the conditional recurrence (5) when it is conditional recurrence is type 1.*

Proof. Base Case: when $n = 0$ is very trivial. For the induction step you want to assume that $k > 0$,

$$X(k) = ite(\theta_1, \mathbf{E}_1(k), ite(\theta_2, \mathbf{E}_2(k), ...ite(\theta_h, \mathbf{E}_h(k), \mathbf{E}_{h+1}(k))..)$$

Now, we tried to show the following:

$$X(k+1) = ite(k = 0, E, ite(\theta_1, \mathbf{E}_1(k+1), ite(\theta_2, \mathbf{E}_2(k+1), ... \\ ite(\theta_h, \mathbf{E}_h(k+1), \mathbf{E}_{h+1}(k+1))..)$$

using conditional recurrence (5).

- Case1: When θ_1 is true, we have

$$X(k+1) = f_1(X(k)) = f_1(X(k)/\mathbf{E}_1(k)) = \mathbf{E}_1(k+1)$$

  As $\mathbf{E}_1$ is the closed form solution of $X(n+1) = f_1(X(n))$
- Similarly other cases can be derived.

So the assumption is true.

Example 21

$$j_6(n_1 + 1) = ite(nondet\_int_2 > 0, \\ j_6(n_1) + (n_1 + 1) + 1, j_6(n_1) + (n_1) + 1) \\ j_6(0) = 0$$

The above conditional recurrence of j_6 satisfies the both the condition of type 1. $\mathcal{C}$ return the following closed-form solution for the conditional recurrence of j_6 :

$$j_6(n_1) = ite(nondet\_int_2 > 0, n_1 * (n_1 + 3)/2, n_1 * (n_1 + 2)/2)$$

Type 2 A conditional recurrence σ of the form (7) is called type 2

$$X(n+1) = ite(n == c_1, C_1, ite(n == c_2, C_2..., \\ ite(n == c_h, C_h, f(X(n), n))) \tag{7}$$

where $c_1, c_2, ...c_h$ and $C_1, C_2, .., C_h$ are integers constants. Given such a type 2 conditional recurrence σ, $\mathcal{C}(\sigma, X(0) = E, n)$ works as follows:

Construct a map $M = \{X(0) : E, X(e_1) : E_1, ..., X(e_h) : E_h)\}$ which contain a entry of the form $X(e_i) = E_i$ for each condition $n == e_i$ and introduce a new function $g(n)$. Then call $rsolve(g(n+1) = f(g(n), n), M, n)$. If the function call failed to find the closed form solution of g, then return $\emptyset$ for $\mathcal{C}(\sigma, X(0) = E, n)$. Otherwise, suppose $g(n) = E(n)$ is the returned closed-form solution for g, then return f_σ, where f_σ is $g(n) = E(n)$.

Example 22

$$f_1(n_1 + 1) = ite(n_1 == 0, 1, ite(n_1 == 1, 1, n_1 * f_1(n_1))) \\ f_1(0) = 1$$

The above conditional recurrence of f_1 satisfies the the condition of type 2. $\mathcal{C}$ return the following closed-form solution for the conditional recurrence of f_1:

$$f_1(n_1) = factor(n_1)$$

where $factor$ is pre-defined function in Sympy [2].

Type 3 A conditional recurrence σ of the form (5) is of type 3 if each condition θ_i, $1 \leq i \leq h$, contains only arithmetic comparisons $>$ or $<$, polynomial functions of n, and does not mention $X(n)$.

Given such a type 3 recurrence, our system tries to compute the ranges of θ_i as follows. For each $1 \leq i \leq h$, 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 $i \leq h$

$$\forall n. C_{i-1} \leq n < C_i \rightarrow \phi_{\pi(i)}(n), \\ \forall n. n \geq C_i \rightarrow \neg\phi_{\pi(i)}(n),$$

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

$$\forall n. 0 \leq n < C \rightarrow \phi_k(n), \\ \forall n. n \geq C \rightarrow \neg\phi_k(n).$$

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 module abort with $\emptyset$ as the return.

Once the system suceeds in computing these constants, it introduces a new function g_i for each $1 \leq i \leq h+1$, and call

$$\begin{aligned} rsolve(g_i(n+1) &= f_{\pi(i)}(g_i(n), n), \\ g_i(0) &= E_{\pi(i)-1}(n/C_{\pi(i)-1}). \end{aligned}$$

If any of these calls does not return a closed-form solution, then the procedure abort and return $\emptyset$. Now suppose for each g_i, $rsolve()$ returns a closed-form solution E_i: $g_i(n) = E_{\pi(i)}(n)$, then the system returns the following closed-form solution:

$$\begin{aligned} X(n) = ite(&C_0 \leq n < C_1, E_1(n), \\ &ite(n < C_2, E_2(n), ..., \\ &\quad ite(n < C_h, E_h(n), E_{h+1}(n))..) \end{aligned} \tag{8}$$

Example 23

$$\begin{aligned} X(n+1) = ite(&7 \leq n < 10, X(n) + 2, \\ &ite(n \geq 10, X(n) - 2, ite(n < 7, \\ &\quad X(n) + 1, X(n) + 3)))) \\ X(0) = 0 \end{aligned}$$

The above conditional recurrence of X satisfies the condition of type 3 conditional recurrence eqution. $\mathcal{C}$ return the following closed-form solution for the conditional recurrence of X.

$$\begin{aligned} X(n) = ite(&0 \leq n < 7, n, \\ &ite(n < 10, 7 + 2 * (n - 7), \\ &\quad 13 - 2 * (n - 10))) \end{aligned}$$

The proof of correctness of this closed-form solution is presented in Proposition (3).

Proposition 3. *Equation (8) represents closed-form solution of the conditional recurrence (5) when it is conditional recurrence is type 3.*

Proof. Base case: The base case for conditional equation (5) is $X(0) = E$. We can also have $X(0) = E_1(0)$ from equation(8) when $n = 0$ as then condition $C_0 \leq n < C_1$ is satisfied. As we know that $E_1(n)(= g_i(n))$ is the closed form solution of $g_i(n+1) = f_1(g_i(n), n)$ with respect to initial value $X(0) = E$. Hence $E_1(0) = E$ and we can conclude that base case hold.

$$
\begin{aligned}
X(k) = ite(&C_0 \leq k < C_1, E_1(k),\\
&ite(k < C_2, E_2(k), ...,\\
&\quad ite(k < C_h, E_h(k), E_{h+1}(k))..)
\end{aligned} \tag{9}
$$

We assume equation (9) is correct where $k \geq 0$. Now using recurrences equation (5) with initial value $X(0) = E$, we tried to find equation (10) also hold using case analysis

$$
\begin{aligned}
X(k+1) = ite(&C_0 \leq (k+1) < C_1, E_1(k+1),\\
&ite((k+1) < C_2, E_2(k+1), ...,\\
&\quad ite((k+1) < C_h, E_h(k+1), E_{h+1}(k+1))..)
\end{aligned} \tag{10}
$$

– **Case 1:** When $C_0 \leq (k+1) < C_1$ is true. We can derive $C_0 \leq k < C_1$ is also true from the assumption (9). Then we have $X(k) = E_1(k)$ from assumption (9). Now from recurrences equation (5), we can also derive condition $\theta_{\pi^{-1}(1)}$ is true using $C_0 \leq (k+1) < C_2$ as it satisfies the following equations.

$$
\begin{aligned}
&\forall n. C_0 \leq n < C_1 \rightarrow \phi_{\pi^{-1}(1)}(n),\\
&\forall n. n \geq C_1 \rightarrow \neg\phi_{\pi^{-1}(1)}(n).
\end{aligned}
$$

As we know that $E_1(n)(= g_{\pi^{-1}(1)}(n))$ is the closed form solution of $g_{\pi^{-1}(1)}(n+1) = f_1(g_{\pi^{-1}(1)}(n), n)$ with respect to initial value $g_{\pi^{-1}(1)}(0) = E$. Hence we can $g_{\pi^{-1}(1)}(n+1) = E_1(n+1)$. Now we have from recurrence equation (5)

$$
X(k+1) = f_{\pi^{-1}(1)}(X(k), k) = f_{\pi^{-1}(1)}(E_1(k), k) = E_1(k+1) \tag{11}
$$

– **Case 2:** When $(k+1) < C_2$ is true. We can derive $k < C_2$ is also true from the assumption (9). Then we have $X(k) = E_1(k)$ from assumption (9). Now from recurrences equation (5), we can also derive condition θ_i is true using $(k+1) < C_2$ as it satisfies the following equations.

$$
\begin{aligned}
&\forall n. C_1 \leq n < C_2 \rightarrow \phi_{\pi^{-1}(2)}(n),\\
&\forall n. n \geq C_1 \rightarrow \neg\phi_{\pi^{-1}(2)}(n).
\end{aligned}
$$

As we know that $E_1(n)(= g_{\pi^{-1}(2)}(n))$ is the closed form solution of $g_{\pi^{-1}(2)}(n+1) = f_1(g_{\pi^{-1}(2)}(n), n)$ with respect to initial value $g_{\pi^{-1}(2)}(0) = E_1(n/C_1)$. Hence we can $g_{\pi^{-1}(2)}(n+1) = E_1(n+1)$. Now we have from recurrence equation (5)

$$
X(k+1) = f_{\pi^{-1}(2)}(X(k), k) = f_{\pi^{-1}(2)}(E_2(k), k) = E_2(k+1) \tag{12}
$$

Similarly we can show the other cases $h-2$ to show that equation (10) correct.

Type 4 When input conditional recurrence equation σ is either in the form of the equation (13) or (14) is called of type 4

$$X(n+1) = ite(f(n)\%c == d, X(n) \pm A, X(n) \pm B) \tag{13}$$
$$X(n+1) = ite(f(n)\%c != d, X(n) \pm A, X(n) \pm B) \tag{14}$$

Such that $c > 0, d >= 0, d < c$ where c, d, A, B are integer constants and $f(n)$ is polynomial functions of n. Then $\mathcal{C}(\sigma, X(0) = E, n)$ return f_σ, where $C = d - f(0)\%c$ and C is an integer.

– f_σ is of the following when input equation σ is in the form of the equation (13)

$$\begin{aligned} X(n) = ite(&0 \le n \le C, E + n * B, \\ &ite(n == C+1, E + A + C * B, \\ &\quad E + A + C * B + (\lfloor (n - C - 1)/c \rfloor) * (\pm A) \\ &\qquad +(n - C - 1 - \lfloor n - C - 1/c \rfloor) * (\pm B))) \end{aligned} \tag{15}$$

The proof of correctness of this closed-form solution is presented in Proposition (4).

Proposition 4. *Equation (15) represents closed-form solution of the conditional recurrence (13) when it is conditional recurrence is type 4.*

Proof. Base case: The base case for conditional equation (5) is $X(0) = E$. We can also have $X(0) = E \pm 0 * B$ from equation(13) when $n = 0$ as then condition $0 \le n \le C$ is satisfied. Hence we can conclude that base case hold.

$$\begin{aligned} X(k) = ite(&0 \le n \le C, E + n * B, ite(n == C+1, \\ &E + A + C * B, E + A + C * B + (\lfloor (k - C - 1)/c \rfloor) * (\pm A) \\ &\quad +(k - C - 1 - \lceil k - C - 1/c \rceil) * (\pm B))) \end{aligned} \tag{16}$$

We assume equation (34) is correct where $k \ge 0$. Now using recurrences equation (24) with initial value $X(0) = E$, we tried to find equation (35) also hold using case analysis

$$\begin{aligned} X(k+1) = ite(&0 \le k+1 \le C, E + n * B, ite(k+1 == C+1, \\ &E + A + C * B, E + A + C * B + (\lfloor (k+1 - C - 1)/c \rfloor) * (\pm A) \\ &\quad +(k+1 - C - 1 - \lceil k+1 - C - 1/c \rceil) * (\pm B))) \end{aligned} \tag{17}$$

- When $0 \le k+1 \le C$ is true, we can derive $0 \le k \le C$ also true. So we have $X(k) = E + k * B$ from inductive assumption equation (16). Lets consider $f(0)\%c = x$ such that $x \ne d$. Similarly $f(1)\%c = x +$

$1, ...f(y)\%c = x + y = d$. Hence we can have $y = d - x = C$. Now we can have following from recurrence equation (13) as $\neg(f(k)\%c == d)$ holds when $0 \leq k \leq C$.

$$X(k+1) = X(k) \pm B = E \pm k * B \pm B = E \pm (k+1) * B \quad (18)$$

- When $k + 1 = C + 1$ is true, then we can derived that $k = C$. So we have $X(k) = E + k * B$ from inductive assumption equation (16). Now we can have following from recurrence equation (13) as $(f(C)\%c == d)$ holds when $0 \leq k \leq C$.

$$X(k+1) = X(k) \pm A = E \pm k * B \pm A = E \pm C * B \pm A \quad (19)$$

- When $k + 1 > C + 1$ is true. We can also derived following possible scenario from assumption equation (16).
  - When $k = C + 1$ is true. So we have $X(k) = E \pm C * B \pm A$ from inductive assumption equation (16). Now we can have following from recurrence equation (13) as $(f(k)\%c == d)$ holds when $k == C + 1$.

$$\begin{aligned} X(k+1) &= X(k) \pm B \\ &= E \pm C * B \pm A \pm B \\ &= E \pm C * B \pm A \pm 0 * A \pm 1 * B \\ &= E \pm C * B \pm A \pm \lfloor (1/c \rfloor * A \pm (1 - \lfloor (1/c \rfloor) * B \\ &= E \pm C * B \pm A \pm \lfloor (k + 1 - C - 1)/c \rfloor * A \pm \\ &\qquad ((k + 1 - C - 1) - \lfloor ((k + 1 - C - 1)/c \rfloor) * B \end{aligned} \quad (20)$$

  - When $k > C + 1$ is true. So we have $X(k) = E + A + C * B + (\lfloor (k - C - 1)/c \rfloor) * (\pm A) + (k - \lceil k - C - 1/c \rceil) * (\pm B)$ from inductive assumption equation (16). When we expand the recurrence equation (13), we can have the following:

$$... + \underbrace{B + ... + B}_{\text{c times}} + A + \underbrace{B ... + B}_{\text{c times}} + A + ...$$

    From that we can conclude that when $f(k)\%c == d$ is true, then $(k - C - 1)\%c == c - 1$ is true. Similarly, when $f(k)\%c == d$ is true, $(k - C - 1)\%c \neq c - 1$ is true.
    - When $f(k)\%c == d$, then we have

$$\begin{aligned} X(k+1) &= X(k) \pm A \\ &= E + A + C * B + (\lfloor (k - C - 1)/c \rfloor) * (\pm A) \\ &\qquad + ((k - C - 1) - \lceil k - C - 1/c \rceil) * (\pm B) \pm A \\ &= E + A + C * B + (\lfloor (k - C - 1)/c \rfloor + 1) * (\pm A) \\ &\qquad + ((k - C - 1) - \lceil k - C - 1/c \rceil) * (\pm B) \\ &= E + A + C * B + (\lfloor (k + 1 - C - 1)/c \rfloor) * (\pm A) \\ &\qquad + ((k + 1 - C - 1) - \lceil k + 1 - C - 1/c \rceil) * (\pm B) \end{aligned} \quad (21)$$

Hence $\lfloor (k+1-C-1)/c \rfloor = \lfloor (k-C-1)/c \rfloor + 1$ as we have $(k-C-1)\%c == c-1$.

· When $f(k)\%c \neq d$, then we have

$$\begin{aligned} X(k+1) &= X(k) \pm B \\ &= E + A + C * B + (\lfloor (k-C-1)/c \rfloor) * (\pm A) \\ &\quad + ((k-C-1) - \lceil k-C-1/c \rceil) * (\pm B) \pm B \\ &= E + A + C * B + (\lfloor (k-C-1)/c \rfloor) * (\pm A) \\ &\quad + ((k+1-C-1) - \lceil k-C-1/c \rceil) * (\pm B) \\ &= E + A + C * B + (\lfloor (k+1-C-1)/c \rfloor) * (\pm A) \\ &\quad + ((k+1-C-1) - \lceil k+1-C-1/c \rceil) * (\pm B) \end{aligned} \tag{22}$$

Hence $\lfloor (k+1-C-1)/c \rfloor = \lfloor (k-C-1)/c \rfloor$ as we have $(k-C-1)\%c \neq c-1$.

From result of (18), (19), (20), (21) and (22), we concluded that equation (17) correct.

– f_σ is of the following when input equation σ is in the form of the equation (14)

$$\begin{aligned} X(n) = ite(&n \leq C, E + n * A, \\ &ite(n == C+1, E + C * A + B, \\ &\quad E + C * A + B + (\lfloor (n-C-1)/c \rfloor) * (\pm B) \\ &\qquad + (n - C - 1 - \lfloor n - C - 1/c \rfloor) * (\pm A))) \end{aligned} \tag{23}$$

The proof of correctness of this closed-form solution is presented in Proposition (5).

Proposition 5. *Equation (8) represents closed-form solution of the conditional recurrence (5) when it is conditional recurrence is type 3.*

Proof. Similarly this also can be proved.

Example 24 *Lets consider following conditional recurrence of type 4*

$$y_3(n_1 + 1) = ite(n_1 \% 3 == 0, y_3(n_1) + A, y_3(n_1) + B)$$

$$y_3(0) = 0$$

The resulting closed-form solution of the conditional recurrence equations are produced by $\mathcal{C}$ *.*

$$y_3(n_1) = \lfloor (n_1/3) \rfloor * A + (n_1 - \lfloor (n_1/3) \rfloor) * B$$

Type 5 A conditional recurrence σ of the form (5) is of type 5 if it is of the following form (24):

$$X(n+1) = ite(\theta, X(n) \pm A, X(n) \mp B), \tag{24}$$

where θ contains only arithmetic comparisons $>$ or $<$, polynomial functions of $X(n)$ and n. A and B are integer constants such that for some h, either $A = B*h$ or $B = A*h$. Then it check whether $\phi(0)$ holds, and then asks the SMT solver to find a model of

$$\forall n.0 \leq n < C \rightarrow \theta(X(n)/(E + n*A)) \\ \wedge \neg\phi(X(n)/(E + C*A))$$

If it returns a model, then $\mathcal{C}(\sigma, X(0) = E, n)$ return f_σ as following:

– If $A = B*h$ for some h, then f_σ is

$$X(n) = ite(0 \leq n < C, E \pm n*A, \\ ite((n-C)\%(h+1) == 0, E \pm C*A, \\ E \pm C*A \mp (((n-C)\%(h+1)))*B)) \tag{25}$$

The proof of correctness of this closed-form solution is presented in Proposition (6).

Proposition 6. *Equation (25) represents closed-form solution of the conditional recurrence (24) when it is conditional recurrence is type 5 and $A = B*h$ for some $h > 0$.*

Proof. Base case: The base case for conditional equation(5) is $X(0) = E$. We can also have $X(0) = E \pm 0*A$ from equation (33) when $n = 0$ as then condition $0 \leq n < C$ is satisfied. Hence we can conclude that base case hold.

$$X(k) = ite(0 \leq k < C, E \pm k*A, ite((k-C)\%(h+1) == 0, \\ E \pm C*A, E \pm C*A \mp (((k-C)\%(h+1)))*B)) \tag{26}$$

We assume equation (34) is correct where $k \geq 0$. Now using recurrences equation (24) with initial value $X(0) = E$, we tried to find equation (35) also hold using case analysis

$$X(k+1) = ite(0 \leq (k+1) < C, E \pm (k+1)*A, ite((k+1-C)\%(h+1) == 0, \\ E \pm C*A, E \pm C*A \mp (((k+1-C)\%(h+1)))*B)) \tag{27}$$

- Case 1: When $0 \leq (k+1) < C$ is true. We can also derived $0 \leq k < C$ is true from equation (34). Now from recurrences equation (33), we can also derive condition θ is true using $0 \leq (k+1) < C$ as it satisfies the following equations.

$$\forall n.0 \leq n < C \rightarrow \theta(X(n)/(E \pm n*A)) \wedge \neg\phi(X(n)/(E \pm C*A))$$

$$X(k+1) = X(k) \pm A = E \pm k*A \pm A = E \pm (k+1)*A \tag{28}$$

- Case 2: When $(k+1-C)\%(h+1) == 0$ is true. We can also derived following possible scenario from assumption equation (34).
  - When $k < C$ and $(k-C)\%(h+1) \neq 0$, we can have $(k-C)\%(h+1) == 1$ and $k = C-1$. From inductive assumption, we have $X(k) = E \pm k * A = E \pm (C-1) * A$. Hence $\theta(X(k)/E \pm (C-1) * A)$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} X(n+1) &= X(n) \pm A \\ &= E \pm (C-1) * A \pm A = E \pm C * A \end{aligned} \tag{29}$$

  - When $k \geq C$ and $(k-C)\%(h+1) \neq 0$. we can have $(k-C)\%(h+1) == h$. From inductive assumption, we have $X(k) = E \pm C * A \mp (((k-C)\%(h+1))) * B))$. We can derive that $(k-C)\%(h+1) == h$.

$$\begin{aligned} &E \pm C * A \mp (((k-C)\%(h+1))) * B)) \\ &= E \pm C * A \mp h * B \end{aligned}$$

    Hence $\theta(X(k)/(E \pm C * A \mp h * B))$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} X(n+1) &= X(n) \pm A \\ &= E \pm C * A \mp h * B \pm A \\ &= E \pm C * A \end{aligned} \tag{30}$$

  - When $k < C$ and $(k-C)\%(h+1) == 0$. This scenario is not possible because it is contradicted the assumption $(k+1-C)\%(h+1) == 0$
  - When $k \geq C$ and $(k-C)\%(h+1) == 0$. This scenario is not possible because it is contradicted the assumption $(k+1-C)\%(h+1) == 0$
- Case 3: When $(k+1) \geq C \wedge (k+1-C)\%(h+1) \neq 0$ is true. We can also derived following possible scenario from assumption equation (34).
  - When $k < C$ and $(k-C)\%(h+1) \neq 0$, we can have $(k-C)\%(h+1) == 1$,$k = C-1$ and $(k+1-C)\%(h+1) == 0$. This scenario is not possible, as it is contradicted the assumption.
  - When $k < C$ and $(k-C)\%(h+1) == 0$. This scenario is not possible, as it is contradictory because $(k-C)\%(h+1) == 0$ hold on when $k = -C$.
  - When $k > C$ and $(k-C)\%(h+1) \neq 0$, we can have $X(k) = E \pm C * A \mp (((k-C)\%(h+1))) * B$. Hence $\neg\theta(X(k)/(E \pm C * A \mp (((k-C)\%(h+1))) * B))$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} &X(n+1) = X(n) \pm B \\ &= E \pm C * A \mp (((k-C)\%(h+1))) * B \mp B \\ &= E \pm C * A \mp (((k-C)\%(h+1)) + 1) * B \\ &= E \pm C * A \mp (((k+1-C)\%(h+1))) * B \end{aligned} \tag{31}$$

* When $k > C$ and $(k-C)\%(h+1) == 0$, we can have $X(k) = E \pm C * A$. Hence $\neg\theta(X(k)/(E \pm (C-1) * A))$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} X(n+1) &= X(n) \pm B \\ &= E \pm (C-1) * A \mp B \\ &= E \pm C * A \mp (((k+1-C)\%(h+1))) * B \end{aligned} \tag{32}$$

From $(k-C)\%(h+1) == 0$ and $(k+1-C)\%(h+1) \neq 0$, we can conclude that $((k+1-C)\%(h+)) == 1$

From result of (28), (29), (30), (31) and (32), we concluded that equation (27) correct.

– If $B = A * h$ for some h, then f_σ is

$$\begin{aligned} X(n) = ite(&0 \leq n < C, E \pm n * A, \\ &ite((n-C)\%(h+1) == 0, E \pm C * A, \\ &\quad ite((n-C)\%(h+1) == 1, E \pm C * A \mp B, \\ &\quad\quad E \pm C * A \mp B \pm (((n-C)\%(h+1)) - 1) * A))) \end{aligned} \tag{33}$$

The proof of correctness of this closed-form solution is presented in Proposition (7).

Proposition 7. *Equation (33) represents closed-form solution of the conditional recurrence (24) when it is conditional recurrence is type 5 and $B = A * h$ for some $h > 0$.*

Proof. Base case: The base case for conditional equation (5) is $X(0) = E$. We can also have $X(0) = E \pm 0 * A$ from equation (33) when $n = 0$ as then condition $0 \leq n < C$ is satisfied. Hence we can conclude that base case hold.

$$\begin{aligned} X(k) = ite(&0 \leq k < C, E \pm k * A, \\ &ite((k-C)\%(h+1) == 0, E \pm C * A, \\ &\quad ite((k-C)\%(h+1) == 1, E \pm C * A \mp B, \\ &\quad\quad , E \pm C * A \mp B \pm (((k-C)\%(h+1)) - 1) * A))) \end{aligned} \tag{34}$$

We assume equation (34) is correct where $k \geq 0$. Now using recurrences equation (24) with initial value $X(0) = E$, we tried to find equation (35) also hold using case analysis

$$\begin{aligned} X(k+1) = ite(&0 \leq (k+1) < C, E \pm (k+1) * A, \\ &ite(((k+1)-C)\%(h+1) == 0, E \pm C * A, \\ &\quad ite(((k+1)-C)\%(h+1) == 1, E \pm C * A \mp B, \\ &\quad\quad , E \pm C * A \mp B \pm ((((k+1)-C)\%(h+1)) - 1) * A))) \end{aligned} \tag{35}$$

- Case 1: When $0 \leq (k+1) < C$ is true. We can also derived $0 \leq k < C$ is true from equation (34). Now from recurrences equation (33), we can also derive condition θ is true using $0 \leq (k+1) < C$ as it satisfies the following equations.

$$\forall n.0 \leq n < C \rightarrow \theta(X(n)/(E + n * A)) \wedge \neg\theta(X(n)/(E + C * A))$$

$$X(k+1) = X(k) \pm A = E \pm k * A \pm A = E \pm (k+1) * A \quad (36)$$

- Case 2: When $(k+1-C)\%(h+1) == 0$ is true. We can also derived following possible scenario from assumption equation (34).
  - When $k < C$ and $(k-C)\%(h+1) \neq 0$, we can have $(k-C)\%(h+1) == 1$ and $k = C-1$. From inductive assumption, we have $X(k) = E \pm k * A = E \pm (C-1) * A$. Hence $\theta(X(k)/E \pm (C-1) * A)$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} X(n+1) &= X(n) \pm A \\ &= E \pm (C-1) * A \pm A = E \pm C * A \end{aligned} \quad (37)$$

  - When $k \geq C$ and $(k-C)\%(h+1) \neq 0$. we can have $(k-C)\%(h+1) == h$. From inductive assumption, we have $X(k) = E \pm C * A \mp B \pm (((n-C)\%(h+1)) - 1) * A = E \pm C * A \mp B \pm (h-1) * A$. Hence $\theta(X(k)/(E \pm C * A \mp B \pm (h-1) * A)$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} X(n+1) &= X(n) \pm A \\ &= E \pm C * A \mp B \pm (h-1) * A \pm A \\ &= E \pm C * A \mp B \pm h * A \\ &= E \pm C * A \end{aligned} \quad (38)$$

    Hence we know that $B = h * A$.
  - When $k < C$ and $(k-C)\%(h+1) == 0$. This scenario is not possible because it is contradicted the assumption $(k+1-C)\%(h+1) == 0$
  - When $k \geq C$ and $(k-C)\%(h+1) == 0$. This scenario is not possible because it is contradicted the assumption $(k+1-C)\%(h+1) == 0$
- Case 3: When $((k+1)-C)\%(h+1) == 1$ is true, we can derive that $(k-C)\%(h+1) == 0$. We can $X(k) = E \pm C * A$. Hence $\neg\theta(X(k)/E \pm C * A)$ is hold, we can have following from recurrence equation (24).

$$X(n+1) = X(n) \mp B = E \pm C * A \mp B = E \pm C * A \mp B \quad (39)$$

- Case 4: When $(k+1) \geq C \wedge (k+1-C)\%(h+1) \neq 0 \wedge (k+1-C)\%(h+1) \neq 1$ is true. We can also derived following possible scenario from assumption equation (34).
  - When $k < C$ and $(k-C)\%(h+1) == 0$. This scenario is not possible, as it is contradictory because $(k-C)\%(h+1) == 0$ hold on when $k = -C$.

* When $k < C$ and , we can have $(k-C)\%(h+1) == 1$,$k = C-1$ and $(k+1-C)\%(h+1) == 0$. This scenario is not possible, as it is contradicted the assumption because we cannot have $(k+1) \geq C$.
* When $k < C$, $(k-C)\%(h+1) \neq 1$ and $(k-C)\%(h+1) \neq 0$. This scenario is not possible, as it is contradicted the assumption.
* When $k \geq C$ and $(k-C)\%(h+1) == 0$, we can conclude that $(k+1-C)\%(h+1) == 1$ which is contradicted the assumption. This scenario is not possible.
* When $k \geq C$ and $(k-C)\%(h+1) == 1$, we can have $X(k) = E \pm C * A \mp B$. Hence $\theta(X(k)/(E \pm C * A \mp B)$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} & X(n+1) = X(n) \pm A \\ & = E \pm C * A \mp B \pm A \\ & = E \pm C * A \mp B \pm 1 * A \\ & = E \pm C * A \mp B \pm (2-1) * A \\ & = E \pm C * A \mp B \pm (((n+1-C)\%(h+1)) - 1) * A \end{aligned} \tag{40}$$

From $(k-C)\%(h+1) == 1$, we can drive $(k+1-C)\%(h+1) == 2$.

* When $k \geq C$, $(k-C)\%(h+1) \neq 1$ and $(k-C)\%(h+1) \neq 0$, we can have $X(k) = E \pm C * A \mp B \pm (((k-C)\%(h+1)) - 1) * A$. Hence $\theta(X(k)/(E \pm C * A \mp B \pm (((k-C)\%(h+1)) - 1) * A)$ is hold, we can have following from recurrence equation (24).

$$\begin{aligned} & X(n+1) = X(n) \pm A \\ & = E \pm C * A \mp B \pm (((k-C)\%(h+1)) - 1) * A \pm A \\ & = E \pm C * A \mp B \pm (((k+1-C)\%(h+1)) - 1) * A \end{aligned} \tag{41}$$

From result of (36), (37), (38), (39), (40) and (41), we concluded that equation (35) correct.

Example 25 *Let us consider following conditional recurrence equation which of type 5.*

$$y_3(n+1) = ite(y_3(n) <= 50, y_3(n)+1, y_3(n)-1)$$

$$y_3(0) = 0$$

The resulting closed form solution of the conditional recurrence equations are produced by $\mathcal{C}$.

$$y_3(n) = ite(0 \leq n < 50, n, ite((n-50)\%2 == 0, 50, 49))$$

2.4 Substitute Result (SR)

The substitute function $SR(\boldsymbol{f}, \mathcal{E})$ takes two arguments. The first argument $\boldsymbol{f}$ is a set of closed-form solutions:

$$\boldsymbol{f} = (f_1(n) = e_1(n), \cdots, f_k(n) = e_k(n)),$$

and the second argument $\mathcal{E}$ is a set of expressions. The function returns a new set of expressions obtained by replacing occurrence of $f_i(t)$ in the expressions in $\mathcal{E}$, for every $1 \leq i \leq k$ and every term t, by $e_i(t)$.

2.5 Simplify Condition

Recurrence Solver use a systematic technique for simplify condition of recurrence equation σ the form present in (5) using Simplify Condition (SC). It use five different rules which are presented in Figure (2). The rule SC_1 produces a non-conditional recurrence equation if identifying $f_1(X(n), n) = f_2(X(n), n) = ... = f_{h-1}(X(n), n) = f_h(X(n), n)$ with an SMT query. Algorithm uses the rules SC_2 on input axiom where it processes the conditions by applying proof strategy sequentially presented in [10]. Upon evaluation of the condition to true, it produces the modified axioms.

$$SC_1 \frac{\sigma}{\dfrac{f_1(X(n), n) = f_2(X(n), n) = ... = f_h(X(n), n) = f_{h+1}(X(n), n)}{X(n+1) = f_1(X(n), n)}}$$

$$SC_2 \frac{\sigma}{\dfrac{\theta_1}{X(n+1) = f_1(X(n), n)}}$$

Fig. 2: The set of rules for simplifying σ

These rules are applied recursively until fixed point. In essence, they remove irrelevant and redundant parts of the recurrence equation(s).

3 Evaluation

The goals of our experiments are 1) to study how the integration of Recurrence Solver increases the capability of our previously proposed fully automated program verification system called VIAP and 2) to demonstrate the usefulness of Recurrence Solver in program equivalence checking tool Eq VIAP .

3.1 Implementation

RS is implemented as a stand-alone application, which uses Sympy [24] computer algebra system and Z3 [25] SMT solver. The current implementation of

VIAP executes using a single thread. The tool along with source code is publicly available in the following URL:
https://github.com/VerifierIntegerAssignment/recSolver

VIAP and Eq-VIAP : VIAPis also implemented as a stand-alone application using python. VIAPis integrated with RS to solve a recurrence and Z3 [25] SMT solver is used as back-end theorem prover. Eq-VIAP is also built on VIAP by adding a sequential composition module. The tool and benchmark are publicly available (TODO: add a link for artifact evaluation?)

3.2 Experimental setup

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

- **RQ1** How effectiveness can VIAP and Eq-VIAP prove the safety property and equivalences compared to the state-of-the-art automated verifies and equivalence checkers respectively?
- **RQ2** How significant is the impact of recurrence solving in VIAP and Eq-VIAP ?

Baseline We compared VIAP and Eq-VIAP with two groups of tools. For the first group, we compared VIAP against three state-of-the-art automatic verifiers (ICRA [17], Ultimate Automizer [19], VeriAbs [18], Seahorn [20], and CPAChecker [21]) by invoking them to verify the product programs constructed by VIAPThe second group consist of the state-of-the-art equivalence checker REVE [26] . Both of them generate verification condition consisting of constraint horn clauses (CHC) that are solved with IC3/PDR [27]. However, they take specialized procedures, other than sequential composition, for constructing the product program. We also demonstrate the impact of integration of RS in the capabilities VIAP and Eq-VIAP by showing the number of the programs it handles before and after integration of RS.

Subjects In our experimental evaluation, we have collected a suite of 77 benchmarks program from the experiment of [17], and 208 benchmarks programs from latest $SV-COMP$ standard benchmark from ReachSafety-Loops sub-category. In the most recent *SV-COMP* 2019 competition,VIAP have successfully prove 124 programs and it score 178 (confirm correct 90 and unconfirm correct 34)[1]. VIAP came in first in the ReachSafety-Arrays and ReachSafety-Recursive sub-category. Similarly, we have collected a suite of 20 pairs of programs in the criteria from equivalence checking papers [16]. The primary objective of the inclusion of programs from [17] is to demonstrate how VIAP can effectively handle programs with multi-path execution body.

All experiments were done on an Intel® i3 1.8 GHz machine with 4GB of memory running on the Ubuntu 16.04 operating system.

[1] https://goo.gl/DNeUCr

3.3 Results

The results of our evaluation are in Table 1. For each tool, we record the status and execution.

RQ1 Table 1 and Table 2 shows that VIAP and Eq-VIAP were able to prove the partial correctness and equivalence of an overwhelming majority of programs to which it was applied. VIAP performs significantly competitive compared to the other tools on these benchmarks.

First, Eq-VIAP successfully proved the equivalence the benchmarks from other state-of-art equivalence checking tools, namely REVE [16] and PEQUOD [28] which presented in Table 2. Similarly, VIAP successfully prove the safety property of programs collected from the experiment of [17] and SV-COMP benchmark which has 5 loop categories. The experimental results which demonstrate that are presented in Table 1.

Second, Eq-VIAP was able to prove the equivalence of programs with non-linear return values. We can observe that other state-of-the-art equivalence checkers (REVE [16] were unable to prove programs of non-linear nature, especially those with return values that are polynomials with arbitrary exponents. The main purpose of this experiment is to highlight that state-of-the-art equivalence checkers leak in handling non-linear reture type program. Benchmark for this experiment are constructed from the benchmarks of NLA test suite of [29] and benchmarks from experiment [17]. Detail result of this experiment is presented in Table 3. Similarly, VIAP can also handle the program of non-linear nature like ICRA [17]. On another hand, other tool failed to do so.

Table 1: The results of the experiments to check equivalences of program pairs

Benchmark	Total	VIAP		VIAP+RS		VeriAbs		Sea		ICRA		UAut.	
Suit	#A	Time	#A	Time	#A	Time	#A	Time	#A	Time	#A	Time	#A
ICRA	77		20		68		44		56		77		20
SV-COMP	208		76		124		182		102		134		146
Total	285												

RQ2 VIAP takes x and y seconds on average to prove benchmarks of [16] and [28] respectively. [16] took z seconds and [28] took w seconds. The main overhead of VIAP comes from translating to first-order logic, solving recurrences and in some cases applying proof strategy to discover appropriate lemma to prove the equivalence.

The tool and benchmarks are publicly available (TODO: add a link for artifact evaluation?)

Table 2: The results of the experiments to check equivalences of program pairs (Todo:few programs from). Program pairs 1-20 are from benchmarks of REVE [16]

SL.no	Program	Eq-VIAP		Eq-VIAP+RS		REVE	
		Status	t_1 (s)	Status	t_1 (s)	Status	t_1 (s)
1	barthe	✓	27	✓	38	✓	12
2	barthe2	×	-	✓	31	✓	15
3	barthe2-big	×	-	✓	48	✓	16
4	break	×	25	✓	35	✓	21
5	bug15	×	28	✓	42	✓	19
6	const_factor	✓	28	✓	29	✓	13
7	cube_square	✓	30	✓	48	✓	17
8	digits10	×	-	×	-	✓	13
9	fib	✓	28	✓	22	✓	18
10	loop	✓	25	✓	31	✓	18
11	loop2	✓	21	✓	29	✓	12
12	loop2	×	-	✓	27	✓	15
13	loop3	×	-	✓	34	✓	18
14	loop_unswitching	×	-	✓	30	✓	29
15	simple-loop	✓	26	✓	33	✓	17
16	square_twice	×	-	✓	41	✓	16
17	while_after_while_if	✓	28	✓	38	✓	25
18	while_if	✓	28	✓	35	✓	20
19	nested˙while	✓	35	✓	38	✓	12
20	claimbstair	×	-	✓	34	✓	18

Table 3: The results of the experiments to check equivalences of program pairs. Program pairs construct from the benchmarks of NLA test suite of [29] and benchmarks from experiment [17]

SL.no	Program	Eq-VIAP		Eq-VIAP+RS		REVE	
		Status	t_1 (s)	Status	t_1 (s)	Status	t_1 (s)
1	program1	✓	60	✓	60	×	-
2	program2	✓	57	✓	57	×	-
3	program3	✓	33	✓	33	×	-
4	program4	✓	37	✓	37	×	-
5	program5	✓	29	✓	29	×	-
6	program6	✓	31	✓	31	×	-
7	program7	✓	37	✓	37	×	-
8	program8	✓	39	✓	39	×	-
9	program9	✓	41	✓	41	×	-
10	program10	✓	33	✓	33	×	-
11	program11	✓	37	✓	37	×	-
12	program12	✓	34	✓	34	×	-

4 Related Work

In recent year program analyzer based on recurrence analysis are gaining importance. The work of [?] is a template-based approach used which abstract interpretation to detect induction variables of recurrence relations of programs containing loops. The scope of this approach is very limited as it can handle a certain type of loop by making the assumption on the CFG of a program. The method of [?], [?] and [?] use recurrences analysis on computing accurate information about syntactically restricted loops. These approaches are also template based which handle a very simple class of recurrence equations. Ancourt et al. [?] is based on affine derivative closure method where it approximates any loop by a translation. This method is specially designed to discover linear recurrence in equations. [?] and [?] computes the polynomial invariants of programs using abstract interpretation. The other methods only handle assignments which can be described by C-Finite recurrences. The [?] present algorithms which find out the symbolic bounds for nested loops. Then compute the closed forms of certain loops to use over-approximation and under-approximation to find out suitable polynomial invariant. Like [?], the method of finding closed form solution is limited to only C-Finite recurrences. The ALIGATOR tool [?], [?] another tool that infers loop invariants that are polynomial equalities by solving the recurrence equations representing the loop body in closed form. Recently published work [17] works on similar lines. But it handle subclass of recurrence relationship as opposed to [?] and [?]. The main advantage of [17] of over [?] and [?] is that it can handle more complex program structure such as nested loop and non-deterministic input. One important difference is invariant generation technique is assertion guided whereas The ALIGATOR tool generate all possible invariant. Recurrences also have been applied to program equivalence checking [14,30,14,31].

None of the above mentioned methods address how to find the closed form solutions when recurrence equations is conditional. . A major disadvantage of those methods is that if they fail to find closed form solution, then they are unable to find suitable invariant to complete the proof. Our method uses recurrence analysis to simplify set of axioms in order to help the theorem proving process. Whereas even if it is unable to solve recurrence equations, our system can use induction proof strategy to complete the proof. Specialized mathematical tool such as Maple [1], Sympy [2] and Mathematica [3] donot privide support to find closed form solution conditional recurrence equations

5 Conclusion

We have presented an automatic tool for computing closed form solution of recurrences. We have integrated this to our previously developed automated program verifier VIAP as well as newly developed program equivalence checking tool eq-VIAP . The interaction extended the capability of both the tools. We have demonstrated that by proving the correctness and equivalence of challenging programs from the benchmark of existing state-of-the-art tools. To the best

of our knowledge, this is the first work that can find the closed form solution of conditional recurrence function. In the future, we plan to extend recSolve to handle wide range of conditional recurrence, mutual recurrence and conditional mutual recurrence functions. We attempted to apply it to more applications such as analysis of the worst-case execution time, automatic generation of the unit test case and automatic program repair. For future work, we want to extend the both of our systems, VIAP and eq-VIAP with more powerful tactic for wide classes of programs, and for proving properties regarding termination.

Acknowledgment

We would like to thank Peisen YAO and Prasnta Saikia for useful discussions. We are grateful to the developers of Z3 and sympy for making their systems available for open use. All errors remain ours.

References

1. D. Redfern, *The Maple Handbook (Maple V Release 3)*. Berlin, Heidelberg: Springer-Verlag, 1994.
2. SymPy Development Team, *SymPy: Python library for symbolic mathematics*, 2016. [Online]. Available: http://www.sympy.org
3. Wolfram Research Inc., *Mathematica 8.0*, 2010. [Online]. Available: http://www.wolfram.com
4. M. Petkovsek, H. S. Wilf, and D. Zeilberger, $A = B$. Wellesley, Mass. : A K Peters, 1996.
5. 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.
6. N. P. Lopes and J. Monteiro, "Automatic equivalence checking of programs with uninterpreted functions and integer arithmetic," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 4, pp. 359–374, Aug 2016. [Online]. Available: https://doi.org/10.1007/s10009-015-0366-1
7. D. Kroening, N. Sharygina, S. Tonetta, A. Tsitovich, and C. M. Wintersteiger, "Loop summarization using state and transition invariants," *Formal Methods in System Design*, vol. 42, no. 3, pp. 221–261, 2013.
8. Z. Fang, X. Zhao, and M. Zhou, "Infer precise program invariant using abstract interpretation with recurrence solving," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, July 2017, pp. 196–201.
9. 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.
10. 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 2016, Timisoara, Romania, September 21-24, 2017*, 2017, pp. 137–144, available at https://github.com/VerifierIntegerAssignment/sv-comp/blob/master/viap-automated-system.pdf.

11. ——, "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
12. 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
13. L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. [Online]. Available: http://dl.acm.org/citation.cfm?id=1792734.1792766
14. N. P. Lopes and J. Monteiro, "Automatic equivalence checking of uf+ ia programs," in *International SPIN Workshop on Model Checking of Software*. Springer, 2013, pp. 282–300.
15. G. Barthe, P. R. D'Argenio, and T. Rezk, "Secure information flow by self-composition," in *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*. IEEE, 2004, pp. 100–114.
16. D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, and M. Ulbrich, "Automating regression verification," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. ACM, 2014, pp. 349–360.
17. 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.
18. B. Chimdyalwar, P. Darke, A. Chauhan, P. Shah, S. Kumar, and R. Venkatesh, "Veriabs: Verification by abstraction competition contribution," in *Proceedings, Part II, of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 10206*. New York, NY, USA: Springer-Verlag New York, Inc., 2017, pp. 404–408. [Online]. Available: https://doi.org/10.1007/978-3-662-54580-5_32
19. M. Heizmann, D. Dietsch, J. Leike, B. Musa, and A. Podelski, "Ultimate automizer with array interpolation," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 455–457.
20. A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas, *The SeaHorn Verification Framework*. Cham: Springer International Publishing, 2015, pp. 343–361. [Online]. Available: https://doi.org/10.1007/978-3-319-21690-4_20
21. D. Beyer and M. E. Keremoglu, "Cpachecker: A tool for configurable software verification," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 184–190.
22. S. A. Abramov, M. Bronstein, and M. Petkovšek, "On polynomial solutions of linear operator equations," in *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '95. New York, NY, USA: ACM, 1995, pp. 290–296. [Online]. Available: http://doi.acm.org/10.1145/220346.220384
23. M. Petkovšek, "Hypergeometric solutions of linear recurrences with polynomial coefficients," *J. Symb. Comput.*, vol. 14, no. 2-3, pp. 243–264, Aug. 1992. [Online]. Available: http://dx.doi.org/10.1016/0747-7171(92)90038-6
24. 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.

25. 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.
26. M. Kiefer, V. Klebanov, and M. Ulbrich, "Relational program reasoning using compiler ir," *Journal of Automated Reasoning*, vol. 60, no. 3, pp. 337–363, 2018.
27. A. R. Bradley, "Sat-based model checking without unrolling," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2011, pp. 70–87.
28. Q. Zhou, D. Heath, and W. Harris, "Completely automated equivalence proofs," *CoRR*, vol. abs/1705.03110, 2017. [Online]. Available: http://arxiv.org/abs/1705.03110
29. 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.
30. D. Barthou, P. Feautrier, and X. Redon, "On the equivalence of two systems of affine recurrence equations," in *European Conference on Parallel Processing*. Springer, 2002, pp. 309–313.
31. K. Shashidhar, M. Bruynooghe, F. Catthoor, and G. Janssens, "Verification of source code transformations by program equivalence checking," in *International Conference on Compiler Construction*. Springer, 2005, pp. 221–236.