Upper and Lower Amortized Cost Bounds of Programs Expressed as Cost Relations

Extended Version

Antonio Flores-Montoya
TU Darmstadt, Dept. of Computer Science
aeflores@cs.tu-darmstadt.de

Abstract. Resource analysis aims at statically obtaining bounds on the resource consumption of programs in terms of input parameters. A well known approach to resource analysis is based on transforming the target program into a set of cost relations, then solving these into a closed-form bound. In this paper we develop a new analysis for computing upper and lower cost bounds of programs expressed as cost relations. The analysis is compositional: it computes the cost of each loop or function separately and composes the obtained expressions to obtain the total cost. Despite being modular, the analysis can obtain precise upper and lower bounds of programs with amortized cost. The key is to obtain bounds that depend on the values of the variables at the beginning and at the end of each program part. In addition we use a novel cost representation called cost structure. It allows to reduce the inference of complex polynomial expressions to a set of linear problems that can be solved efficiently. We implemented our method and performed an extensive experimental evaluation that demonstrates its power.

Keywords: Cost analysis, Cost relations, Amortized cost, Lower bounds

1 Introduction

Cost or resource analysis aims at statically obtaining bounds on the resource consumption (such as time or memory consumption) of programs in terms of their input parameters. Such bounds constitute useful feedback for developers and help detect performance bugs. This is particularly relevant in the context of cloud applications where one pays according to the amount of resources used.

One common approach for computing both upper and lower bounds is based on cost relations (CRs) which are similar to recurrence equations annotated with linear constraints [2]. In this approach, the cost analysis is carried out in two phases: (1) given a program, for the given resource we want to measure (time, memory, etc.), we generate a set of recursive cost relations (CRs) that represent the cost of the program for the given resource; and (2) the CRs are then analyzed and a closed-form upper (or lower) bound expression is computed. Here CRs act as a language-independent intermediate representation. The second phase of the analysis can be reused to solve CRs generated from programs written in
different source languages (e.g., Java bytecode [4], ABS [1,16], Llvm IR [17]) and to measure different kinds of resources such as time or memory. Our work focuses on that second part of the analysis. Given a set of CRs, we present an analysis that obtains closed-form upper and lower bounds of its cost.

Example 1. Consider program 1 in Fig. 1. We use tick(c) annotations to indicate that c resource units are consumed (or released if c is negative) at an execution point. The term * (in line 6) represents an unknown value. Assuming the initial values of x, y and z are positive, the upper and lower cost bounds of function p1 are $\text{max}(2,z) \cdot (x+y)$ and $\text{min}(2,z) \cdot (x+y)$, respectively.

In the CR representation, we have 5 cost relations: p1, wh3, wh6, wh9 and wh12: one for the function p1 and one for each while loop located at lines 3, 6, 9 and 12. Each cost relation is composed of a set of cost equations. Each cost equation (CE) corresponds to a path of a loop or function and defines its cost. Each CE is annotated with set of linear constraints that model the conditions for its applicability and its behavior.

Consider CE 8 that represents the case where the loop condition is unsatisfied. Its cost is 0 and its constraint set is $\{i \geq z\}$. Conversely, CE 9 represents the case where $i < z$ and the loop body is executed. CE 9 defines the cost of wh12(i, z) as the cost of one iteration plus the cost of the remaining loop wh12(i', z), where $i'$ represents the value of i after one iteration $i' = i + 1$. In loop wh6 the cost of one iteration is 2 and the final value of y (i.e., yo) is included in the abstraction. Observe that at the base case of wh6 in CE 4 the initial and final values of y are equal: $y = yo$. The inclusion of final variable values in loops such as wh6 and wh3 is essential to compute precise bounds. Note that wh6 is non-deterministic, because the constraints of CE 4 and 5 are not mutually exclusive (due to the unknown value *).

Fig. 1. Program 1 and its cost relations
Cost relations have several advantages over other abstract representations: They support recursive programs naturally. In fact, loops are modelled as recursive definitions and that allows us to analyze loops and recursive functions in a uniform manner. In contrast, difference constraints do not support recursion [25] and integer rewrite systems need to be extended [9]. More importantly, CRs have a modular structure. Each loop or function is abstracted into a separate cost relation. This enables a compositional approach to compute the cost of a program by combining the costs of its parts.

In our example, we first compute the cost of entering the inner loop \( wh6 \), then use it to compute the cost of the outer loop \( wh3 \). Similarly for loops \( wh12 \) and \( wh9 \). Finally, we combine the cost of loop \( wh3 \) with that of loop \( wh9 \) to obtain the total cost of the program. Each relation is computed only once.

Besides being compositional, we want our analysis to be precise. This is challenging for program 1, because it presents amortized cost: taken individually, the cost of entering loop \( wh6 \) once is at most \( 2 \times (x + y) \) (in terms of \( p1 \)'s input parameters). But the loop can be entered \( x \) times and still its total cost is at most \( 2 \times (x + y) \) and not \( 2 \times (x + y) \times x \) as one might expect. This is even more relevant for lower bounds. Considered individually, the cost of \( wh3 \) can be 0 (if no iterations of the inner loop \( wh6 \) are executed) and the cost of \( wh9 \) can also be 0 (if the inner loop \( wh6 \) iterates until \( y \) reaches 0). However, the lower cost bound of \( wh3 \) followed by \( wh9 \) is \( \min(2, z) \times (x + y) \). We know of no other cost analysis method that can infer a precise lower bound of program 1.

As noted in [8], a key aspect to obtain precise bounds for programs with amortized cost is to take the final variable values into account. In our example, if we infer that the cost of \( wh3 \) and \( wh9 \) is \( 2 \times (x + y - y_0) \) and \( z \times (y_0) \), respectively (in the context of CE 1), we can cancel the positive and negative \( y_0 \) summand and obtain the upper and lower bounds reported in Fig. 1. Unfortunately, the approach of [8] is computationally expensive and does not scale to larger programs. We propose instead to represent cost by a combination of simple expressions and constraints (cost structures), where the inference of complex resource bounds is reduced to the solution of (relatively) small linear programming problems.

The contributions are: (1) A new cost representation (cost structure) that can represent complex polynomial upper and lower bounds (Sec. 3); and (2) techniques to infer cost structures of cost relations in terms of the initial and final values of the variables and compose them precisely (obtaining amortized cost) and efficiently (Secs. 4, 5); (3) the implementation of the analysis as part of an open source cost analysis tool CoFloCo \(^1\); (4) an extensive experimental evaluation for both upper and lower bounds comparing our tool with other cost analysis tools: KoAT [9], Loopus [25], C4B [10] and the previous version of CoFloCo [14] for upper bounds and PUBS [5] for lower bounds (Sec. 7).

\(^1\)https://github.com/aeflores/CoFloCo
2 Preliminaries

In this section, we formally define the concepts and conventions used in the rest of the paper. The symbol \( \tau \) represents a sequence of variables \( x_1, x_2, \ldots, x_n \) of any length. We represent the concatenation of \( \tau \) and \( \eta \) as \( \tau \eta \). A variable assignment \( \alpha : V \mapsto D \) maps variables from the set of variables \( V \) to elements of a domain \( D \). Let \( t \) be a term, \( \alpha(t) \) denotes the replacement of all the variables \( x \) in \( t \) by \( \alpha(x) \). The variable assignment \( \alpha|_V \) is the restriction of \( \alpha \) to the domain \( V \). A linear expression has the form \( l(\tau) := q_0 + q_1 \cdot x_1 + \cdots + q_n \cdot x_n \) where \( q_i \in \mathbb{Q} \) and \( x_1, x_2, \ldots, x_n \) are variables. A linear constraint over \( \tau \) is \( lc(\tau) := l(\tau) \geq 0 \) where \( l(\tau) \) is a linear expression. For readability we often express linear constraints as \( l_1 \leq l_2 \), \( l_1 = l_2 \) or \( l_1 \geq l_2 \). These can be easily transformed to the form above. A constraint set \( \varphi(\tau) \) is a conjunction of linear constraints \( lc_1(\tau) \wedge lc_2(\tau) \wedge \cdots \wedge lc_n(\tau) \). A constraint set \( \varphi(\tau) \) is satisfiable if there exists an assignment \( \alpha : V \mapsto \mathbb{Q} \) such that \( \varphi(\alpha(x)) \) is valid \((\alpha \text{ satisfies } \varphi(\tau))\). We say that \( \varphi(\tau) \Rightarrow \varphi'(\tau) \) if every assignment that satisfies \( \varphi(\tau) \) satisfies \( \varphi'(\tau) \) as well. Next, we define cost relations which are our abstract representation of programs:

**Definition 1 (Cost relation).** A cost relation \( C \) is a set of cost equations \( e := \langle C(\tau) = q + \sum_{i=1}^n D_i(\eta_i), \varphi(\tau \eta) \rangle \), where \( q \in \mathbb{Q} \); \( C \) and \( D_i \) are cost relation symbols; and \( \varphi(\tau \eta) \) is a constraint set that relates the variables on the left side \( C(\tau) \) and those in the \( D_i(\eta_i) \) where \( \eta = \eta_1 \eta_2 \cdots \eta_n \).

A cost equation (CE) \( \langle C(\tau) = q + \sum_{i=1}^n D_i(\eta_i), \varphi(\tau \eta) \rangle \) states that the cost of \( C(\tau) \) is \( q \) plus the sum of the costs of each \( D_i(\eta_i) \). The constraint set \( \varphi(\tau \eta) \) serves two purposes: it restricts the applicability of the equation with respect to the input variables \( \tau \) and it relates the variables \( \tau \) with each \( \eta_i \). One can view a CR \( C \) as a non-deterministic function that executes a cost equation in \( C \). Given a cost equation \( \langle C(\tau) = q + \sum_{i=1}^n D_i(\eta_i), \varphi(\tau \eta) \rangle \), \( C \) consumes \( q \) resources and calls the functions \( D_1, D_2, \ldots, D_n \).

2.1 Cost relation refinement

In this work, we do not consider arbitrary CRs but instead CRs that are the result of a control-flow refinement presented in [14]. This refinement produces a set of execution patterns (called chains and denoted \( ch \)) for each CR. These execution patterns are regular expressions of CE identifiers and represent all possible executions of the CR. The formal definition of chains is as follows:

**Definition 2 (Phase, Chain).** Let \( C \) be a cost relation. A phase \( (ph) \) can be:
1. one or more recursive CEs executed a positive number of times \((c_1 \lor \cdots \lor c_n)^+\) with \( c_i \in C \); or
2. a single (non-recursive) CE executed once \((c_i)\).

A chain \( (ch) \) is a sequence of phases \( ch := [ph_1 \cdot ph_2 \cdots ph_n] \) in \( C \). A chain can represent a terminating execution if \( ph_n \) contains a single non-recursive CE \((c_i)\) or a non-terminating execution if \( ph_n \) has the form \((c_1 \lor \cdots \lor c_n)^+\).

For instance, the CR \( wh6 \) contains two phases \((5)^+\) and \((4)\) (where a number \( n \) refers to CE \( n \) in Fig. 1). From these phases, we can have two chains ‘[4]’
1.1: \( p_1(x, y, z) = wh3((3.1 \lor 3.2)^+ 2)(x, y, x_o, y_o) + wh9 [6](y_o, z) \)
\( \{ x > 0, y > 0, z > 0, x_o = 0, y_o \leq 0 \} \)

1.2: \( p_1(x, y, z) = wh3((3.1 \lor 3.2)^+ 2)(x, y, x_o, y_o) + wh9 [7.1^+ 6](y_o, z) \)
\( \{ x > 0, y > 0, z > 0, x_o = 0, y_o > 0, x \leq y \leq y_o \} \)

3.1: \( wh3(x, y, x_o, y_o) = wh6 [4](y_1, y_2) + wh3(x', y', x_o, y_o) \)
\( \{ x > 0, x' = x - 1, y_1 = y + 1, y' = y_2, y_2 = y_1 \} \)

3.2: \( wh3(x, y, x_o, y_o) = wh6 [5^+ 4](y_1, y_2) + wh3(x', y', x_o, y_o) \)
\( \{ x > 0, x' = x - 1, y_1 = y + 1, y' = y_2, y_2 < y_1 \} \)

7.1: \( wh9(y, z) = wh12 [9^+ 8](0, z) + wh9(y', z) \)
\( \{ y \geq 1, y' = y - 1, z > 0 \} \)

**Fig. 2.** Refined cost equations from Program 1

and \( [5^+ 4] \)' that represent the case where the loop body is not executed \( [4] \)
and the case when it is executed a finite number of times \( [5^+ 4] \). In principle,
we could also have a non-terminating chain \( [5^+] \) but the refinement in [14]
discards non-terminating chains that can be proved terminating. Any external
reference to a CR \( C_1 \) from another CR \( C_2 \) is annotated with a chain: \( C_1 ch \)
that determines which CEs will be applied and in which order. In this manner,
the cost equations are refined. CE 3 from Fig. 1 becomes CE 3.1 and 3.2 in
Fig. 2 which contain annotated references to \( wh6 \) with the corresponding chains
\( wh6 [4](y_1, y_2) \) and \( wh6 [5^+ 4](y_1, y_2) \). Similarly, CE 1 becomes 1.1 and 1.2 in Fig. 2
and CE 7 becomes 7.1. The constraint sets of the refined equations also contain
a summary of the behavior of these references (the bold constraints in Fig. 2).
Note that the refinement discards unfeasible references. For example, CR \( wh9 \)
does not have a reference to \( wh12 [8] \) because \( z \) is guaranteed to be positive.

The refined CRs can be ordered in a sequence \( \langle C_1, C_2 \ldots C_n \rangle \), in which a
cost equation of \( C_i \) can contain at most one recursive reference to \( C_i \) and any
number of references to \( C_j, j \geq i \) annotated with chains of \( C_j \). Its general form
is: \( \langle C_i(\overline{\alpha}) = q + \sum_{i=1}^n Dch_i(\overline{\alpha}) + C_i(\overline{\alpha}'), \varphi(\overline{xx'}) \rangle \) where \( D \in \{ C_{i+1}, \ldots, C_n \} \) if it
is recursive or without the summand \( +C_i(\overline{\alpha}') \) if it is non-recursive.

Most programs can be expressed as refined CRs [14]. The only current limita-
tion of this approach is the analysis of CRs with multiple recursion (when a
CE contains more than one recursive recursion).

### 2.2 Refined cost relation semantics

Cost relations can be evaluated to a cost with respect to a variable assignment
\( \alpha: V \rightarrow \mathbb{Q} \). We define the evaluation relation \( \|$ for refined CRs. This relation is
not meant to be executed but rather to serve as a formal definition of the cost
of CRs. Fig. 3 contains the rules for evaluating chains, phases and CEs.

We write a non-recursive CE \( \langle C(\overline{\alpha}) = k_0 + \sum_{i=1}^n Dch_i(\overline{\alpha}), \varphi(\overline{xx'y}) \rangle \) as nrc(\( \overline{\alpha} \)).
Rule (N\( \text{ON-RECURSIVE CE} \) extends the assignment \( \alpha \) to \( \alpha' \) such that it is
defined for \( \overline{y} \) and the constraint set of the CE is valid \( \varphi(\alpha'(\overline{yy})) \). The cost
of nrc(\( \overline{\alpha} \)) with variable assignment \( \alpha \) is the sum of the costs of the evaluations
of the chains referenced by nrc(\( \overline{\alpha} \)) plus \( k_0 \). A recursive CE \( \langle C(\overline{\alpha}) =
\) \( k_0 + \sum_{i=1}^n Dch_i(\overline{\alpha}) + C(\overline{\alpha}'), \varphi(\overline{xx'y}) \rangle \) is written rc(\( \overline{xx'} \)). Because a recursive CE
always appears within a recursive phase \((c_1 \lor \cdots \lor c_n)^+\), we will not include the recursive reference during its evaluation. That is, (RECURSIVE CE) does not add the cost of the recursive reference. That will be instead considered in the evaluation of the phase. Hence, (RECURSIVE CE) and (NON-RECURSIVE CE) are almost identical, but we include the variables \(\overline{x}\) of the recursive reference in the former so they can be matched with the initial variables of the next CE in the phase. Rules (REC PHASE) and (BASE PHASE) define the recursive evaluation of a phase. As before we include the variables of the last recursive reference \(\overline{x}\) in the phase representation \((c_1 \lor \cdots \lor c_n)^+ (\overline{x})\) so they can be matched with the initial variables of the next phase in the chain. Finally, the evaluation of a chain is the sum of the evaluations of its phases. If the chain is terminating, \(\text{ph}_n\) will be \((nrc(\overline{x}))\) and the sequence of variables \(\overline{x}_{n+1}\) will be empty. If the chain is non-terminating, \(\text{ph}_n\) will be \((c_1 \lor \cdots \lor c_n)^+\) and \(\overline{x}_{n+1}\) will be undefined.

We follow the same evaluation structure to compute bounds. We also compute bounds that depend on the variables of the recursive references for CEs \((\overline{x})\) and for phases \((\overline{x})\). This might seem unnecessary at first, but it allows us to compute precise bounds in a modular way. Consider the chain \(\{5^+4\}\) of CR wh6. We want to obtain the precise (upper and lower) bound \(2(y - y_0)\) but when we consider the phase \((5)^+\), we do not have any information about how \(y_0\) relates to \(y\) (which is contained in CE 4). Instead, we infer the cost of \((5)^+\) as \(2(y - y_f)\), where \(y_f\) is the value of \(y\) in the last recursive reference of \((5)^+\). Later we combine this bound with the information of CE 4 \(\{y = y_o\}\) to obtain \(2(y - y_0)\).

### 3 Cost Structures

In order to obtain upper and lower bounds, we developed a symbolic cost representation that can represent the costs of chains, phases or CEs. We call this cost representation cost structure.

We define cost structures as combinations of linear expressions in such a way that they can be inferred and composed by merely solving problems over sets of linear constraints. Instead of a single complex expression, we use simple linear
Chain/Phase/CE(Variables): Cost Structure

- 

\[ \begin{array}{c}
\sum_{k=1}^{n} iv_k \triangleright SE \quad \text{where SE can be} \quad SE := l(\overline{iv}) \mid iv_i \mid \max(\overline{iv}) \mid \min(\overline{iv}).
\end{array} \]

- 

\[ FC(\pi) \text{ is a set of final constraints of the form } \sum_{k=1}^{n} iv_k \triangleright l(\overline{iv}) \mid l(\overline{iv}) := \max(l(\overline{iv}), 0) \quad \text{and } l(\overline{iv}) \text{ is a linear expression over the CR variables.} \]

Fig. 4. Some of the cost structures of Program 1

---

cost expressions \( E \) over intermediate variables \( iv \) and constraints that bind the intermediate variables to the variables of the CRs. We distinguish two kinds of constraints. non-final constraints \( IC \) that relate intermediate variables among each other and final constraints \( FC(\pi) \) that relate intermediate variables with the variables of the CRs \( \pi \). The formal definition of cost structures is as follows:

**Definition 3 (Cost Structure).** A cost structure is a tuple \( \langle E, IC, FC(\pi) \rangle \).

---

- \( E \) is the main cost expression and is a linear expression \( l(\overline{iv}) \) over intermediate variables. Intermediate variables always represent positive numbers.

- Let \( \triangleright \) be \( \leq \) or \( \geq \). \( IC \) is a set of non-final constraints of the form

\[ \sum_{k=1}^{n} iv_k \triangleright SE \quad \text{where SE can be} \quad SE := l(\overline{iv}) \mid iv_i \mid \max(\overline{iv}) \mid \min(\overline{iv}). \]

- \( FC(\pi) \) is a set of final constraints of the form \( \sum_{k=1}^{n} iv_k \triangleright l(\overline{iv}) \mid l(\overline{iv}) := \max(l(\overline{iv}), 0) \quad \text{and } l(\overline{iv}) \text{ is a linear expression over the CR variables.} \)

Even though the constraints in \( IC \) and \( FC(\pi) \) are relatively simple, we can express complex polynomial expressions by combining them. In Fig. 4 we have some of the cost structures of program 1 that will be obtained in the following sections (\( a = b \) stands for \( a \leq b \) and \( a \geq b \)). Thanks to the constraints we can represent both upper and lower bounds with a single cost structure. Moreover, we can have several constraints that bind the same intermediate variables and thus represent multiple bound candidates. Finally, having multiple \( iv \) on the left side of the constraints can represent a disjunction or choice. This is the case for \( iv_6 + iv_3 = |y + z| \) of chain \( [1, 2] \). The bigger \( iv_6 \) is, the smaller \( iv_3 \) becomes. This capability is key to obtain a non-trivial lower bound for program 1.

We infer cost structures incrementally. In a sequence of CRs \( \langle C_1, C_2 \ldots C_n \rangle \), we start with \( C_n \) and proceed backwards until \( C_1 \). For each \( C_i \) we compute the cost structures of the CEs first (Sec. 4), then of the phases (Sec. 5) and finally of the chains (Sec. 4). This way, at each step, the cost structures of all the components have already been computed and it suffices to compose them.

**Example 2.** The sequence of CRs in Program 1 is \( \langle p1, wh3, wh6, wh9, wh12 \rangle \). We start computing cost structures for \( wh12 \) and finish by computing cost structures for \( p1 \). For each CR, we compute cost structures for the CEs, the phases and the chains. Consider CR \( wh9 \) for instance. We compute the cost of CEs 7.1 and 6 first. These are \( \langle iv_1, \emptyset, \{ iv_1 = |z| \} \rangle \) which originates from its reference to
Definition 4 (Valid Cost Structure). Let $T(\overline{x})$ be a chain, phase or CE. The cost structure $(E, IC, FC(\overline{x}))$ is valid for $T$ if for every $\langle \alpha, T(\overline{x}) \rangle \Downarrow k$, there exists an extension of $\alpha$ denoted $\alpha'$ ($\alpha'|_{\overline{x}} = \alpha$) that assigns all the intermediate variables such that $\alpha'(IC \land FC(\overline{x}))$ is valid and $\alpha'(E) = k$.

A valid cost structure of $T(\overline{x})$ can be evaluated to any cost $k$ s.t. $\langle \alpha, T(\overline{x}) \rangle \Downarrow k$. Given a valid cost structure $(E, IC, FC(\overline{x}))$, we can easily obtain closed-form upper/lower bounds such as the ones given in Fig. 1 by maximizing/minimizing the main cost expression $E$ according to the constraints $IC$ and $FC(\overline{x})$. This is done by incrementally substituting intermediate variables in $E$ for their upper/lower bounds defined in the constraints until $E$ does not contain any intermediate variable. The details on how this process is implemented can be found in App. C.

Example 3. The lower bound of chain $[1,2]$ is computed as follows: We start from the main cost expression $iv_2 + 2iv_6$ and we minimize each $iv$ using the constraints: (1) $iv_2 \geq iv_3 * iv_4$ (2) $iv_4 \geq |z|$ and (3) $iv_3 + iv_6 \geq y + x$:

$iv_2 + 2iv_6 \geq (1) iv_3 * iv_4 + (2) iv_4 \geq (3) iv_3 + iv_6 \geq y + x$.

Example 4. The upper bound of chain $[1,2]$ is computed as follows: We start from the main cost expression $iv_2 + 2iv_6$ and we maximize each $iv$ using the constraints: (1) $iv_2 \geq iv_3 * iv_4$ (2) $iv_4 \geq |z|$ and (3) $iv_3 + iv_6 \geq y + x$.

$iv_2 + 2iv_6 \geq (1) iv_3 * iv_4 + (2) iv_4 \geq (3) iv_3 + iv_6 \geq y + x$.

4 Cost Structures of Cost Equations and Chains

We want to obtain a valid cost structure of a recursive CE $rc(\overline{x})r := \langle C(\overline{x}) \rangle = k_0 + \sum_{i=1}^{n} Dch_i(\overline{y}) + C(\overline{x}), \phi(\overline{x} | \overline{y}) \rangle$ (the non-recursive case is analogous). Let $k_i$ be the cost of $ch_i(\overline{y})$, the cost of $rc(\overline{x})r$ is $k_0 + \sum_{i=1}^{n} k_i$ (See Fig. 3). Similarly, we can obtain a valid cost structure for $rc(\overline{x})r$ by composing the cost structures of each $ch_i(\overline{y})$.

Remark 1. Let $\langle E_{ch_i}, IC_{ch_i}, FC_{ch_i}(\overline{y}) \rangle$ be a valid cost structure of $ch_i(\overline{y})$, the following cost structure is valid for $rc(\overline{x})r$:

$\langle k_0 + \sum_{i=1}^{n} E_{ch_i}, \bigcup_{i=1}^{n} (IC_{ch_i}), \bigcup_{i=1}^{n} (FC_{ch_i}(\overline{y})) \rangle$
We add the main cost expressions $E_{ch}$, plus $k_0$ and join the constraint sets $IC_{ch}$, and $FC_{ch}(\overline{y})$. Note that in the base case (i.e. when $n = 0$), the resulting cost structure is simply $(k_0, \emptyset, \emptyset)$. Unfortunately, the final constraints in $\bigcup_{i=1}^{n}(FC_{ch}(\overline{y}))$ contain variables other than $\overline{xx'}$ and have to be transformed to obtain a cost structure that only contains CR variables in $\overline{xx'}$.

Transformation of final constraints We perform this transformation with the help of the CE’s constraint set $\varphi(\overline{xx'y})$. Recall that final constraints are of an almost linear form ($\sum_{k=1}^{m} iv_k \propto l(\overline{y})$). If we guarantee that $l(\overline{y})$ is non-negative ($\varphi(\overline{xx'y}) \Rightarrow l(\overline{y}) \geq 0$), we can simply use the linear constraint $\sum_{k=1}^{m} iv_k \propto l(\overline{y})$. Let $FC^+$ be the set of all constraints obtained thus from $\bigcup_{i=1}^{n}(FC_{ch}(\overline{y}))$. We perform (Fourier-Motzkin) quantifier elimination on $\exists \overline{y}.(FC^+ \land \varphi(\overline{xx'y}))$ and obtain a constraint set that relates directly the intermediate variables of $FC^+$ with $\overline{xx'}$. We can then extract syntactically from the resulting constraint set new final constraints in terms of $\overline{xx'}$.

Example 4. We combine the cost of chains $[(3.1 \lor 3.2)^+2]$ and $[7.1^+6]$ from Fig. 4 into that of CE 1.2, instantiated according to CE 1.2 with variables $(x, y, z)$, respectively. The resulting expression is: $\langle iv_2 + 2iv_6, \{iv_2 = iv_3 \cdot iv_4, iv_6 = |y - y_o + x|, iv_3 = |y_o|, iv_4 = |z|\} \rangle$. This is the cost structure of [1.2] in Fig. 4 except for the final constraints which need to be transformed. The constraint set of CE 1.2 from Fig. 2 ($\varphi_{1,2}$) guarantees that $y - y_o + x, y_o$ and $z$ are non-negative. Therefore, we generate a constraint set $FC^+ = \{iv_6 = y - y_o + x, iv_3 = y_o, iv_4 = z\}$ and perform quantifier elimination over $\exists x_o, y_o, (FC^+ \land \varphi_{1,2})$. This results in $\{iv_6 + iv_3 = y + x, iv_4 = z, x > 0, y > 0, z > 0\}$ from which we syntactically extract the constraints $iv_3 + iv_6 = |y + x|$ and $iv_4 = |z|$. This procedure allows us to find dependencies among constraints $(iv_6 = y - y_o + x$ and $iv_3 = y_o)$ and merge them precisely (into $iv_3 + iv_6 = |y + x|$).

We transform the rest of the final constraints, i.e. the ones that cannot be guaranteed to be positive, one by one. Let $\sum_{k=1}^{m} iv_k \propto |l(\overline{y})|$ be a constraint, if we find $l'(\overline{xx'})$ such that $\varphi(\overline{xx'y}) \Rightarrow l(\overline{y}) \propto l'(\overline{xx'})$, then we have that $\sum_{k=1}^{m} iv_k \propto |l'(\overline{xx'})|$ holds as well.

Chains The case of computing a cost structure $(E_{ch}, IC_{ch}, FC_{ch}(\overline{y}))$ of a chain $ch = [ph_1 \cdot ph_2 \cdots \cdot ph_n]$ is analogous. Let $(E_{ph_i}, IC_{ph_i}, FC_{ph_i}(\overline{xx'+1})$ be the cost structure of $ph_i(\overline{xx'+1})$, and add the main cost expressions and join the constraint sets to obtain: $\langle \sum_{i=1}^{n} E_{ph_i}, \bigcup_{i=1}^{n}(IC_{ph_i}), \bigcup_{i=1}^{n}(FC_{ph_i}(\overline{xx'+1})) \rangle$. We transform the final constraints $FC_{ph_i}(\overline{xx'+1})$ to express them in terms of the initial variables $\overline{y}$ as above. But this time we perform the transformation incrementally. We transform first $FC_{ph_n}(\overline{y})$ and $FC_{ph_n-1}(\overline{y})$ in terms of $\overline{y}$. Then, we transform the result together with $FC_{ph_{n-2}}(\overline{y})$ in terms of $\overline{y}$ and so on until we reach the first phase of the chain. In each step the constraint set used is $\varphi_{ph_i}(\overline{xx'+1})$ which is a summary of the behaviors of $ph_i, \cdots, ph_n$.

\footnote{This can be easily seen by distinguishing cases $(l(\overline{y}) \geq 0$ and $l(\overline{y}) \leq 0$).}
5 Cost Structures of Phases

Let $ph = (c_1 \lor \cdots \lor c_n)^+$ be a phase. Our objective is to compute a valid cost structure $\langle E_{ph}, IC_{ph}, FC_{ph}(\overline{x,y}) \rangle$ for the phase $ph$. Such a cost structure must be expressed in terms of initial values of the variables ($\overline{x,y}$) and the values of the variables in the last recursive call of the phase ($\overline{x,y}$) and must represent the sum of all the evaluations of $c_i \in ph$ (according to the semantics Fig. 3). For each evaluation of $c_i$, we can define an instantiation of its cost structure.

**Definition 5 (Cost Structure Instances).** Let $\langle E_{c_i}, IC_{c_i}, FC_{c_i}(\overline{x,y}) \rangle$ be a valid cost structure of $c_i$ and let $\#c_i$ be the number of times $c_i$ is evaluated in $ph$. $\langle E_{c_i,j}, IC_{c_i,j}, FC_{c_i,j}(\overline{x_{c_i,j}}) \rangle$ represents the cost structure instance of the $j$-th CE evaluation of $c_i$ for $1 \leq j \leq \#c_i$. That is, the cost structure of $c_i$ instantiated with the variables corresponding to the $j$-th CE evaluation of $c_i$: $x_{c_i,j}^i$.

**Remark 2.** The total cost of a phase is the sum of all the cost structure instances for $1 \leq j \leq \#c_i$ and for all $c_i \in ph$:

$$\left( \sum_{i=1}^{n} \sum_{j=1}^{\#c_i} E_{c_i,j} \right) + \left( \bigcup_{i=1}^{n} \bigcup_{j=1}^{\#c_i} (IC_{c_i,j}) \right) + \left( \bigcup_{i=1}^{n} \bigcup_{j=1}^{\#c_i} (FC_{c_i,j}(\overline{x_{c_i,j}})) \right)$$

Based on this, we generate a cost structure $\langle E_{ph}, IC_{ph}, FC_{ph}(\overline{x,y}) \rangle$ in three steps: (1) we transform the expression $\sum_{i=1}^{n} \sum_{j=1}^{\#c_i} E_{c_i,j}$ into a valid main cost expression $E_{ph}$; (2) we generate non-final constraints $IC_{ph}$ using the CEs’ non-final constraints $IC_{c_i}$ (in Sec. 5.1); and (3) we generate final constraints $FC_{ph}(\overline{x,y})$ using the CEs’ final constraints $FC_{c_i}(\overline{x_{c_i}})$ and the CE definitions (in Sec. 5.2).

In order to transform $\sum_{i=1}^{n} \sum_{j=1}^{\#c_i} E_{c_i,j}$ into a valid cost expression $E_{ph}$, we have to remove the sums over the unknowns $\#c_i$. For this purpose, we define the following new intermediate variables:

**Definition 6 (Sum intermediate variables).** Let $iv$ be an intermediate variable in $\langle E_{c_i}, IC_{c_i}, FC_{c_i}(\overline{x,y}) \rangle$. The intermediate variable $smiv := \sum_{i=1}^{\#c_i} iv_j$ is the sum of all instances of $iv$ in the different evaluations of $c_i$ in the phase.

Now, we can reformulate each $\sum_{j=1}^{\#c_i} E_{c_i,j}$ into a linear expression in terms of $smiv$. Let $E_{c_i} := q_0 + q_1 \cdot iv_1 + \cdots + q_m \cdot iv_m$, we have that $\sum_{j=1}^{\#c_i} E_{c_i,j} = q_0 \cdot \#c_i + q_1 \cdot smiv + \cdots + q_m \cdot smiv_m$ (where $\#c_i$ is also an intermediate variable). If we do this transformation for each $i$ in $\sum_{i=1}^{n} \sum_{j=1}^{\#c_i} E_{c_i,j}$, we obtain a valid cost expression for the phase $E_{ph}$.

**Example 5.** Consider phase $(3.1 \lor 3.2)^+$. Let $E_{3.1} = 0$ and $E_{3.2} = 2iv_5$. The main cost expression of the phase is $E_{(3.1 \lor 3.2)^+} = \sum_{j=1}^{\#c_{3.1}} 0 + \sum_{j=1}^{\#c_{3.2}} 2iv_{5_j} = 2smiv_5$ (where $smiv_5$ corresponds to $iv_6$ in Fig. 4).
5.1 Transforming Non-final Constraints

In this section we want to generate a new set of non-final constraints $IC_{ph}$ that bind the new intermediate variables (smiv) that appear in our main cost expression $E_{ph}$.

We iterate over the non-final constraints of each $IC_{c_i}$ for $c_i \in ph$. For each constraint $\sum_{k=1}^{m} iv_k \preceq SE \in IC_{c_i}$, we sum up all its instances $\sum_{j=1}^{\#c_i} \sum_{k=1}^{m} iv_{k,j} \preceq \sum_{j=1}^{\#c_i} SE_j$ and reformulate the constraint using smiv variables. We reformulate the left-hand side directly: $\sum_{j=1}^{\#c_i} \sum_{k=1}^{m} iv_{k,j} = \sum_{k=1}^{m} smiv_k$

However, the right-hand side of the constraints might contain sums over non-linear expressions. These sums cannot be reformulated only in terms of Sum variables. Therefore, we introduce a new kind of intermediate variable:

**Definition 7 (Max/Min intermediate variables).** The variables $[iv] := \max_{1 \leq j \leq \#c_i} (iv_j)$ and $[iv] := \min_{1 \leq j \leq \#c_i} (iv_j)$ are the maximum and minimum value that an instance $iv_j$ of $iv$ can take in a evaluation of $c_i$ in $ph$.

With the help of this new kind of variables we can reformulate the right hand side of the expression: $\sum_{j=1}^{\#c_i} SE_j$. We distinguish cases for each possible $SE$:

- $SE := q_0 + q_1 * iv_1 + \cdots + q_m * iv_m$:
  - We have that $\sum_{j=1}^{\#c_i} SE_j = q_0 * \#c_i + q_1 * smiv_1 + \cdots + q_m * smiv_m$.
- $SE := iv_k * iv_p$: We approximate $\sum_{j=1}^{\#c_i} SE_j$ with the help of $[iv]_p$ or $[iv]_p$ depending on whether $\preceq$ is $\leq$ or $\geq$:
  - $\sum_{j=1}^{\#c_i} SE_j \leq smiv_k * [iv]_p$ and $\sum_{j=1}^{\#c_i} SE_j \geq smiv_k * [iv]_p$.
- $SE := max([iv]) or min([iv])$: We reduce this to the previous case. We reformulate $SE$ as $1 * SE$ and substitute each factor by a fresh intermediate variable: $iv_k * iv_p$. Then, we add the constraints $iv_k \preceq 1$ and $iv_p \preceq SE$ to $IC_{c_i}$ so they are later transformed. This way, $smiv_p$ is not generated ($[iv]_p$ or $[iv]_p$ will be generated instead) and we do not have to compute $\sum_{j=1}^{\#c_i} SE_j$.

In the generated constraints new variables of the form $[iv]$ and $[iv]$ might have been introduced that also need to be bound. We iterate over the constraints in $IC_{c_i}$ from $c_i \in ph$ again to generate constraints over $[iv]$ and $[iv]$ variables.

Let $iv \leq SE \in IC_{c_i}$ (the $\geq$ case is symmetric). We distinguish cases for $SE$:

- $SE := q_0 + q_1 * iv_1 + \cdots + q_m * iv_m$: Let $V_k := [iv]_k$ if $q_k \geq 0$ or $V_k := [iv]_k$ if $q_k < 0$. We generate $[iv] \leq q_0 + q_1 * V_1 + \cdots + q_m * V_m$.
- $SE := iv_k * iv_p$: We generate $[iv] \leq [iv]_k * [iv]_p$.
- $SE := max(iv_1 \cdots iv_n)$: We generate $[iv] \leq max([iv]_1 \cdots [iv]_n)$.
- $SE := min(iv_1 \cdots iv_n)$: We generate $[iv] \leq [iv]_k$ (for $1 \leq k \leq n$).

All these newly generated constraints form the non-final constraint set $IC_{ph}$.

---

3 We could also approximate to $[iv]_k * smiv_p$ and $[iv]_k * smiv_p$ but in general the chosen approximation works better. The variable $iv_k$ usually represents an outer loop and $iv_p$ and inner loop (See basic product strategy in Sec. 5.2).

4 This transformation is not valid for constraints with multiple variables on the left side. The constraints with $\leq$ can be split ($\sum_{k=1}^{m} iv_k \leq SE$ implies $iv_k \leq SE$ for $1 \leq k \leq m$). But this is not the case for the constraints with $\geq$.
5.2 Transforming Final Constraints

Previously, we computed a main cost expression \( E_{ph} \) and a set of non-final constraints \( IC_{ph} \) for a phase \( ph = (c_1 \lor \cdots \lor c_n) \). We complete the phase’s cost structure with a set of final constraints \( FC_{ph}(x_s x_f) \) (and possibly additional non-final constraints) that bind the intermediate variables of \( E_{ph} \) and \( IC_{ph} \). We propose the following algorithm:

**Algorithm initialization** For each \( c_i \) with cost structure \( \langle E_{c_i} , IC_{c_i} , FC_{c_i} (x(x')) \rangle \) the algorithm maintains two sets of pending constraints:
1. \( Psums^{c_i} \) is initialized with the constraints \( \sum_{k=1}^m iv_k \bowtie \|l(x(x'))\| \in FC_{c_i}(x(x')) \) such that some \( smiv_k \) appear in our phase cost structure (in \( E_{ph} \) or \( IC_{ph} \)) and \( iv_{it} \leq 1 \) and \( iv_{it} \geq 1 \) if \#\( c_i \) appears in our phase cost structure. The variable \( iv_{it} \) represents the number of times \( c_i \) is evaluated and \( smiv_{it} = \#c_i \).
2. \( Pms^{c_i} \) is initialized with the constraints \( iv \bowtie \|l(x(x'))\| \in FC_{c_i}(x(x')) \) such that \( [iv] \) or \( [iv] \) appear in our phase cost structure.

**Algorithm** At each step, the algorithm removes one constraint from one of the pending sets and applies one or several strategies to the removed constraint. A strategy generates new constraints (final or non-final) for the phase’s cost structure; they are added to the sets \( IC_{ph} \) or \( FC_{ph}(x_s x_f) \). A strategy can also add additional pending constraints to the sets \( Psums^{c_i} \) or \( Pms^{c_i} \) to be processed later. The algorithm repeats the process until \( Psums^{c_i} \) and \( Pms^{c_i} \) are empty or all the intermediate variables in \( E_{ph} \) and \( IC_{ph} \) are bound by constraints.

In principle, the algorithm can finish without generating constraints for all intermediate variables. For instance, if the cost of the phase is actually infinite. It can also not terminate if new constraints keep being added to the pending sets indefinitely. This does not happen often in practice and we can always stop the computation after a number of steps. We propose the following strategies:

**Inductive Sum Strategy** Let \( \sum_{k=1}^m iv_k \bowtie \|l(x(x'))\| \in Psums^{c_i} \), the strategy will try to find a linear expression that approximates the sum \( \sum_{j=1}^{\#c_i} \|l(x_{c,j} x'_{c,j})\| \) in terms of the initial and final variables of the phase \( (x_s x_f) \).

Let us consider first the simple case where \( c_j \) is the only CE in the phase. The strategy uses the CE’s constraint set \( \varphi_j(x_{c,j} x'_{c,j}) \) and Farkas’ Lemma to generate a candidate linear expression \( cd(\tau) \) such that \( \varphi_j(x_{c,j} x'_{c,j}) \Rightarrow (\|l(x(x'))\| \bowtie cd(\tau)) \geq 0 \). If a candidate \( cd(\tau) \) is found, we have:

\[
\sum_{j=1}^{\#c_i} \|l(x_{c,j} x'_{c,j})\| \bowtie \sum_{j=1}^{\#c_i} (cd(x_{c,j}) - cd(x'_{c,j})) = cd(x_s) - cd(x_f)
\]

This is because each intermediate -\( cd(x_{c,j}) \) and \( cd(x'_{c,j+1}) \) cancel each other \((cd(x'_{c,j}) = cd(x_{c,j+1}) )\). Therefore, the constraint \( \sum_{k=1}^m smiv_k \bowtie |cd(\tau_s) - cd(\tau_f)| \) is valid and can be added to \( FC_{ph}(x_s x_f) \).

**Example 6.** This is the case of phase \( 9^+(i_s, z_s, i_f, z_f) \) with \( Psums^{9} = \{iv_{it_0} \leq 1, iv_{it_0} \geq 1\} \). The strategy generates the candidate -\( i \) and the final constraints
Condition when ≻ is ≤ | Condition when ≻ is ≥ | Defines
\[ Cnt \left( \sum_{k=1}^{m} iv_k \varpropto |l'(x^x)| \right) \in Psums^{ce} \land |l'(x^x)| \varpropto cd(x^x) - cd(x^x) \geq 0 \] \[ cnt_{ce} = \sum_{k=1}^{m} smiv_k \]
\[ Dc \] \[ 0 \leq dc_{ce}(x^x) = cd(x^x) - cd(x^x) \]
\[ Ic \] \[ ic_{ce}(x^x) \geq cd(x^x) - cd(x^x) \]
\[ Rst \] \[ cd(x^x) \varpropto rst(x^x) \]
\[ w_{dce} = |dc_{ce}(x^x)| \]
\[ w_{ic_{ce}} = |ic_{ce}(x^x)| \]
\[ w_{rst_{ce}} = |rst_{ce}(x^x)| \]

Fig. 5. Classes of CE \( ce \) w.r.t a candidate \( cd(x^x) \), their condition and defined term

\( smiv_{it_{ce}} \leq |i_f - i_s| \) and \( smiv_{it_{ce}} \geq |i_f - i_s| \). Later \( |i_f - i_s| \) will become \( |z - i| \) in \([9 + 8]\) and \( |z| \) in CE 7.1. The variable \( smiv_{it_{ce}} \) corresponds to \( iv_1 \) in Fig. 4.

If the phase contains other CEs \( ce \) \( (e \neq i) \), we have to take their behavior into account. E.g. suppose that we have another \( ce \) \( (e \neq i) \) that increments our candidate by two \( (\varphi_e(x^yx^y) \Rightarrow cd(x^x) = cd(x^x) + 2) \). Let \#\( ce \) be the number of evaluations of \( ce \), the sum is \( \sum_{j=1}^{\#_{ce}} cd(x^x_{ce_j}) = cd(x^x) - cd(x^x) + 2 \#_{ce} \).

That is, the sum computed for the simple case \( cd(x^x) - cd(x^x) \) plus the sum of all the increments to the candidate \( 2 \#_{ce} \) effect of CE \( ce \).

In the general case, the strategy generates a candidate (using \( ce \) constraint set \( \varphi_{ce}(x^y) \) and Farkas’ Lemma as before); it classifies the CEs of the phase \( ce \in ph \) (including \( ce \)) according to their effect on the candidate; and it uses this classification to generate constraints that take these effects into account.

**Cost Equation Classification** Each class has a condition and it defines a (linear) term (See Fig. 5). In order to classify a CE \( ce \) into a class, its condition has to be implied by the corresponding CE’s constraint set \( \varphi_{ce}(x^y) \). This implication can be verified and the unknown linear expressions \( dc_{ce}(x^x) \), \( ic_{ce}(x^x) \), or \( rst_{ce}(x^x) \) (For the classes \( Dc \), \( Ic \) and \( Rst \) respectively) can be inferred using Farkas’ Lemma. The considered classes in this strategy are:\[^5^]

- **Cnt**: \( ce \in Cnt \) if there is a constraint \( \sum_{k=1}^{m} iv_k \varpropto |l'(x^x)| \) that can also be bound by the candidate: \( |l'(x^x)| \varpropto cd(x^x) - cd(x^x) \). We can incorporate \( \sum_{k=1}^{m} smiv_k \) to the left hand side of our constraint. We define \( cnt_{ce} := \sum_{k=1}^{m} smiv_k \) as a shorthand. Note that \( ce \), whose constraint was used to generate the candidate, trivially satisfies the condition and thus \( ce \in Cnt \).
- **Dc**: \( ce \in Dc \) if in each evaluation of \( ce \) the candidate is decremented by at least \( dc_{ce}(x^x) \) (or at most \( dc_{ce}(x^x) \) if \( \varpropto \) is \( \geq \)). We assign a fresh intermediate variable to this amount \( iv_{dce} := |dc_{ce}(x^x)| \). To generate a valid constraint, we will subtract the sum of all those decrements i.e. \( smiv_{dce} \).
- **Ic**: \( ce \in Ic \) if in each evaluation of \( ce \) the candidate is incremented by at most \( ic_{ce}(x^x) \) (or at least \( ic_{ce}(x^x) \) if \( \varpropto \) is \( \geq \)). As before, we assign a fresh intermediate variable to that amount \( iv_{ic_{ce}} := |ic_{ce}(x^x)| \). To generate a valid constraint, we will add the sum of all those increments i.e. \( smiv_{ic_{ce}} \).

\[^5^\) The class \( Rst \) will be used and explained in the Max-Min strategy.
Lemma 1. Let \( \triangleright \) be the reverse of \( \triangleright \) (e.g. \( \geq \) if \( \triangleright = \leq \)). If we classify every \( c_e \in ph \) into \( Cnt \), \( Ic \) or \( Dc \) w.r.t. \( cd(\overline{x}) \), the following constraints are valid:

\[
\sum_{c_e \in Cnt} cnt_e \triangleright iv_{cd+} - iv_{cd-} + \sum_{c_e \in Ic} smiv_{ic_e} - \sum_{c_e \in Dc} smiv_{dc_e},
\]

\[
iv_{cd+} \triangleright |cd(\overline{x}) - cd(\overline{y})|, \quad iv_{cd-} \triangleright | - cd(\overline{x}) + cd(\overline{y})|.
\]

These are the constraints generated by the Inductive Sum strategy. Note that \( iv_{cd+} \) and \( -iv_{cd-} \) represent the positive and negative part of \( cd(\overline{x}) - cd(\overline{y}) \). The constraints bind the sum of all \( smiv \) in \( cnt_e \) (for each \( c_e \in Cnt \)) to \( cd(\overline{x}) - cd(\overline{y}) \) plus all the increments \( \sum_{c_e \in Ic} smiv_{ic_e} \) minus all the decrements \( \sum_{c_e \in Dc} smiv_{dc_e} \).

If \( Ic \) is empty, \( cd(\overline{x}) - cd(\overline{y}) \) is guaranteed to be positive (the candidate is never incremented) and we can eliminate the summand \( -iv_{cd-} \) (and its corresponding constraint \( iv_{cd-} \triangleright | - cd(\overline{x}) + cd(\overline{y})| \)).

Finally, the strategy adds constraints for the new intermediate variables \( iv_{ic_e} \) and \( iv_{dc_e} \) to the pending sets so their sums \( smiv_{ic_e} \) and \( smiv_{dc_e} \) are bound afterwards: \( iv_{ic_e} \triangleright |ic(\overline{x}^c)| \) is added to \( Psums^{c_e} \) for each \( c_e \in Ic \), and \( iv_{dc_e} \triangleright |dc(\overline{x}^c)| \) is added to \( Psums^{c_e} \) for each \( c_e \in Dc \).

Example 7. In phase \((3.1 \lor 3.2)^+\) we have \( iv_5 \leq |y - y' + 1| \in Psums^{3,2}. \) A valid candidate is \( y + x \). The CEAs are classified as follows: CE 3.2 \( \in Cnt \) because it has generated the candidate \((cnt_{3,2} := smiv_5)\); and CE 3.1 \( \in Dc \) because \( y + x \) decreases in CE 3.1 by \( dc_{3,1} = 0 \). The generated constraints are: \( smiv_5 \leq iv_{cd+} - iv_{cd-} - smiv_{dc} \), \( iv_{cd+} \leq |(y_s + x_s) - (y_f + x_f)| \) and \( iv_{cd+} \leq |(y_s + x_s) + (y_f + x_f)| \). However, given that \( Ic \) is empty and \( dc_{3,1} = 0 \), we can simplify them to a single constraint: \( smiv_5 \leq |(y_s + x_s) - (y_f + x_f)| \) (where \( smiv_5 \) is \( iv_6 \) in Fig. 4).

Example 8. The class \( Cnt \) allows us to bind sum variables of different \( c_i \) under a single constraint. For instance, if we had \( 6 \ iv_{it_{3,1}} \geq 1 \in Psums^{3,1} \) and \( 6 \ iv_{it_{3,2}} \geq 1 \in Psums^{9,2} \), the expression \( x \) would be a valid candidate with the classification \( Cnt = \{3.1, 3.2\} \) with \( cnt_{3,1} := smiv_{it_{3,1}} \) and \( cnt_{3,2} := smiv_{it_{3,2}} \). The strategy would generate (the simplified) constraint \( smiv_{it_{3,1}} + smiv_{it_{3,2}} \geq |x_s - x_f| \) which is equivalent to \( \#c_{3,1} + \#c_{3,2} \geq |x_s - x_f| \) and represents that \( wh_3 \) iterates at least \( |x_s - x_f| \) times. Without \( Cnt \), we would fail to obtain a non-trivial lower bound for \( \#c_{3,1} \) or \( \#c_{3,2} \) as they can both be 0 (if considered individually).

**Basic Product Strategy** Often, given a constraint \( \sum_{k=1}^{m} iv_k \triangleright |l(\overline{x}x)| \in Psums^{c_e} \), it is impossible to infer a linear expression representing \( \sum_{j=1}^{n} |l(\overline{x}_jx'_j)| \).

Example 9. Consider the cost computation of phase \( 7.1^+ \). We have a constraint \( iv_1 \leq |z| \in Psums^{7,1} \). The variable \( z \) does not change in CE 7.1 and \( \#c_{7,1} \) is at most \( y \) so \( \sum_{j=c_{7,1}} \leq |y| \leq |z| \) which is not linear. We can obtain this result by rewriting the constraint \( iv_1 \leq |z| \) as \( iv_1 \leq 1 * |z| \). Then, we generate the constraint \( smiv_1 \leq smiv_{it_{7,1}} * [iv]_{m(z)} \) (that corresponds to \( iv_2 \leq iv_3 * iv_4 \)).
in Fig. 4) and add $iv_{it_1} \leq 1$ to $Psums^{7,1}$ and $iv_{mz} \leq |z|$ to $Pms^{7,1}$. These constraints will be later processed by the strategies Inductive Sum and Max-Min respectively.

In general, given a $\sum_{k=1}^{m} iv_k \leq |l(x')| \in Psums^{c_i}$ where $l(x')$ is not a constant, the Basic Product strategy generates $\sum_{k=1}^{m} smiv_k \leq smiv_{it_1} \cdot |iv|_p$ and adds the pending constraints $iv_{it_1} \leq 1$ to $Psums^{c_i}$ and $iv_{p} \leq |l(x')|$ to $Pms^{c_i}$. This way, the strategy reduces a complex sum into a simpler sum and a max/minimization. The strategy proceeds analogously for constraints with $\geq$.

**Max-Min Strategy** This strategy deals with constraints $iv \bowtie |l(x')| \in Pms^{c_i}$ and its role is to generate constraints for Max $|iv|$ and Min $|iv|$ variables.

Similarly to the Inductive Sum strategy, it generates a candidate $cd(\overline{x})$ using the CE’s constraint set $\phi_i(x'y)$ and then it classifies the CEs in the phase according to their effect on the candidate. However, the condition used to generate the candidate is different since we want to bind a single instance of $l(x')$ instead of the sum of all its instances. Additionally, this strategy considers the class $Rst$ for the classification but not the class $Cnt$ (See Fig. 5). If $c_e \in Rst$ the candidate is reset to a value of at most $|rst_e(\overline{x})|$ (or at least $|rst_e(\overline{x})|$ if $\infty \geq$). A fresh intermediate variable is assigned to such reset value $iv_{rst_e} := |rst_e(\overline{x})|$.

**Lemma 2.** Let $iv \leq |l(x')| \in Pms^{c_i}$ and let $cd(\overline{x})$ be a candidate such that $\phi_i(x'y) \Rightarrow l(x') \leq cd(\overline{x})$. If we classify every $c_e \in ph$ into $Dc, Ic$ and $Rst$ with respect to $cd(\overline{x})$, the following constraints are valid:

$$[iv] \leq iv_{max} + \sum_{c_e \in Ic} smiv_{ic_e}, \quad iv_{max} \leq \max_{c_e \in Rst} (|iv|_{rst_e}, iv_{cd}), \quad iv_{cd} \leq |cd(\overline{x})|$$

These are the constraints generated by the Max-Min strategy. They bind $[iv]$ to the sum of all the increments $smiv_{ic_e}$ for $c_e \in Ic$ plus the maximum of all the maximum values that the resets can take $[iv]_{rst_e}$. This maximum also includes the candidate $cd(\overline{x})$ in case it is never reset.

Finally, the strategy adds the constraints $iv_{ic_e} \leq |ic_e(x')|$ to $Psums^{c_i}$ and $iv_{rst_e} \leq |rst_e(\overline{x})|$ to $Pms^{c_i}$ so $smiv_{ic_e}$ and $[iv]_{rst_e}$ are bound later. The strategy proceeds analogously for constraints with $\geq$ but it subtracts the decrements instead of adding the increments and takes the minimum of the resets $[iv]_{rst_e}$.

**Example 10.** In Example 9 we added $iv_{mz} \leq |z|$ to $Pms^{7,1}$ during the computation of the cost of $7.1^+$. Using the Max-Min strategy, we generate a candidate $z$ and classify CE 7.1 in $Dc$ with $dc_{7,1} := 0$ ($z$ is not modified in CE 7.1). The resulting (simplified) constraint is $|iv|_{mz} \leq |z|$ (which corresponds to $iv_{4} \leq |z_4|$ in Fig. 4).

To summarize, we transform the complex problem of obtaining a cost structure for a phase into a set of simpler problems: computation of sums, maximization, minimization of simple constraints. These smaller problems are solved incrementally through strategies that collaborate with each other by adding new constraints to the pending sets. The inference problems in the strategies can be
solved efficiently using Farkas’ Lemma as they only use the constraint set of one CE at a time. We provide two extra strategies in App. A to obtain upper bounds defined only in terms of $\vec{r}_s$ and to obtain better bounds for sums of expressions whose value varies in each iteration.

6 Soundness

**Theorem 1.** Let $T(\pi)$ be a chain, a phase or a CE. Then the cost structure $\langle E_T, IC_T, FC_T(\pi) \rangle$ obtained following the algorithms of Secs. 4 and 5 is valid.

**Proof sketch.** The cost structures in Remarks 1 and 2 result from applying the semantics rules to the cost structures of the components. The latter transformation of $E_{ph}$ is syntactic and the constraints generated in Secs. 4, 5.1 and 5.2 are implied (logical consequence) by the ones in Remarks 1 and 2 and the CEs’ constraint sets. Therefore, they can be added to the cost structures without compromising their validity. This implication for the constraints in Sec. 5.2 (Lemmas 1 and 2) is proved by induction on the number of CEs evaluations (App. E).

7 Related Work and Experiments

This work constitutes a significant improvement over previous techniques based on cost relations [3,5,7,14]. It builds on the refinement in [14] but presents a new approach for obtaining bounds that is much more powerful. We define a new cost structure representation that has more expressive power than the cost structures in [14] (it can represent lower bounds) and yet it can be inferred by applying simple rules to its constraints (See Secs. 5.1 and 5.2). In [14], ranking functions are used to bind the sums of constant expressions but the rest of the sums are obtained using (a variant of) the Basic Product strategy. Therefore, the system in [14] fails to obtain amortized costs except for simple cases. In particular, it fails to infer a linear upper bound for $wh3$. In the work [7] Farkas’ Lemma is used to obtain sums of linear expressions. However, it cannot infer bounds for expressions that are incremented or reset. Also, their generated bounds do not depend on the final variables of the phase and thus they are unable to obtain amortized cost. Finally, neither [7] nor [14] can obtain lower bounds.

Other approaches include KoAt [9] which obtains complexity bounds of integer programs by alternating size and bound analysis. Loopus [24,25] follows a similar schema using in a representation based on difference constraints and can compute amortized bounds. These ideas are present in how the cost is computed in this work. Instead of sizes and bounds there is a similar interplay between the the computation of constraints for $smiv$ and $\lceil iv \rceil/\lfloor iv \rfloor$ variables in Sec. 5. None of the mentioned work can compute lower bounds. It is worth to mention the SPEED project [18,19,20,27] where different cost analyses are proposed based on counter instrumentation [19], control flow refinement and progress invariants [18], proof rules [20] and the size-change abstraction [27]. These approaches are
not publicly available so we cannot perform an experimental comparison. However, our experimental evaluation includes all examples from these papers.

Another active line of research is about amortized cost analysis based on the potential method [10,21,22]. The authors of [21] present a type inference system that is able to obtain polynomial cost upper bounds for functional programs with data structures such as lists or trees. The key advantage of this analysis is its ability to reduce the polynomial cost inference to a linear programming problem (using type inference). In [22], they extend this analysis to deal with natural numbers. The system C4B [10] (to which we compare) adapts this approach for C programs with integers, but it can only infer linear bounds at the moment.

Based on the pioneering work of [26], several cost analyses based on recurrence relations were developed [11,12,23]. The authors of [5] present an analysis which extracts recurrence relations that approximate the cost of CRs and can later be solved by an external solver. Some of these approaches can also compute lower bounds but are unable to find cost bounds for loops with increments or resets or for programs that present amortized cost (such as program 1). Finally, the technique presented in [15] infers “worst” lower bounds (a lower bound on the derivation height) which are not comparable to our “best” lower bounds.

We perform one experimental evaluation for upper bounds and one for lower bounds. The results of these experiments are summarized in Fig. 6. In all evaluations, the tools are run with a timeout of 60 secs. per example. In the first evaluation we analyze a total of 121 challenging programs written in C mainly extracted from [6,10]. We compare our approach with Loopus [25], the previous version of CoFloCo (called “Old” in the table) [14], KoAt [9], and C4B [10]. We use the tool llvm2kittel [13] to transform the llvm-IR programs into integer rewrite systems (for KoAT) which are translated to cost relations by a dedicated script. These CRs are used by our tool, and “Old”. On Fig. 6, we can see for each tool how many examples are reported in each complexity category and the average time in seconds needed per program for each of the tools. The times of CoFloCo and Old include the refinement process of [14]. On the right-bottom, we report for how many programs CoFloCo computes a better or worse asymptotic bound that the other tools. For instance, CoFloCo computes a better bound than KoAt in 28 examples and Loopus computes a better bound than CoFloCo in 3 examples. It can be seen that CoFloCo computes better bounds for a higher number of examples.

<table>
<thead>
<tr>
<th>UB</th>
<th>n</th>
<th>n^2</th>
<th>n^3</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoFloCo</td>
<td>3</td>
<td>62</td>
<td>33</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Old</td>
<td>3</td>
<td>55</td>
<td>32</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Loopus</td>
<td>2</td>
<td>56</td>
<td>27</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>KoAT</td>
<td>3</td>
<td>45</td>
<td>40</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>C4B</td>
<td>1</td>
<td>42</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LB</th>
<th>n</th>
<th>n^2</th>
<th>n^3</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoFloCo</td>
<td>4</td>
<td>85</td>
<td>23</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>PUBS</td>
<td>95</td>
<td>38</td>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>CoFloCo</td>
<td>KoAT</td>
<td>Loopus</td>
<td>Old</td>
<td>LB</td>
<td>Pubs(LB)</td>
</tr>
<tr>
<td>worse</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table: Experimental results: The number of examples with a given complexity order or (F)ailed for upper (UB) and lower (LB) bounds. (T) is the average time per example in secs. On the right bottom, a comparison between CoFloCo and the other tools.
than any other tool. The second evaluation compares CoFloCo and PUBS [5] for computing lower bounds. We analyze a total of 160 examples. The 121 examples from the first evaluation plus the examples of PUBS’s evaluation. CoFloCo obtains a better result (a higher complexity order) in 60 examples. In contrast PUBS obtains better bounds in 2 examples. A detailed experimental report is online: http://cofloco.se.informatik.tu-darmstadt.de/experiments.

Acknowledgements Research partly funded by the EU project FP7-610582 ENVISAGE: Engineering Virtualized Services. I thank the anonymous reviewers, R.Hähnle, F.Zuleger, M.Sinn and S.Genaim for their careful reading.

References


A Additional Strategies

In this section we provide additional strategies for transforming final constraints during the computation of cost structures of phases.

A.1 Inductive Sum Strategy (with resets)

This variant of the Inductive Sum strategy allows us to obtain upper bounds of sums in terms of the initial variables of the phase only \((\overline{\pi}_f)\). Such bounds are not only valid for \(\sum_{i=1}^{n} |l(x_ix_i')|\) but also for any partial sum \(\sum_{i=1}^{p} |l(x_ix_i')|\) with \(p \leq \#c_i\) and they are valid for non-terminating phases where \(\overline{\pi}_f\) is not defined.

In order to compute such a bound, the strategy proceeds as before: Let \(\sum_{k=1}^{m} iv_k \leq |l(x_f x_f')| \in Psums^{c_i}\), it generates a candidate \(cd(\overline{\pi})\) using the constraint set of \(c_i \varphi_i(\overline{\pi} x y)\) and Farkas’ Lemma. However, this time the strategy uses the condition \(|l'(|x_f x_f'|)| \leq cd(\overline{\pi}) - cd(\overline{\pi}_f) \land |l'(|x_f x_f'|)| \leq cd(\overline{\pi})\). This condition contains the extra conjunction \(|l'(|x_f x_f'|)| \leq cd(\overline{\pi}_f)\) which allows us to ignore the final value of the candidate \(cd(\overline{\pi}_f)\) and guarantee that the generated constraint is valid for any partial sum. This strategy considers the class \(Cntr\), instead of \(Cnt\), which incorporates the extra conjunction as well (See Fig. 7). It also considers the \(Rst\) class to support phases where the candidate is reset.

**Lemma 3.** If we classify every \(c_e \in ph\) into \(Cntr\), \(Ic\), \(Dc\) and \(Rst\) with respect to a candidate \(cd(\overline{\pi})\), the following constraints are valid:

\[
\sum_{c_e \in Cntr} cntr_c \leq iv_{cd} + \sum_{c_e \in Ic} smiv_{ic_c} + \sum_{c_e \in Rst} smiv_{rst_c}, \quad iv_{cd} \leq |cd(\overline{\pi}_f)|
\]

For each \(c_e \in Ic\), the strategy adds the pending constraints \(iv_{ic_e} \leq |ic(\overline{\pi}_f)|\) to \(Psums^{c_e}\) and for each \(c_e \in Rst\), it adds \(iv_{rst_e} \leq |rst(\overline{\pi})|\) to \(Psums^{c_e}\). Note that this strategy adds the sum of all the resets instead of considering the maximum. In addition to that, it ignores the decrements of \(c_e \in Dc\). This is necessary to guarantee that the constraints are also valid for partial evaluations of the phase.

<table>
<thead>
<tr>
<th>(Cntr)</th>
<th>Condition when (\bowtie) is (\leq)</th>
<th>Defines</th>
<th>(cntr_c = \sum_{k=1}^{p} smiv_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>\sum_{k=1}^{m} iv_k \leq</td>
<td>l'(</td>
<td>x_f x_f'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(CntTri)</th>
<th>Condition when (\bowtie) is (\leq)</th>
<th>Condition when (\bowtie) is (\geq)</th>
<th>Defines</th>
<th>(cntTri_c = \sum_{k=1}^{p} smiv_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>\sum_{k=1}^{m} iv_k \bowtie</td>
<td>l'(</td>
<td>x_f x_f'</td>
<td>)</td>
</tr>
</tbody>
</table>

| \(NoCnt\) | \(cd(\overline{\pi}_f) - cd(\overline{\pi}) = 0\) |

**Fig. 7.** Additional Classes of CE \(c_e\) with respect to a candidate \(cd(\overline{\pi})\), their condition and defined term.
A.2 Triangular Sum Strategy

This strategy represents an alternative to the Basic Product strategy for dealing with constraints \( \sum_{k=1}^{m} w_k \propto |l(xx')| \in Psums^c \) where \(|l(xx')|\) varies in each iteration by a constant amount.

Example 11. A typical example is Program 2 in Fig. 8. In this example, cost equations 1 and 2 represent the outer loop and 3 and 4 the inner loop. The chain that represents the total cost of the program is \([1+2]\). We did not include the final values of the variables \(x_o, y_o\) and \(n_o\) in the CRs to simplify the presentation. Let us consider obtaining the lower bound of such example.

We consider that the cost of the inner loop (chain \([3+4]\)) is \((iv_1, \emptyset, \{iv_1 = |n-y|\})\) which yields a cost of \((iv_1, \emptyset, \{iv_1 = |n-x|\})\) for CE 1. The main cost expression of the phase 1+ is \(E_1+ := smiv_1\), no constraints are generated from the non-final constraints and the pending sets are: \(Psums^t = \{iv_1 \leq |n-x|, iv_1 \geq |n-x|\}\) and \(Pms^t = \emptyset\). If we apply the basic product strategy to \(iv_1 \geq |n-x|\), we would obtain \(smiv_1 \geq smiv_{it1} \ast [iv]_2\) and later \(smiv_{it1} \geq |n_s - x_s|\) and \([iv]_2 \geq 1\) (the minimum value of \(|n-x|\) is 1 in the last iteration) which represents the imprecise lower bound \(|n-x|*1\).

Instead, we consider that \(|n-x|\) decreases by at most 1 in each iteration so we can reformulate:

\[
\sum_{j=1}^{\#c_1} |n_j - x_j| \geq \sum_{j=1}^{\#c_1} (|n_s - x_s| - (j - 1)) = |n_s - x_s| \ast \#c_1 - \sum_{j=0}^{\#c_1-1} j = |n_s - x_s| \ast \#c_1 - (\#c_1 \ast \#c_1 - \#c_1) / 2
\]

This expression can be represented with constraints as follows:

\[
smiv_1 \geq iv_{p1} - \frac{1}{2} iv_{p2} + \frac{1}{2} smiv_{it1}, \quad iv_{p1} \geq iv_{ini} \ast smiv_{it1}, \quad iv_{p2} \leq smiv_{it1} \ast smiv_{it1}, \quad iv_{ini} \geq |n_s - x_s|
\]

Note that the constraint over \(iv_{p2}\) has \(\leq\) instead of \(\geq\). This is because \(iv_{p2}\) appears as a negative summand in the first constraint and it has to be maximized. Later, applying the Inductive Sum strategy to \(iv_{it1} \leq 1\) and \(iv_{it1} \geq 1\) (in \(Psums^t\)), we generate \(smiv_{it1} = \left( |n_s - x_s| - (n_f - x_f) \right)\). Once we compute the cost of the complete chain \([1+2]\), we will transform \(\left( |n_s - x_s| - (n_f - x_f) \right)\) into \(|n_s - x_s|\) (Because \(n_f - x_f\) must be 0 in chain \([1+2]\)). If we minimize the cost...

---

Program 2

```plaintext
1 for (int x=0;x<n; x++)
2   for (int y=x; y<n; y++)
3     //tick(1);
```

Refined cost relations

<table>
<thead>
<tr>
<th></th>
<th>Refined cost relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(for_1(x, n) = for_2[3^4](y, n) + : for_1(x', n)) { (y = x, x &lt; n, x' = x + 1) }</td>
</tr>
<tr>
<td>2</td>
<td>(for_1(x, n) = 0 { x \geq n } )</td>
</tr>
<tr>
<td>3</td>
<td>(for_2(y, n) = 1 + for_3(y', n) { y &lt; n, y' = y + 1 } )</td>
</tr>
<tr>
<td>4</td>
<td>(for_4(y, n) = 0 { y \geq n } )</td>
</tr>
</tbody>
</table>

Lower bound = \(|n|^2/2 + |n|/2 \)

---

Fig. 8. Program 2 and its refined cost relations.
of the resulting cost structure, we obtain:
\[
|n_x - x_i|^2 - |n_x - x_i|^2/2 + |n_x - x_i|/2 = (1)\ |n_x|^2/2 + |n_x|/2 \quad \text{(because } x_s = 0)\]

In general, given a constraint \( \sum_{k=1}^m s_{mv_k} \triangleright |l(xlx)| \in Psums^{c_{iv}} \), the strategy generates a candidate \( cd(x) \) that approximates the cost of one instance of \( |l(xlx)| \), it is positive and it varies by a constant amount \( q_i \in Q \). That is:
\[
\varphi_i(xlx'y) \Rightarrow (l(xlx') \triangleright cd(x) \geq 0 \land cd(x) - cd(x') \triangleright q_i)
\]

This strategy considers the classes CntTri and NoCnt (See Fig. 7). If \( c_{iv} \in CntTri \), there exists a constraint \( (\sum_{k=1}^m iv_{k} \triangleright |l'(xlx')|) \in Psums^{c_{iv}} \) that is bound by the candidate and the candidate varies by an amount \( q_{iv} \). As in previous strategies, this condition coincides with the one used to generate the candidate and \( c_{iv} \in CntTri \). The CEs \( c_{iv} \in NoCnt \) do not modify the candidate.

**Lemma 4.** If we classify every \( c_{iv} \in ph \) into CntTri and NoCnt with respect to a candidate \( cd(x) \). Let \( q \) be \( q := \max_{c_{iv} \in CntTri} (q_{iv}) \) when \( \triangleright \) is \( \leq \) or \( q := \min_{c_{iv} \in CntTri} (q_{iv}) \) when \( \triangleright \) is \( \geq \). The following constraints are valid:
\[
\sum_{c_{iv} \in CntTri} cntTri_{c_{iv}} \triangleright iv_{p1} + \frac{4}{3} iv_{p2} - \frac{4}{3} iv_{its} , \quad iv_{p1} \triangleright iv_{init} * iv_{its}
\]
\[
iv_{p2} = iv_{its} * iv_{its} , \quad iv_{its} = \sum_{c_{iv} \in CntTri} smiv_{c_{iv}} , \quad iv_{init} \triangleright |cd(x)|
\]

The constraints of the form \( iv = x \) stand for \( iv \leq x \) and \( iv \geq x \). The intermediate variable \( iv_{its} \) represents the sum of all the iterations \( #c_{iv} \) of \( c_{iv} \in CntTri \). The constraints \( iv_{its} \leq 1 \) and \( iv_{its} \geq 1 \) are added to each \( Psums^{c_{iv}} \) such that \( c_{iv} \in CntTri \) (if they are not in the set yet).

**B Complete Example of Phase Cost Structure**

In this section we present a complete example of the computation of a cost structure for a phase to illustrate how the different strategies work together.

Fig. 9 contains program 3 and its refined cost relations (Note that we reused the cost relations of program 1 for the inner loop). This example has 5 cost equations. CE 10 and 11 represent the loop paths that reach the inner loop. In CE 10 the body of the inner loop is executed at least once and in CE 11 the body of inner loop is not executed. CE 12 corresponds to the loop path that visits line 5 in which \( y \) is incremented. CE 13 corresponds the loop path that visits line 6. There \( y \) is reset to \( y \). Finally, CE 14 is the exit path of the loop.

We will compute the cost structure of the phase \((10 \lor 11 \lor 12 \lor 13)^+\) based on the cost structures of CEs 10 – 13. We assume we have the cost structure \( \langle 2iv_{5}, 0, iv_{5} \leq |y - y'| \rangle \) for CE 10 and the cost structures of CEs 11 – 13 are empty. For simplicity, we only consider constraints for upper bounds (with \( \leq \)).
The main cost expression of the phase is $2 smiv_5$. Fig. 10 contains all the steps of constraint generation from final constraints of the CEs. Each step has four parts:

1. **State**: The state of each of the pending sets $Psums$ and $Pms$.
2. **SelConstr**: The constraint selected from one of the pending sets.
3. **Strategy**: The strategy applied to the selected constraint: ISR (Inductive Sum with Resets), BP (Basic Product) or MM (Max-Min). Additionally we express the classification of the $ce \in ph$ and the related defined terms $ctnre$, $ic_c$, etc.
4. **NewCs**: The constraints generated by the applied strategy. The constraints added to the pending sets are not included here but they can be seen in the state of the next step.

We apply 4 steps until all the intermediate variables are bound. The resulting cost structure $\langle E, IC, FC(x) \rangle$ contains all the generated constraints (NewCs):

$$
E = 2 smiv_5 \\
IC = \{ smiv_5 \leq iv_1 + smiv_{ic_{12}} + smiv_{rst_{13}}, \ smiv_{rst_{13}} \leq smiv_{it_{13}} \} \\
FC = \{ iv_1 \leq |y|, \ |iv|_2 \leq |z|, \ smiv_{ic_{12}} + smiv_{it_{13}} \leq |x| \}
$$

This cost structure represents the upper bound $2(|y| + max(|x|, |x| * |z|))$.

**C Solving Cost Structures**

In this section, we detail how upper and lower bound expressions can be obtained from a given cost structure. In order to compute upper bounds of a cost structure $\langle E, IC, FC(\pi) \rangle$, we maximize the positive summands and minimize the negative summands of the main expression $E$. Conversely, we minimize positive summands and maximize negative ones to obtain lower bounds. This is done by assigning symbolic expressions over $\pi$ to the intermediate variables according to the constraints in $IC$ and $FC(\pi)$. In general this is immediate for constraints that contain a single $iv$ on their left side. If we have $iv \leq x$ we simply assign $x$ to $iv$. 

24
If we have constraints with several $iv$ on the left side, it is less straightforward how to obtain a maximizing/minimizing assignment. Consider a constraint $iv_1 + iv_2 \leq x$. For upper bound constraints, a simple but imprecise alternative is to assign $\alpha(iv_1) = x$ and $\alpha(iv_2) = x$. This is equivalent to splitting the constraint into two weaker constraints $iv_1 \leq x$ and $iv_2 \leq x$. Unfortunately, this is not possible for lower bound constraints like $iv_1 + iv_2 \geq x$. Another possibility is to consider the extreme cases, when $\alpha(iv_1) = x$ and $\alpha(iv_2) = 0$ and vice-versa $\alpha(iv_1) = 0$ and $\alpha(iv_2) = x$. If we have a cost of the form $c_1 * iv_1 + c_2 * iv_2$ then $\max(c_1 * x + c_2 * 0, c_1 * 0 + c_2 * x)$ is a valid upper bound and $\min(c_1 * x + c_2 * 0, c_1 * 0 + c_2 * x)$ is a valid lower bound. This approach allows us to obtain upper and lower bounds that are more precise but it is limited to intermediate variables that only appear linearly. That is, if we have an expression like $iv_1 * iv_2$, $\max(iv_1, iv_2)$ or $\min(iv_1, iv_2)$, the extreme cases do not represent the maximum or minimum cost. For example, the maximum cost of $iv_1 * iv_2$ would correspond to assigning $\alpha(iv_1) = x/2$ and $\alpha(iv_2) = x/2$. In those cases we can resort to assigning the maximal cost to both variables for upper bounds ($\alpha(iv_1) = x$ and $\alpha(iv_2) = x$) and assign both variables to zero for lower bounds ($\alpha(iv_1) = 0$ and $\alpha(iv_2) = 0$). Fortunately, in most cases where we have constraints with multiple variables on the left side, these variables appear only linearly in the rest of the cost structure.

**Example 12.** Consider the cost structure of chain $[1.2]$ of program 1: $\langle 1iv_2 + 2iv_6, \{iv_2 = iv_3 * iv_4\}, \{iv_3 + iv_6 = |y + x|, iv_4 = |z|\} \rangle$. Let us obtain an upper
bound of chain $[1,2]$. First, we solve the constraints with a single variable on
the left side and we obtain: $1(iv_3 \cdot |z|) + 2iv_6$ such that $iv_3 + iv_6 = |y + x|$. Then, we check that $iv_6$ and $iv_3$ do not appear multiplied by each other (or inside the same $max$ or $min$ expression) and we consider the extreme cases: (1) $iv_6 = |y + x|$ and $iv_3 = 0$; and (2) $iv_6 = 0$ and $iv_3 = |y + x|$. For obtaining an upper bound, we take the maximum of both cases and simplify:

$$\max(|y + x| \cdot |z|) + 2 = 0, 1(0 \cdot |z|) + 2|y + x| = \max(|y + x| \cdot |z|, 2|y + x|)$$

$$= \max(|z| 2) \cdot |y + x|$$

For obtaining a lower bound, we consider the minimum of both cases:

$$\min(|y + x| \cdot |z|) + 2 = 0, 1(0 \cdot |z|) + 2|y + x| = \min(|z|, 2) \cdot |y + x|$$

Example 13. Consider now the cost structure of phase $(10 \lor 11 \lor 12 \lor 13)^+$ of program 3:

$$E = 2 \cdot smiv_5$$
$$IC = \{ smiv_5 \leq iv_1 + smiv_{inc_{12}} + smiv_{rst_{13}}, smiv_{rst_{13}} \leq smiv_{it_{13}} \cdot \lceil iv \rceil_2 \}$$
$$FC = \{ iv_1 \leq |y|, \ lceil iv \rceil_2 \leq |z|, \ smiv_{inc_{12}} + smiv_{it_{13}} \leq |x| \}$$

As in the previous example, we solve all the constraints with only one variable on
the left side first and obtain: $2 * (|y| + smiv_{inc_{12}} + (smiv_{it_{13}} \cdot |z|))$ such that $smiv_{inc_{12}} + smiv_{it_{13}} \leq |x|$. Also in this case, $smiv_{inc_{12}}$ and $smiv_{it_{13}}$ do not appear multiplying each other (or inside the same $max$ or $min$ expression). Therefore, we consider the extreme cases and take their maximum and simplify:

$$\max(2 \cdot (|y| + |x|), 2 \cdot (|y| + (|x| \cdot |z|))) = 2(|y| + \max(|x|, |x| \cdot |z|))$$

If we split $smiv_{inc_{12}} + smiv_{it_{13}} \leq |x|$ into $smiv_{inc_{12}} \leq |x|$ and $smiv_{it_{13}} \leq |x|$ the resulting upper bound is less precise but still valid: $2 * (|y| + |x| + (|x| \cdot |z|))$.

Note that given a cost structure, there might be multiple bound expressions
that can be extracted depending on which constraints are considered. Moreover,
the different possible bound expressions are often not comparable among each
other. For instance, given a cost structure $(iv, 0, \{ iv \leq |x|, iv \leq |y| \})$, both $|x|$ and $|y|$ are valid upper bounds. These upper bounds are not comparable (we
do not know whether $x$ is bigger than $y$ or not) and actually the best upper
bound is $\min(|x|, |y|)$. In our implementation, we prioritize efficiency and do not
try to obtain the best bound. Instead we select the constraints that we con-
sider heuristically trying to obtain a simple bound with the best the asymptotic
complexity.

D Additional Experiments

In the recent work of [25], an extensive experimental evaluation is presented. In
that evaluation 1659 functions from a compiler optimization benchmark (cBench)
are evaluated. We replicated this evaluation (with 1625 examples)\(^7\) with our tool, the previous version of CoFloCo and KoAt. The results are provided in Fig. 11. However, we realized that our tool fails to compute a bound in many examples because the translation using llvm2kittel does not consider structs, arrays and simple pointer references that are better handled by Loopus. In order to get a measure of this, we transformed the examples generated by llvm2kittel back into C programs\(^8\) and run Loopus on the resulting programs. This corresponds to the row Loopus\(^*\) in Fig. 11. The results indicate that the translation plays a major role on the results. Factoring out the translation, all tools report similar results in terms of number of examples analyzed successfully (Loopus is still much faster).

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
& UB & n & \(n^2\) & \(\geq n^3\) & Failed & # Timeouts & Avg. Time \\
\hline
CoFloCo & 216 & 152 & 39 & 0 & 1113 & 105 & 3.76 \\
Old & 213 & 149 & 38 & 0 & 1109 & 116 & 4.89 \\
Loopus & 204 & 487 & 97 & 14 & 806 & 17 & 0.37 \\
KoAT & 204 & 138 & 38 & 1 & 1104 & 140 & 5.58 \\
Loopus\(^*\) & 197 & 142 & 40 & 0 & 1226 & 20 & 0.42 \\
\hline
CoFloCo & KoAT & Loopus & Old & Loopus\(^*\) & \\
\hline
better & 35 & 24 & 8 & 40 & \\
worse & 5 & 416 & 2 & 12 & \\
\hline
\end{tabular}
\caption{Replication of experimental evaluation of [25]. Complexity, Failed, Timeouts and average Time in seconds}
\end{table}

E Soundness Proofs

**Theorem 1.** Let \(T(\pi)\) be a chain, a phase or a CE. Then the cost structure \(\langle E_T, IC_T, FC_T(\pi) \rangle\) obtained following the algorithms of Secs. 4 and 5 is valid.

For the cost structures generated in Sec. 4 (For CEs and chains), the main cost expression \(E_T\) was derived following the semantic rules in Fig. 3 and the validity of the constraints \(IC_T\) and \(FC_T(\pi)\) follows directly from the validity of the cost structures of its components and, in the case of chains, also from the validity of the summaries \(\varphi_{ph}\) used in the quantifier elimination.

For the cost structures generated in Sec. 5 (For phases), The validity of the main cost expression \(E_T\) is immediate given the definition of smiv. The constraints generated in Sec. 5.1 are generated directly from the non-final constraints of the cost structures of the CEs in the phase and their validity follows directly from the definitions of smiv, \([iv]\) and \([iv]\). In what follows, we prove that the constraints generated by the different strategies in Sec. 5.2 and App. A are also valid.

\(^7\)We excluded some examples where the translation tools failed

\(^8\)Using the script available at [https://github.com/s-falke/kittel-koat](https://github.com/s-falke/kittel-koat)
E.1 Soundness of Constraints Generated from Final Constraints

An evaluation of a phase \( ph = (c_1 \lor \cdots \lor c_n)^+ \) consist on a sequence of evaluations of CEs \( \langle \alpha, c_i(\overline{x_j}x_{j+1}) \rangle \downarrow k_j \) for \( 1 \leq j < f \) and \( c_i \in ph \).

We define the auxiliary function \( CE? \) that given an index \( j \) tell us which CE of the phase has been evaluated in the \( j-th \) position. That is \( CE? (j) = c_i \) such that the \( j-th \) CE evaluation within the phase evaluation is: \( \langle \alpha, c_i(\overline{x_j}x_{j+1}) \rangle \downarrow k_j \).

Additionally, we define the following auxiliary notion:

**Definition 8 (Partial sum).** Let \( iv \) be an intermediate variable defined in CE \( c_i \), we define the partial sum \( psmiv^{c_i}[a..b] \) as the sum of all the instances of \( iv \) in the CE evaluations such that \( CE? (j) = c_i \) in the segment \( a < j < b \) of the phase evaluation:

\[
psmiv^{c_i}[a..b] = \sum_{a \leq j < b \wedge CE? (j) = c_i} iv_j
\]

It follows from the definition that \( psmiv^{c_i}[a..b] \leq smiv \) for every \( 1 \leq a \leq b < f \) and in particular \( psmiv^{c_i}[1..f] = smiv \). Also \( psmiv^{c_i}[a..b+1] = psmiv^{c_i}[a..b] + iv_b \) if \( CE? (b) = c_i \) (where \( iv_b \) is an instance of \( iv \)) and \( psmiv^{c_i}[a..b+1] = psmiv^{c_i}[a..b] \) otherwise.

**Soundness of Inductive Sum Strategy (Lemma 1)** We have to prove that, given a candidate \( cd(\overline{x}) \) such that we could classify every \( c_e \in ph \) into the classes \( Cnt, Dc \) and \( Ic \) according to their definitions in Sec. 5.2. The following constraints are valid.

\[
\sum_{c_e \in Cnt} cnt_c \otimes iv_{cd} + iv_{cd-} = \sum_{c_e \in Ic} smiv_{ic} - \sum_{c_e \in Dc} smiv_{dc} + psmiv^{c_e}[a..b],
\]

These constraints involve additional intermediate variables \( iv_{ic} \) and \( iv_{dc} \) for each \( c_e \in Ic \) and \( c_e \in Dc \) whose value is defined as \( iv_{ic} := |ic(\overline{x^f})| \) and \( iv_{dc} := |dc(\overline{x^f})| \). The addition of \( iv_{ic} \otimes |ic(\overline{x^f})| \) and \( iv_{dc} \otimes |dc(\overline{x^f})| \) to \( Psums^{c_e} \) follows directly from these definitions.

In this setting, we are not restricted by the format of the cost structures and we can merge the three constraints into one:

\[
\sum_{c_e \in Cnt} cnt_c \otimes cd(\overline{x}) - cd(\overline{x^f}) + \sum_{c_e \in Ic} smiv_{ic} - \sum_{c_e \in Dc} smiv_{dc} = \sum_{c_e \in Cnt} psmiv^{c_e}[1..n]
\]

Then, we can define a generalized constraint that holds for a segment of the evaluation \( [1..n] \). The constraint above corresponds to the case where consider the complete phase evaluation \( (n = f) \). The generalized constraint has the following form:

\[
\sum_{c_e \in Cnt} ( \sum_{smiv_{ic} \in cnt_c} psmiv^{c_e}_{ic}[1..n] ) \otimes cd(\overline{x}) - cd(\overline{x_n}) + \sum_{c_e \in Ic} ( psmiv^{c_e}_{ic}[1..n] ) - \sum_{c_e \in Dc} ( psmiv^{c_e}_{dc}[1..n] )
\]

We prove that the generalized constraint holds for all \( n \) by induction.
Base Case For \( n = 1 \) we have \( 0 \gg 0 \) because the intervals for all \( ps \) \( \text{are} \) empty and \( cd(x_1) - cd(x_1) = 0 \).

Inductive Case For the inductive case, we assume the expression holds for \( n \) and prove it for \( n + 1 \). We distinguish cases depending on which CE is evaluated in the \( n \)-th place. In particular, whether \( CE(n) \) belongs to \( Cnt, Dc \) or \( Ic \). In each case, we reduce the \( n + 1 \) case to the \( n \) case and prove that the additional summands maintain the inequality.

- If \( CE(n) = c_i \in Cnt \), the left hand side of the constraint is:

\[
\sum_{c_e \in Cnt} \left( \sum_{smiv_k \in cnt} ps \text{[1..n + 1]} \right) = \sum_{c_e \in Cnt} \sum_{smiv_k \in cnt} ps \text{[1..n]} + iv_k \]

Where \( iv_k \) are the instances of the variables \( iv \) in the \( n \)-th CE evaluation of the phase. The right hand side of the constraint is:

\[
\sum_{smiv_k \in cnt} iv_k \gg cd(x_1) - cd(x_{n+1}) \]

If we apply the induction hypothesis, we are left to prove:

\[
\sum_{smiv_k \in cnt} iv_k \gg cd(x_1) - cd(x_{n+1}) \]

This constraint is directly guaranteed by the classification condition of \( Cnt \):

\[
\sum_{smiv_k \in cnt} iv_k \gg [l'(x'f)] \in Ps \wedge |l'(x'f')| \gg cd(x) - cd(x') \]

which is valid for any evaluation of a \( c \in Cnt \) and in particular for the evaluation of \( c \) in the \( n \)-th place.

- If \( CE(n) = c_i \in Dc \), the left side of the constraint does not change with respect to the case with \( n \):

\[
\sum_{c_e \in Cnt} \left( \sum_{smiv_k \in cnt} ps \text{[1..n + 1]} \right) = \sum_{c_e \in Cnt} \sum_{smiv_k \in cnt} ps \text{[1..n]} \]

And the right hand side is:

\[
\sum_{c_e \in Dc} \left( ps \text{[1..n + 1]} \right) = \sum_{c_e \in Dc} \sum_{smiv_k \in cnt} ps \text{[1..n]} - iv_{dc} \]

And the right hand side is:

\[
\sum_{c_e \in Cnt} \left( \sum_{smiv_k \in cnt} ps \text{[1..n + 1]} \right) = \sum_{c_e \in Cnt} \sum_{smiv_k \in cnt} ps \text{[1..n]} + iv_{dc} \text{n} \]

\[
\sum_{c_e \in Dc} \left( ps \text{[1..n + 1]} \right) = \sum_{c_e \in Dc} \sum_{smiv_k \in cnt} ps \text{[1..n]} - iv_{dc} \text{n} \]

29
Where \( iv_{dc,n} \) is the instance of \( iv_{dc_i} \) in the \( n \)-th CE evaluation. We have to prove:

\[
0 \triangleright cd(\overline{f_n}) - cd(\overline{f_{n+1}}) - iv_{dc,n}
\]

By definition of \( Dc \) we have \( cd(\overline{f_n}) - cd(\overline{f_{n+1}}) \) is positive and \( iv_{dc,n} = |dc_c(\overline{f_n}, \overline{f_{n+1}})| \triangleright cd(\overline{f_n}) - cd(\overline{f_{n+1}}) \) which guarantees the condition that we want to prove.

- If \( CE?(n) = c_i \in Ic \), the left side of the constraint does not change with respect to the case of \( n \) as in the previous case. The right hand side of the constraint can be decomposed as follows:

\[
cd(\overline{f_1}) - cd(\overline{f_{n+1}}) + \sum_{c_e \in Ic} (psemiv_c^{ic_e}[1..n + 1])
- \sum_{c_e \in Dc} (psemiv_c^{dc_e}[1..n + 1])
- \sum_{c_e \in Dc} (psemiv_c^{dc_e}[1..n])
= cd(\overline{f_n}) + cd(\overline{f_{n+1}}) - cd(\overline{f_{n+1}}) + \sum_{c_e \in Ic} (psemiv_c^{ic_e}[1..n]) + iv_{ic,n}
\]

Therefore, we have to prove:

\[
0 \triangleright cd(\overline{f_n}) - cd(\overline{f_{n+1}}) + iv_{ic,n}
\]

This is directly guaranteed by the definition of \( Ic \) (given that \(|ic_c(\overline{f_n}, \overline{f_{n+1}})| = iv_{ic,n}\).

**Soundness of Inductive Strategy with Resets (Lemma 3)** We have to prove that, given a candidate \( cd(\overline{x}) \) such that we could classify every \( c_e \in ph \) into the classes \( Cntr, Dc, Ic \) and \( Rst \) according to their definitions in Secs. 5.2 and A.1. The following constraints are valid:

\[
\sum_{c_e \in Cntr} cntre_c \leq iv_{cd} + \sum_{c_e \in Ic} smiv_{ic_e} + \sum_{c_e \in Rst} smiv_{rst_e}, \ iv_{cd} \leq |cd(\overline{x})|
\]

These constraints involve additional intermediate variables \( iv_{ic_e} \) and \( iv_{rst_e} \) for each \( c_e \in Ic \) and \( c_e \in Rst \) whose value is defined as \( iv_{ic_e} := |ic_c(\overline{x})| \) and \( iv_{rst_e} := |rst_e(\overline{x})| \). The addition of \( iv_{ic_e} \leq |ic(\overline{x})| \) and \( iv_{rst_e} \leq |rst(\overline{x})| \) to \( Psms^{c_e} \) follows directly from these definitions.

Similarly to the previous proof, we merge the constraints in a single one:

\[
\sum_{c_e \in Cntr} cntre_c \leq |cd(\overline{x})| + \sum_{c_e \in Ic} smiv_{ic_e} + \sum_{c_e \in Rst} smiv_{rst_e}
\]

Then, we can define a generalized constraint that holds for a segment of the evaluation \([1..n]\). The constraint above corresponds to the case where consider the complete phase evaluation \((n = f)\). The generalized constraint has the following form:

\[
\sum_{c_e \in Cntr} smiv_{ic_e}[1..n] \leq |cd(\overline{x})| - |cd(\overline{x_f})| + \sum_{c_e \in Ic} (psemiv_c^{ic_e}[1..n]) + \sum_{c_e \in Rst} (psemiv_c^{rst_e}[1..n])
\]
Note that \(|cd(\pi_1)| - |cd(\pi_n)| \leq |cd(\pi_1)|\) for all \(n\) so our constraint is a safe over-approximation of the generalized constraint. We prove that the generalized constraint holds for all \(n\) by induction.

**Base Case** For \(n = 1\) (the empty sequence) we have \(0 \leq |cd(\pi_1)| - |cd(\pi_1)| = 0\) which is trivially true.

**Inductive Case** For the inductive case, we assume the expression holds for \(n\) and prove it for \(n+1\). We distinguish cases depending on which CE is evaluated in the \((n)\)-th place. In particular, whether \(CE(n)\) belongs to \(Cntr, Dc, Ic\) or \(Rst\). In each case, we reduce the \(CE\) to the \((n)\)-th place. In particular, whether \(CE(n)\) is positive, \(CD(n)\) or \(-CD(n)\). We distinguish cases depending on which CE is evaluated in the \((n)\)-th place. The right hand side of the constraint is:

\[
(\sum_{c_e \in Ic} \sum_{s_miv_k \in Cntr_e} psimiv_e \sum_k [1..n+1]) = (\sum_{c_e \in Ic} \sum_{s_miv_k \in Cntr_e} psimiv_e [1..n]) + \sum_{s_miv_k \in Cntr_e} iv_{kn}
\]

Where \(iv_{kn}\) are the instances of the variables \(iv_k\) in the \(n\)-th CE evaluation of the phase. The right hand side of the constraint is:

\[
|cd(\pi_1)| - |cd(\pi_{n+1})| + \sum_{c_e \in Ic} (psimiv_e \sum_k [1..n+1]) + \sum_{c_e \in Rst} (psimiv_e \sum_k [1..n])
\]

If we apply the induction hypothesis, we are left to prove:

\[
\sum_{s_miv_k \in Cntr_e} iv_{kn} \propto |cd(\pi_n)| - |cd(\pi_{n+1})|
\]

According to the definition of \(Cntr\), we have:

1. \(\sum_{s_miv_k \in Cntr_e} iv_{kn} \leq |l'(\pi_n, \pi_{n+1})| - |cd(\pi_n)|\)
2. \(\sum_{s_miv_k \in Cntr_e} iv_{kn} \leq |l'(\pi_n, \pi_{n+1})| - |cd(\pi_n)|\)

The property (2) implies that \(cd(\pi_n)\) is positive \((cd(\pi_n) = |cd(\pi_n)|)\).

- If \(cd(\pi_{n+1})\) is positive, \(cd(\pi_{n+1}) = |cd(\pi_{n+1})|\) and we apply (1):

  \[
  \sum_{s_miv_k \in Cntr_e} iv_{kn} \leq |cd(\pi_n)| - |cd(\pi_{n+1})| = |cd(\pi_n)| - |cd(\pi_{n+1})|
  \]

- If \(cd(\pi_{n+1})\) is negative, \(|cd(\pi_{n+1})| = 0\) and we apply (2):

  \[
  \sum_{s_miv_k \in Cntr_e} iv_{kn} \leq |l'(\pi_n, \pi_{n+1})| \leq |cd(\pi_n)| \leq |cd(\pi_n)| = |cd(\pi_n)| - |cd(\pi_{n+1})|
  \]
– If $CE? (n) = c_i \in Dc$, the left side of the constraint does not change with respect to the case with $n$ and the right side is:

$$|cd(x_1)| - |cd(x_{n+1})| + \sum_{c_e \in Ic} (psmiv_{ce_{ic_e}}[1..n + 1])$$

$$+ \sum_{c_e \in Rst} (psmiv_{ce_{rst_e}}[1..n + 1]) =$$

$$|cd(x_1)| - |cd(x_n)| + |cd(x_{n+1})| - |cd(x_{n+1})| + \sum_{c_e \in Ic} (psmiv_{ce_{ic_e}}[1..n])$$

$$+ \sum_{c_e \in Rst} (psmiv_{ce_{rst_e}}[1..n])$$

Therefore, we have to prove:

$$0 \leq |cd(x_n)| - |cd(x_{n+1})|$$

Because $CE? (n) = c_i \in Dc$, we have:

$$1 \quad 0 \leq dc_i(x_{n+1}) \leq cd(x_n) - cd(x_{n+1})$$

- if $cd(x_{n+1})$ is positive, $cd(x_n)$ is also positive and $0 \leq cd(x_n) - cd(x_{n+1}) = (1)$

$$|cd(x_n)| - |cd(x_{n+1})| =$$

- if $cd(x_{n+1})$ is negative, $|cd(x_n)| = |cd(x_{n+1})| \geq 0$ (by definition of $|x| = max(x, 0)$).

– If $CE? (n) = c_i \in Ic$, the left side of the constraint does not change with respect to the case with $n$. We can decompose the right side as before and as a result we have to prove:

$$0 \leq |cd(x_n)| - |cd(x_{n+1})| + iv_{ic_n}$$

From the definition of $Ic$, we know:

$$1 \quad 0 \leq cd(x_n) - cd(x_{n+1}) + |ic_e(x_n x_{n+1})|$$

We distinguish cases:

- If $cd(x_{n+1})$ is negative, $|cd(x_{n+1})| = 0$ and we have $0 \leq |cd(x_n)| + iv_{ic_n}$ which is trivially true (both summands are positive).

- If $cd(x_{n+1})$ is positive, we know: $|cd(x_{n+1})| = cd(x_{n+1}) \leq (1) cd(x_n) + |ic_e(x_n x_{n+1})| = cd(x_n) + iv_{ic_n} \leq |cd(x_n)| + iv_{ic_n}.$

– If $CE? (n) = c_i \in Rst$, the left side of the constraint does not change with respect to the case with $n$. We can decompose the right side as before and as a result we have to prove:

$$0 \leq |cd(x_n)| - |cd(x_{n+1})| + iv_{rst_n}$$

By the definition of $Rst$ we have $iv_{rst_n} = rst_i(x_n) \geq |cd(x_{n+1})|$ which is sufficient to prove that $|cd(x_n)| - |cd(x_{n+1})| + iv_{rst_n}$ is positive.

Note that the generated constraint does not contain variables from the end of the phase $\overline{x}$. Moreover, we did not use the fact that the execution is finite at any point of the proof. Therefore, this constraint is also valid for infinite executions.
Soundness of Max and Min Strategy for $\leq$ (Lemma 2) Given a constraint $iv \leq |l(x^r)| \in Pms^{c_i}$, if we manage to classify all the $c_e \in ph$ into $Dc$, $Ic$, and $Rst$, we generate:

$$[iv] \leq iv_{\max} + \sum_{c_e \in Ic} smiv_{ic_e}, \quad iv_{\max} \leq \max \limits_{c_e \in Rst} ([iv]_{rst_e}, iv_{cd}), \quad iv_{cd} \leq |cd(x^s)|$$

We will prove that for any instance $iv_j$ of $iv$, either:

- $iv_j \leq |cd(x^r)| + \sum_{c_e \in Ic} smiv_{ic_e}$;
- or there is a $c_e \in Rst$ such that:
  $$iv_j \leq [iv]_{rst_e} + \sum_{c_e \in Ic} smiv_{ic_e}.$$  

If all instances $iv_j$ are smaller or equal than an amount, $[iv]$ (which is the biggest instance) is also smaller or equal than such amount. Note that if we generated a constraint, we have that for every $c_e \in ph$, $c_e$ belongs to $Dc$, $Ic$ or $Rst$. Given an instance $iv_j$ occurring a the j-th evaluation of a CE, $CE?_{}(j) = c_i$ (we extracted the constraint from $Pms^{c_i}$) and it is bounded by the candidate at that point $iv_j \leq |l(x^r_j)| \leq |cd(x^r_j)|$. We consider the CE evaluations that happen before. Consider the last CE evaluation such that it belongs to $Rst$ $CE?_{}(l) = c_e \in Rst$. The sequence of evaluations from the index $l$ to $j$ contains only CE evaluations of $c_e \in Dc$ or $c_e \in Ic$.

- For each evaluation such that $CE?_{}(k) = c_e \in Dc$, we have $|cd(x^r_{k+1})| \leq |cd(x^r_k)|$.
- For each evaluation such that $CE?_{}(k) = c_e \in Ic$, we have $|cd(x^r_{k+1})| \leq |cd(x^r_k)| + iv_{ic_e,k}$.

* $(cd(x^r_{k+1}) \leq cd(x^r_k) + ic_e(x^r_k x^r_{k+1})$ implies $|cd(x^r_{k+1})| \leq |cd(x^r_k)| + iv_{ic_e,k}$.

This way, we have:

$$iv_j \leq |cd(x^r_{j+1})| + \sum_{c_e \in Ic} (psmiv^{ce}_{ic_e}[l + 1..j])$$

Given that $CE?_{}(l) = c_e \in Rst$, we have that $|cd(x^r_{l+1})| \leq |rst_r(x^r_l)| = iv_{rst_r,l}$ which by definition is $iv_{rst, l} \leq [iv]_{rst_r}$. Therefore, we conclude:

$$iv_j \leq [iv]_{rst_r} + \sum_{c_e \in Ic} (psmiv^{ce}_{ic_e}[l + 1..j]) \leq [iv]_{rst_r} + \sum_{c_e \in Ic} smiv_{ic_e}$$

If there is no $l \leq j$ such that $CE?_{}(l) = c_e \in Rst$, we can carry the transformation up to the beginning of the phase execution and obtain:

$$iv_j \leq |cd(x^r)| + \sum_{c_e \in Ic} (psmiv^{ce}_{ic_e}[1..j]) \leq |cd(x^r)| + \sum_{c_e \in Ic} smiv_{ic_e}$$

The maximum of the different cases for the different $c_e \in Rst$ corresponds to the constraint generated. The proof for $|iv| \geq |l(x^r)| \in Pms^{c_i}$ is analogous.
Soundness of Triangular Sum Strategy (Lemma 4) We prove that, given a candidate \( cd(\overline{x}) \) such that we could classify every \( c_e \in ph \) into the sets \( CntTri \) and \( NoCnt \) according to their definitions in Sec. A.2. The following constraints are valid:

\[
\sum_{c_e \in CntTri} cntTri_e \succ |cd(\overline{x})| \cdot iv_{its} + \frac{q}{2} iv_{its}^2 - \frac{q}{2} iv_{its} \cdot iv_{its} = \sum_{c_e \in CntTri} smiv_{it}
\]

Which is a merged version of the constraints stated in Sec. A.2. Here \( q = \max_{c_e \in CntTri} (q_e) \) if \( \triangleright \) is \( \leq \) or \( q = \min_{c_e \in CntTri} (q_e) \) otherwise. The variable \( iv_{its} \) represents the number of evaluations of CE in the phase evaluation s.t. \( c_e \in CntTri \).

We introduce the following auxiliary notion:

**Definition 9 (Partial count).** \( iv_{its}[1..n] \) is the partial count of \( CntTri \) in \([1..n]\) and it represents the number of CE evaluations of \( c_e \in CntTri \) in the segment \( 1 \leq j \leq n \) of the phase evaluation. Note that we have \( iv_{its} = iv_{its}[1..f] \).

Then, we define the following lemma:

**Lemma 5.** For all \( n \) in the phase evaluation \( cd(\overline{x}_n) \triangleright cd(\overline{x}_1) + q \cdot iv_{its}[1..n] \).

**Proof.** we prove it by induction over \( n \).

- Base case: For \( n = 1 \), the interval in \( iv_{its}[1..1] \) is empty, \( iv_{its}[1..1] = 0 \) and \( cd(\overline{x}_1) \triangleright cd(\overline{x}_1) + 0 \).
- Inductive case: We assume \( cd(\overline{x}_n) \triangleright cd(\overline{x}_1) + q \cdot iv_{its}[1..n] \) and prove it for \( n + 1 \). We distinguish two cases:
  - If \( CE?(n) \in CntTri \), we have:
    \[
    cd(\overline{x}_{n+1}) \triangleright cd(\overline{x}_n) + q \triangleright cd(\overline{x}_1) + q \cdot iv_{its}[1..n] + q
    = (IH) \quad cd(\overline{x}_1) + q \cdot (iv_{its}[1..n] + 1) = cd(\overline{x}_1) + q \cdot (iv_{its}[1..n+1])
    \]
  - If \( CE?(n) \in NoCnt \), we have:
    \[
    cd(\overline{x}_{n+1}) = cd(\overline{x}_n) = (IH) \quad cd(\overline{x}_1) + q \cdot iv_{its}[1..n]
    = cd(\overline{x}_1) + q \cdot iv_{its}[1..n+1]
    \]

According to the definition of \( CntTri \), we have:

\[
\sum_{c_e \in CntTri} cntTri_e \succ \sum_{1 \leq j < f \land CE?(j) \in CntTri} cd(\overline{x}_j)
\]

We prove that the left side of the constraint that we generate is a valid approximation of such constraint:
\[ \sum_{1 \leq j < f \in CntTri} cd(x_j) \]

1. Because of Lemma 5.
2. Definition of \( iv_{its}[1..n] \) and distributivity.
3. Express sum as indexed sum.
4. Definition of \( iv_{its}[1..n] \): \( iv_{its} = iv_{its}[1..f] \).
5. Solve arithmetic sequence.