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

Program 1	Cost relations:
1 $\{x > 0, y > 0, z > 0\}$	1: $p1(x, y, z) = wh3(x, y, x_o, y_o) + wh9(y_o, z)$
2 **void** p1(**int** x,y,z){	$\{x > 0, y > 0, z > 0\}$
3 **while**(x>0) {	2: $wh3(x, y, x_o, y_o) = 0 \quad \{x = x_o = 0, y_o = y\}$
4 x−−;	3: $wh3(x, y, x_o, y_o) = wh6(y_1, y_2) + wh3(x', y', x_o, y_o)$
5 y++;	$\{x > 0, x' = x - 1, y_1 = y + 1, y' = y_2\}$
6 **while**(y>0 && ∗)	4: $wh6(y, y_o) = 0 \quad \{y = y_o\}$
7 y−−;//$tick(2)$;	5: $wh6(y, y_o) = 2 + wh6(y', y_o) \quad \{y \geq 1, y' = y - 1\}$
8 }	6: $wh9(y, z) = 0 \quad \{y \leq 0\}$
9 **while**(y>0){	7: $wh9(y, z) = wh12(0, z) + wh9(y', z)$
10 y−−;	$\{y \geq 1, y' = y - 1, z > 0\}$
11 **int** i=0;	8: $wh12(i, z) = 0 \quad \{i \geq z\}$
12 **while**(i<z)	9: $wh12(i, z) = 1 + wh12(i', z) \quad \{i < z, i' = i + 1\}$
13 i++;//$tick(1)$;	Upper bound = $max(2, z) * (x + y)$
14 }}	Lower bound = $min(2, z) * (x + y)$

Fig. 1. Program 1 and its cost relations

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 $max(2, z) * (x + y)$ and $min(2, z) * (x + y)$, 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., y_o) 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: $y = y_o$. 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 $*$).

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 $2 * (x + y)$ (in terms of $p1$'s input parameters). But the loop can be entered x times and still its total cost is at most $2 * (x + y)$ and not $2 * (x + y) * x$ 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 $min(2, z) * (x + y)$. 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 $2*(x+y-y_0)$ and $z*(y_0)$, 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 $\overline{x}$ represents a sequence of variables $x_1, x_2, \cdots, x_n$ of any length. We represent the concatenation of $\overline{x}$ and $\overline{y}$ as $\overline{xy}$. 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 α to the domain V. A *linear expression* has the form $l(\overline{x}) := q_0 + q_1 * x_1 + \cdots + q_n * x_n$ where $q_i \in \mathbb{Q}$ and $x_1, x_2, \cdots, x_n$ are variables. A *linear constraint* over $\overline{x}$ is $lc(\overline{x}) := l(\overline{x}) \geq 0$ where $l(\overline{x})$ 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* $\varphi(\overline{x})$ is a conjunction of linear constraints $lc_1(\overline{x}) \wedge lc_2(\overline{x}) \wedge \cdots \wedge lc_n(\overline{x})$. A constraint set $\varphi(\overline{x})$ is *satisfiable* if there exists an assignment $\alpha : V \mapsto \mathbb{Q}$ such that $\varphi(\overline{\alpha(x)})$ is valid (α satisfies $\varphi(\overline{x})$). We say that $\varphi(\overline{x}) \Rightarrow \varphi'(\overline{x})$ if every assignment that satisfies $\varphi(\overline{x})$ satisfies $\varphi'(\overline{x})$ 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 $c := \langle C(\overline{x}) = q + \sum_{i=1}^{n} D_i(\overline{y_i}), \varphi(\overline{xy})\rangle$*, where* $q \in \mathbb{Q}$*;* C *and* D_i *are cost relation symbols; and* $\varphi(\overline{xy})$ *is a constraint set that relates the variables on the left side* $C(\overline{x})$ *and those in the* $D_i(\overline{y_i})$ *where* $\overline{y} = \overline{y_1 y_2 \cdots y_n}$*.*

A cost equation (CE) $\langle C(\overline{x}) = q + \sum_{i=1}^{n} D_i(\overline{y_i}), \varphi(\overline{xy})\rangle$ states that the cost of $C(\overline{x})$ is q plus the sum of the costs of each $D_i(\overline{y_i})$. The constraint set $\varphi(\overline{xy})$ serves two purposes: it restricts the applicability of the equation with respect to the input variables $\overline{x}$ and it relates the variables $\overline{x}$ with each $\overline{y_i}$. One can view a CR C as a non-deterministic function that executes a cost equation in C. Given a cost equation $\langle C(\overline{x}) = q + \sum_{i=1}^{n} D_i(\overline{y_i}), \varphi(\overline{xy})\rangle$, 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* $(c_1 \vee \cdots \vee c_n)^+$ *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* $(c_1 \vee \cdots \vee c_n)^+$*.*

For instance, the CR *wh*6 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]'

$$
\begin{array}{ll}
1.1: & p1(x,y,z) = wh3[(3.1 \vee 3.2)^{+}2](x,y,x_o,y_o) + wh9[6](y_o,z) \\
 & \qquad \{x>0, y>0, z>0, \mathbf{x_o = 0}, \mathbf{y_o \leq 0}\} \\
1.2: & p1(x,y,z) = wh3[(3.1 \vee 3.2)^{+}2](x,y,x_o,y_o) + wh9[7.1^{+}6](y_o,z) \\
 & \qquad \{x>0, y>0, z>0, \mathbf{x_o = 0}, \mathbf{y_o > 0}, \mathbf{x+y \geq y_o}\} \\
3.1: & wh3(x,y,x_o,y_o) = wh6[4](y_1,y_2) + wh3(x',y',x_o,y_o) \\
 & \qquad \{x>0, x' = x-1, y_1 = y+1, y' = y_2, \mathbf{y_2 = y_1}\} \\
3.2: & wh3(x,y,x_o,y_o) = wh6[5^{+}4](y_1,y_2) + wh3(x',y',x_o,y_o) \\
 & \qquad \{x>0, x' = x-1, y_1 = y+1, y' = y_2, \mathbf{y_2 < y_1}\} \\
7.1: & wh9(y,z) = wh12[9^{+}8](0,z) + wh9(y',z) \quad \{y \geq 1, y' = y-1, z>0\}
\end{array}
$$

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 $\langle C_1, C_2 \dots C_n \rangle$. 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 $j > i$ annotated with chains of C_j. Its general form is: $\langle C_i(\overline{x}) = q + \sum_{i=1}^{n} Dch_i(\overline{y_i}) + C_i(\overline{x'}), \varphi(\overline{xx'y}) \rangle$ where $D \in \{C_{i+1} \dots C_n\}$ if it is recursive or without the summand $+C_i(\overline{x'})$ if it is non-recursive.

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

2.2 Refined cost relation semantics

Cost relations can be evaluated to a cost with respect to a variable assignment $\alpha : V \mapsto \mathbb{Q}$. We define the evaluation relation $\Downarrow$ 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 $\langle C(\overline{x}) = k_0 + \sum_{i=1}^{n} Dch_i(\overline{y_i}), \varphi(\overline{xx'y}) \rangle$ as $nrc(\overline{x})$. Rule (Non-recursive CE) extends the assignment α to α' such that it is defined for $\overline{y}$ and the constraint set of the CE is valid $\varphi(\alpha'(\overline{xy}))$. The cost of $nrc(\overline{x})$ with variable assignment α is the sum of the costs of the evaluations of the chains referenced by $nrc(\overline{x})$ plus k_0. A recursive CE $\langle C(\overline{x}) = k_0 + \sum_{i=1}^{n} Dch_i(\overline{y_i}) + C(\overline{x'}), \varphi(\overline{xx'y}) \rangle$ is written $rc(\overline{xx'})$. Because a recursive CE

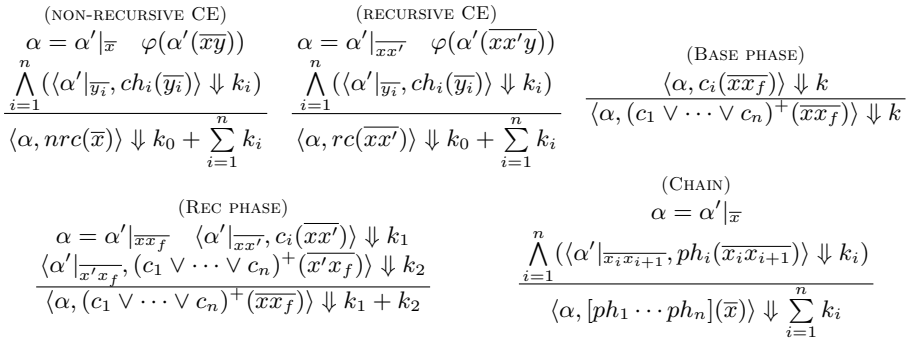

Fig. 3. Semantics for the evaluation of chains, phases and cost equations

always appears within a recursive phase $(c_1 \vee \cdots \vee c_n)^+$, 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 $\overline{x_f}$ in the phase representation $(c_1 \vee \cdots \vee c_n)^+(\overline{xx_f})$ 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, 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, ph_n will be $(c_1 \vee \cdots \vee c_n)^+$ 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 $(\overline{x_f})$. 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

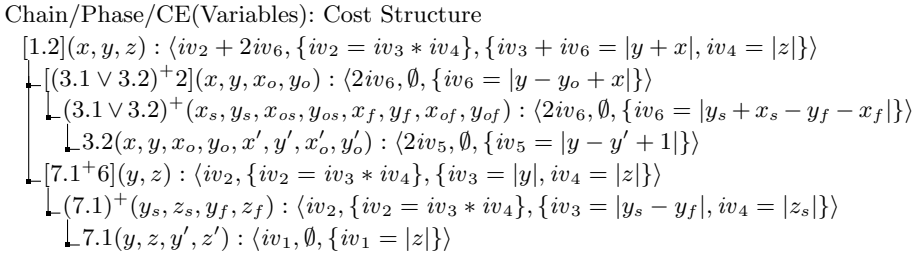

Fig. 4. Some of the cost structures of Program 1

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 $FC(\overline{x})$ that relate intermediate variables with the variables of the CRs ($\overline{x}$). The formal definition of cost structures is as follows:

Definition 3 (Cost Structure). *A* cost structure *is a tuple* $\langle E, IC, FC(\overline{x})\rangle$.

- *E is the main cost expression and is a linear expression $l(\overline{iv})$ over intermediate variables. Intermediate variables always represent positive numbers.*
- *Let $\bowtie$ be $\leq$ or $\geq$. IC is a set of* non-final *constraints of the form $\sum_{k=1}^{m} iv_k \bowtie SE$ where SE can be $SE := l(\overline{iv}) \mid iv_i * iv_j \mid max(\overline{iv}) \mid min(\overline{iv})$.*
- *$FC(\overline{x})$ is a set of* final *constraints of the form $\sum_{k=1}^{m} iv_k \bowtie |l(\overline{x})|$ where $|l(\overline{x})| := max(l(\overline{x}), 0)$ and $l(\overline{x})$ is a linear expression over the CR variables.*

Even though the constraints in IC and $FC(\overline{x})$ 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 $iv_6 + iv_3 = |y + x|$ 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

$wh12[9^+8]$ (See Fig. 2) and $\langle 0, \emptyset, \emptyset \rangle$ (See CE 6 in Fig. 1). Then, we compute the cost of phase 7.1^+. In phase 7.1^+ CE 7.1 is evaluated a number of times and each time it has a cost $\langle iv_1, \emptyset, \{iv_1 = |z|\} \rangle$. The cost of 7.1^+ is the sum of all these costs. In particular iv_2 corresponds to the sum of all the copies of iv_1 of all the evaluations of CE 7.1. The variables iv_3 and iv_4 have an auxiliary role. They maintain the two parts of the cost separated $|y_s - y_f|$ and $|z_s|$ and, together with the non-final constraint, represent a non-linear bound. Finally, the cost of $[7.1^+6]$ is the sum of the costs of 7.1^+ and 6 but expressed only in terms of the initial variable values y and z. The process is similar for other CRs. In CR $wh3$, we compute the costs for CEs 3.1 and 3.2 and 2, we combine the ones from 3.1 and 3.2 to obtain the cost of $(3.1 \vee 3.2)^+$ which in turn we combine with the cost of 2 to obtain the cost of $[(3.1 \vee 3.2)^+2]$. Here, iv_6 represents the sum of all iv_5 of all the evaluations of CE 3.2 in phase $(3.1 \vee 3.2)^+$.

Definition 4 (Valid Cost Structure). *Let $T(\overline{x})$ be a chain, phase or CE. The cost structure $\langle E, IC, FC(\overline{x}) \rangle$ is* valid *for T if for every $\langle \alpha, T(\overline{x}) \rangle \Downarrow k$, there exists an extension of α denoted α' ($\alpha'|_{\overline{x}} = \alpha$) that assigns all the intermediate variables such that $\alpha'(IC \wedge FC(\overline{x}))$ 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 $\langle E, IC, FC(\overline{x}) \rangle$, 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) $iv_3 + iv_6 \geq y + x$:
$iv_2 + 2iv_6 \geq^{(1)} iv_3 * iv_4 + 2iv_6 \geq^{(2)} iv_3 * |z| + 2iv_6$
$\geq^{(3)} min((|y + x| * |z|) + 0,\ 0 + 2|y + x|) = min(|z|, 2) * |y + x|$.

4 Cost Structures of Cost Equations and Chains

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

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

$$\langle k_0 + \sum_{i=1}^{n} E_{ch_i}, \bigcup_{i=1}^{n} (IC_{ch_i}), \bigcup_{i=1}^{n} (FC_{ch_i}(\overline{y_i})) \rangle$$

We add the main cost expressions E_{ch_i} plus k_0 and join the constraint sets IC_{ch_i} and $FC_{ch_i}(\overline{y_i})$. Note that in the base case (i.e. when $n = 0$), the resulting cost structure is simply $\langle k_0, \emptyset, \emptyset \rangle$. Unfortunately, the final constraints in $\bigcup_{i=1}^{n}(FC_{ch_i}(\overline{y_i}))$ 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 $(\sum_{k=1}^{m} iv_k \bowtie |l(\overline{y})|)$. 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 $\sum_{k=1}^{m} iv_k \bowtie l(\overline{y})$. Let FC^+ be the set of all constraints obtained thus from $\bigcup_{i=1}^{n} FC_{ch_i}(\overline{y_i})$. We perform (Fourier-Motzkin) quantifier elimination on $\exists \overline{y}.(FC^+ \wedge \varphi(\overline{xx'y}))$ 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 $[(3.1 \vee 3.2)^+2]$ and $[7.1^+6]$ from Fig. 4 into that of CE 1.2, instantiated according to CE 1.2 with variables (x, y, x_o, y_o) and (y_o, z), respectively. The resulting expression is: $\langle iv_2 + 2iv_6, \{iv_2 = iv_3 * iv_4\}, \{iv_6 = |y - y_o + x|, iv_3 = |y_o|, iv_4 = |z|\}\rangle$. 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 $\exists x_o, y_o.(FC^+ \wedge \varphi_{1.2})$. 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 $iv_3 = y_0$) 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 $\sum_{k=1}^{m} iv_k \bowtie |l(\overline{y})|$ be a constraint, if we find $l'(\overline{xx'})$ such that $\varphi(\overline{xx'y}) \Rightarrow l(\overline{y}) \bowtie l'(\overline{xx'})$, then we have that $\sum_{k=1}^{m} iv_k \bowtie |l'(\overline{xx'})|$ holds as well.[2] We find $l'(\overline{xx'})$ by creating a linear template of $l'(\overline{xx'})$ and finding coefficients that satisfy $\varphi(\overline{xx'y}) \Rightarrow l(\overline{y}) \bowtie l'(\overline{xx'})$ using Farkas' Lemma.

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

[2] 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 $ph = (c_1 \vee \cdots \vee c_n)^+$ be a phase. Our objective is to compute a valid cost structure $\langle E_{ph}, IC_{ph}, FC_{ph}(\overline{x_s x_f})\rangle$ for the phase ph. Such a cost structure must be expressed in terms of initial values of the variables ($\overline{x_s}$) and the values of the variables in the last recursive call of the phase ($\overline{x_f}$) 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 $\langle E_{c_i}, IC_{c_i}, FC_{c_i}(\overline{xx'})\rangle$ be a valid cost structure of c_i and let $\#c_i$ be the number of times c_i is evaluated in ph. $\langle E_{c_ij}, IC_{c_ij}, FC_{c_ij}(\overline{x_{c_ij}x'_{c_ij}})\rangle$ 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: $\overline{x_{c_ij}x'_{c_ij}}$.*

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 $c_i \in ph$:

$$\langle \sum_{i=1}^{n} \sum_{j=1}^{\#c_i} E_{c_ij},\ \bigcup_{i=1}^{n} \bigcup_{j=1}^{\#c_i} (IC_{c_ij}),\ \bigcup_{i=1}^{n} \bigcup_{j=1}^{\#c_i} (FC_{c_ij}(\overline{x_{c_ij}x'_{c_ij}}))\rangle$$

Based on this, we generate a cost structure $\langle E_{ph}, IC_{ph}, FC_{ph}(\overline{x_s x_f})\rangle$ in three steps: (1) we transform the expression $\sum_{i=1}^{n}\sum_{j=1}^{\#c_i} E_{c_ij}$ 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 $FC_{ph}(\overline{x_s x_f})$ using the CEs' final constraints $FC_{c_i}(\overline{x_{c_i}x'_{c_i}})$ and the CE definitions (in Sec. 5.2).

In order to transform $\sum_{i=1}^{n}\sum_{j=1}^{\#c_i} E_{c_ij}$ 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 $\langle E_{c_i}, IC_{c_i}, FC_{c_i}(\overline{xx'})\rangle$. The intermediate variable $smiv := \sum_{j=1}^{\#c_i} iv_j$ is the sum of all instances of iv in the different evaluations of c_i in the phase.*

Now, we can reformulate each $\sum_{j=1}^{\#c_i} E_{c_ij}$ into a linear expression in terms of $smiv$. Let $E_{c_i} := q_0 + q_1 * iv_1 + \cdots + q_m * iv_m$, we have that $\sum_{j=1}^{\#c_i} E_{c_ij} = q_0 * \#c_i + q_1 * smiv_1 + \cdots + q_m * smiv_m$ (where $\#c_i$ is also an intermediate variable). If we do this transformation for each i in $\sum_{i=1}^{n}\sum_{j=1}^{\#c_i} E_{c_ij}$, we obtain a valid cost expression for the phase E_{ph}.

Example 5. Consider phase $(3.1 \vee 3.2)^+$. Let $E_{3.1} = 0$ and $E_{3.2} = 2iv_5$. The main cost expression of the phase is $E_{(3.1\vee3.2)^+} = \sum_{j=1}^{\#c_{3.1}} 0 + \sum_{j=1}^{\#c_{3.2}} 2iv_{5j} = 2smiv_5$ (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 $\sum_{k=1}^{m} iv_k \bowtie SE \in IC_{c_i}$, we sum up all its instances $\sum_{j=1}^{\#c_i} \sum_{k=1}^{m} iv_{kj} \bowtie \sum_{j=1}^{\#c_i} SE_j$ and reformulate the constraint using *smiv* variables. We reformulate the left-hand side directly: $\sum_{j=1}^{\#c_i} \sum_{k=1}^{m} iv_{kj} = \sum_{k=1}^{m} smiv_k$ 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* $\lceil iv \rceil := \max_{1 \le j \le \#c_i}(iv_j)$ *and* $\lfloor iv \rfloor := \min_{1 \le j \le \#c_i}(iv_j)$ *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 $\lfloor iv \rfloor_p$ or $\lceil iv \rceil_p$ depending on whether $\bowtie$ is $\le$ or $\ge$:
  $\sum_{j=1}^{\#c_i} SE_j \le smiv_k * \lceil iv \rceil_p$ and $\sum_{j=1}^{\#c_i} SE_j \ge smiv_k * \lfloor iv \rfloor_p$.[3]
- $SE := max(\overline{iv})$ or $min(\overline{iv})$: 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 $iv_k \bowtie 1$ and $iv_p \bowtie SE$ to IC_{c_i} so they are later transformed. This way, $smiv_p$ is not generated ($\lceil iv \rceil_p$ or $\lfloor iv \rfloor_p$ 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 $\lfloor iv \rfloor$ and $\lceil iv \rceil$ 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 $\lfloor iv \rfloor$ and $\lceil iv \rceil$ variables. Let $iv \le SE \in IC_{c_i}$ (the $\ge$ case is symmetric). We distinguish cases for SE:[4]

- $SE := q_0 + q_1 * iv_1 + \cdots + q_m * iv_m$: Let $V_k := \lceil iv \rceil_k$ if $q_k \ge 0$ or $V_k := \lfloor iv \rfloor_k$ if $q_k < 0$. We generate $\lceil iv \rceil \le q_0 + q_1 * V_1 + \cdots + q_m * V_m$.
- $SE := iv_k * iv_p$: We generate $\lceil iv \rceil \le \lceil iv \rceil_k * \lceil iv \rceil_p$.
- $SE := max(iv_1 \cdots iv_n)$: We generate $\lceil iv \rceil \le max(\lceil iv \rceil_1 \cdots \lceil iv \rceil_n)$.
- $SE := min(iv_1 \cdots iv_n)$: We generate $\lceil iv \rceil \le \lceil iv \rceil_k$ (for $1 \le k \le n$).

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

[3] We could also approximate to $\lfloor iv \rfloor_k * smiv_p$ and $\lceil iv \rceil_k * smiv_p$ 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 $\le$ can be split ($\sum_{k=1}^{m} iv_k \le SE$ implies $iv_k \le SE$ for $1 \le k \le m$). But this is not the case for the constraints with $\ge$.

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 $ph = (c_1 \vee \cdots \vee c_n)^+$. 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 $\langle E_{c_i}, IC_{c_i}, FC_{c_i}(\overline{xx'})\rangle$ the algorithm maintains two sets of *pending* constraints:
(1) $Psums^{c_i}$ is initialized with the constraints $\sum_{k=1}^{m} iv_k \bowtie |l(\overline{xx'})| \in FC_{c_i}(xx')$ such that some $smiv_k$ appear in our phase cost structure (in E_{ph} or IC_{ph}) and $iv_{it_i} \leq 1$ and $iv_{it_i} \geq 1$ if $\#c_i$ appears in our phase cost structure. The variable iv_{it_i} represents the number of times c_i is evaluated and $smiv_{it_i} = \#c_i$.
(2) Pms^{c_i} is initialized with the constraints $iv \bowtie |l(\overline{xx'})| \in FC_{c_i}(xx')$ such that $\lceil iv \rceil$ or $\lfloor iv \rfloor$ 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 $\sum_{k=1}^{m} iv_k \bowtie |l(\overline{xx'})| \in Psums^{c_i}$, the strategy will try to find a linear expression that approximates the sum $\sum_{j=1}^{\#c_i} |l(\overline{x_{c_i j} x'_{c_i j}})|$ in terms of the initial and final variables of the phase $(\overline{x_s x_f})$.

Let us consider first the simple case where c_i is the only CE in the phase. The strategy uses the CE's constraint set $\varphi_i(\overline{xx'y})$ and Farkas' Lemma to generate a candidate linear expression $cd(\overline{x})$ such that $\varphi_i(\overline{xx'y}) \Rightarrow (|l(\overline{xx'})| \bowtie cd(\overline{x}) - cd(\overline{x'}) \geq 0)$. If a candidate $cd(\overline{x})$ is found, we have:

$$\sum_{j=1}^{\#c_i} |l(\overline{x_{c_i j} x'_{c_i j}})| \bowtie \sum_{j=1}^{\#c_i} (cd(\overline{x_{c_i j}}) - cd(\overline{x'_{c_i j}})) = cd(\overline{x_s}) - cd(\overline{x_f})$$

This is because each intermediate $-cd(\overline{x'_{c_i j}})$ and $cd(\overline{x_{c_i j+1}})$ cancel each other $(cd(\overline{x'_{c_i j}}) = cd(\overline{x_{c_i j+1}}))$. Therefore, the constraint
$\sum_{k=1}^{m} smiv_k \bowtie |cd(\overline{x_s}) - cd(\overline{x_f})|$ 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 $Psums^9 = \{iv_{it_9} \leq 1, iv_{it_9} \geq 1\}$. The strategy generates the candidate $-i$ and the final constraints

	Condition when $\bowtie$ is $\leq$	Condition when $\bowtie$ is $\geq$	Defines
Cnt	$(\sum_{k=1}^{m} iv_k \bowtie \lvert l'(\overline{xx'})\rvert) \in Psums^{c_e} \wedge \lvert l'(\overline{xx'})\rvert \bowtie cd(\overline{x}) - cd(\overline{x'}) \geq 0$		$cnt_e = \sum_{k=1}^{m} smiv_k$
Dc	$0 \leq dc_e(\overline{xx'}) \leq cd(\overline{x}) - cd(\overline{x'})$	$dc_e(\overline{xx'}) \geq cd(\overline{x}) - cd(\overline{x'})$	$iv_{dc_e} = \lvert dc_e(\overline{xx'})\rvert$
Ic	$ic_e(\overline{xx'}) \geq cd(\overline{x'}) - cd(\overline{x})$	$0 \leq ic_e(\overline{xx'}) \leq cd(\overline{x'}) - cd(\overline{x})$	$iv_{ic_e} = \lvert ic_e(\overline{xx'})\rvert$
Rst	$cd(\overline{x'}) \bowtie \lvert rst(\overline{x})\rvert$		$iv_{rst_e} = \lvert rst_e(\overline{x})\rvert$

Fig. 5. Classes of CE c_e w.r.t a candidate $cd(\overline{x})$, their condition and defined term

$smiv_{it_9} \leq |i_f - i_s|$ and $smiv_{it_9} \geq |i_f - i_s|$. Later $|i_f - i_s|$ will become $|z - i|$ in $[9^+8]$ and $|z|$ in CE 7.1. The variable $smiv_{it_9}$ corresponds to iv_1 in Fig. 4.

If the phase contains other CEs c_e ($e \neq i$), we have to take their behavior into account. E.g. suppose that we have another c_e ($e \neq i$) that increments our candidate by two ($\varphi_e(\overline{xx'y}) \Rightarrow cd(\overline{x'}) = cd(\overline{x}) + 2$). Let $\#c_e$ be the number of evaluations of c_e, the sum is $\sum_{j=1}^{\#c_i} cd(\overline{x_{c_ij}}) - cd(\overline{x'_{c_ij}}) = cd(\overline{x_s}) - cd(\overline{x_f}) + 2 * \#c_e$. That is, the sum computed for the simple case $cd(\overline{x_s}) - cd(\overline{x_f})$ plus the sum of all the increments to the candidate $2 * \#c_e$ effected by CE c_e.

In the general case, the strategy generates a candidate (using c_i constraint set $\varphi_i(\overline{xx'y})$ and Farkas' Lemma as before); it classifies the CEs of the phase $c_e \in ph$ (including c_i) 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 c_e into a class, its condition has to be implied by the corresponding CE's constraint set $\varphi_e(\overline{xx'y})$. This implication can be verified and the unknown linear expressions $dc_e(\overline{xx'})$ $ic_e(\overline{xx'})$ or $rst_e(\overline{xx'})$ (For the classes Dc, Ic and Rst respectively) can be inferred using Farkas' Lemma. The considered classes in this strategy are[5]:

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

[5]The class Rst will be used and explained in the *Max-Min* strategy.

Lemma 1. *Let* $!\bowtie$ *be the reverse of* $\bowtie$ *(e.g.* $\geq$ *if* $\bowtie$*=*$\leq$*). If we classify every* $c_e \in ph$ *into* Cnt*,* Ic *or* Dc *w.r.t.* $cd(\overline{x})$*, the following constraints are valid:*

$$\sum_{c_e \in Cnt} cnt_e \bowtie iv_{cd+} - iv_{cd-} + \sum_{c_e \in Ic} smiv_{ic_e} - \sum_{c_e \in Dc} smiv_{dc_e} \ ,$$
$$iv_{cd+} \bowtie |cd(\overline{x_s}) - cd(\overline{x_f})| \ , \quad iv_{cd-} ! \bowtie | - cd(\overline{x_s}) + cd(\overline{x_f})|$$

These are the constraints generated by the *Inductive Sum* strategy. Note that iv_{cd+} and $-iv_{cd-}$ represent the positive and negative part of $cd(\overline{x_s}) - cd(\overline{x_f})$. The constraints bind the sum of all $smiv$ in cnt_e (for each $c_e \in Cnt$) to $cd(\overline{x_s}) - cd(\overline{x_f})$ 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, $cd(\overline{x_s}) - cd(\overline{x_f})$ is guaranteed to be positive (the candidate is never incremented) and we can eliminate the summand $-iv_{cd-}$ (and its corresponding constraint $iv_{cd-} ! \bowtie | - cd(\overline{x_s}) + cd(\overline{x_f})|$).

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: $iv_{ic_e} \bowtie |ic(\overline{xx'})|$ is added to $Psums^{c_e}$ for each $c_e \in Ic$, and $iv_{dc_e} ! \bowtie |dc(\overline{xx'})|$ is added to $Psums^{c_e}$ for each $c_e \in Dc$.

Example 7. In phase $(3.1 \vee 3.2)^+$ we have $iv_5 \leq |y - y' + 1| \in Psums^{3.2}$. A valid candidate is $y + x$. The CEs 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 $iv_{it_{3.2}} \geq 1 \in Psums^{3.2}$, 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 $wh3$ 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 $\sum_{k=1}^{m} iv_k \bowtie |l(\overline{xx'})| \in Psums^{c_i}$, it is impossible to infer a linear expression representing $\sum_{j=1}^{\#c_i} |l(\overline{x_j x'_j})|$.

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 $\sum_{j=1}^{\#c_7} |z| = |y| * |z|$ 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 $smiv_1 \leq smiv_{it_{7.1}} * \lceil iv \rceil_{mz}$ (that corresponds to $iv_2 \leq iv_3 * iv_4$

[6] $smiv_{it_{3.1}}$ and $smiv_{it_{3.2}}$ are actually not needed for computing the cost of the program in this case. Therefore, these constraints are never added to the pending sets.

in Fig. 4) and add $iv_{it_{7.1}} \leq 1$ 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 $\sum_{k=1}^{m} iv_k \leq |l(\overline{xx'})| \in Psums^{c_i}$ where $l(\overline{xx'})$ is not a constant, the *Basic Product* strategy generates $\sum_{k=1}^{m} smiv_k \leq smiv_{it_i} * \lceil iv \rceil_p$ and adds the pending constraints $iv_{it_i} \leq 1$ to $Psums^{c_i}$ and $iv_p \leq |l(\overline{xx'})|$ 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 $iv \bowtie |l(\overline{xx'})| \in Pms^{c_i}$ and its role is to generate constraints for Max $\lceil iv \rceil$ and Min $\lfloor iv \rfloor$ variables.

Similarly to the *Inductive Sum* strategy, it generates a candidate $cd(\overline{x})$ using the CE's constraint set $\varphi_i(\overline{xx'y})$ 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 $l(\overline{xx'})$ 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 $\bowtie$ is $\geq$). A fresh intermediate variable is assigned to such reset value $iv_{rst_e} := |rst_e(\overline{x})|$.

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

$$\lceil iv \rceil \leq iv_{max} + \sum_{c_e \in Ic} smiv_{ic_e}, \quad iv_{max} \leq \max_{c_e \in Rst} (\lceil iv \rceil_{rst_e}, iv_{cd}), \quad iv_{cd} \leq |cd(\overline{x_s})|$$

These are the constraints generated by the *Max-Min* strategy. They bind $\lceil iv \rceil$ 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 $\lceil iv \rceil_{rst_e}$. This maximum also includes the candidate $cd(\overline{x_s})$ in case it is never reset.

Finally, the strategy adds the constraints $iv_{ic_e} \leq |ic_e(\overline{xx'})|$ to $Psums^{c_i}$ and $iv_{rst_e} \leq |rst_e(\overline{x})|$ to Pms^{c_i} so $smiv_{ic_e}$ and $\lceil iv \rceil_{rst_e}$ 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 $\lfloor iv \rfloor_{rst_e}$.

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 $\lceil iv \rceil_{mz} \leq |z_s|$ (which corresponds to $iv_4 \leq |z_s|$ 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 $\overline{x_s}$ and to obtain better bounds for sums of expressions whose value varies in each iteration.

6 Soundness

Theorem 1. *Let $T(\overline{x})$ be a chain, a phase or a CE. Then the cost structure $\langle E_T, IC_T, FC_T(\overline{x})\rangle$ 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 *wh*3. 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

UB	1	n	n^2	n^3	$> n^3$	F	T
CoFloCo	3	62	33	2	1	20	1.19
Old	3	55	32	1	1	29	2.11
Loopus	2	56	27	0	2	34	0.03
KoAT	3	45	40	8	4	21	5.12
C4B	1	42	-	-	-	78	1.24

LB	1	n	n^2	n^3	$> n^3$	F	T
CoFloCo	48	85	23	2	0	2	1.89
PUBS	95	38	9	4	0	14	7.58

CoFloCo	KoAT	Loopus	Old	C4B	Pubs(LB)
better	28	20	12	59	60
worse	5	3	1	1	2

Fig. 6. 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.

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

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

1. E. Albert, P. Arenas, S. Genaim, M. Gómez-Zamalloa, and G. Puebla. COSTABS: A Cost and Termination Analyzer for ABS. In O. Kiselyov and S. Thompson, editors, *Proceedings of the 2012 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2012, Philadelphia, Pennsylvania, USA, January 23-24, 2012*, pages 151–154. ACM Press, 2012.
2. E. Albert, P. Arenas, S. Genaim, and G. Puebla. Cost Relation Systems: a Language–Independent Target Language for Cost Analysis. In *Spanish Conference on Programming and Computer Languages (PROLE'08)*, volume 248 of *Electronic Notes in Theoretical Computer Science*, pages 31–46. Elsevier, 2009.
3. E. Albert, P. Arenas, S. Genaim, and G. Puebla. Closed-Form Upper Bounds in Static Cost Analysis. *Journal of Automated Reasoning*, 46(2):161–203, 2011.
4. E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. COSTA: A Cost and Termination Analyzer for Java Bytecode. In *Proceedings of the Workshop on Bytecode Semantics, Verification, Analysis and Transformation (Bytecode)*, Electronic Notes in Theoretical Computer Science, Budapest, Hungary, Apr. 2008. Elsevier.
5. E. Albert, S. Genaim, and A. N. Masud. On the Inference of Resource Usage Upper and Lower Bounds. *ACM Transactions on Computational Logic*, 14(3):22:1–22:35, 2013.
6. C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *SAS*, volume 6337 of *LNCS*, pages 117–133. Springer, 2010.
7. D. E. Alonso-Blas, P. Arenas, and S. Genaim. Precise Cost Analysis via Local Reasoning. In D. V. Hung and M. Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2013.
8. D. E. Alonso-Blas and S. Genaim. On the limits of the classical approach to cost analysis. In A. Miné and D. Schmidt, editors, *Static Analysis - 19th International Symposium, SAS 2012, Deauville, France, September 11-13, 2012. Proceedings*, volume 7460 of *Lecture Notes in Computer Science*, pages 405–421. Springer, Sept. 2012.
9. M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Alternating runtime and size complexity analysis of integer programs. In *TACAS*, 2014.
10. Q. Carbonneaux, J. Hoffmann, and Z. Shao. Compositional certified resource bounds. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming*

Language Design and Implementation, PLDI 2015, pages 467–478, New York, NY, USA, 2015. ACM.

11. S. K. Debray and N. W. Lin. Cost Analysis of Logic Programs. *ACM Transactions on Programming Languages and Systems*, 15(5):826–875, November 1993.
12. S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin. Lower Bound Cost Estimation for Logic Programs. In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.
13. S. Falke, D. Kapur, and C. Sinz. Termination Analysis of C Programs Using Compiler Intermediate Languages. In M. Schmidt-Schauß, editor, *22nd International Conference on Rewriting Techniques and Applications (RTA'11)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41–50, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
14. A. Flores Montoya and R. Hähnle. Resource analysis of complex programs with cost equations. In J. Garrigue, editor, *12th Asian Symposium on Programming Languages and Systems (APLAS'14)*, volume 8858 of *Lecture Notes in Computer Science*, pages 275–295. Springer, Nov. 2014.
15. F. Frohn, M. Naaf, J. Hensel, M. Brockschmidt, and J. Giesl. Lower runtime bounds for integer programs. In *Proceedings of IJCAR'16*. Springer, June 2016.
16. A. Garcia, C. Laneve, and M. Lienhardt. Static analysis of cloud elasticity. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14-16, 2015*, pages 125–136. ACM, 2015.
17. N. Grech, K. Georgiou, J. Pallister, S. Kerrison, J. Morse, and K. Eder. Static analysis of energy consumption for llvm ir programs. In *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '15, pages 12–21, New York, NY, USA, 2015. ACM.
18. S. Gulwani, S. Jain, and E. Koskinen. Control-flow refinement and progress invariants for bound analysis. In *PLDI*, 2009.
19. S. Gulwani, K. K. Mehra, and T. Chilimbi. Speed: Precise and efficient static estimation of program computational complexity. In *POPL*, pages 127–139, New York, NY, USA, 2009. ACM.
20. S. Gulwani and F. Zuleger. The reachability-bound problem. In *PLDI'10*, pages 292–304, New York, NY, USA, 2010. ACM.
21. J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. *SIGPLAN Not.*, 46(1):357–370, Jan. 2011.
22. J. Hoffmann and Z. Shao. Type-based amortized resource analysis with integers and arrays. In M. Codish and E. Sumii, editors, *Functional and Logic Programming*, volume 8475 of *LNCS*, pages 152–168. Springer International Publishing, 2014.
23. A. Serrano, P. López-García, and M. V. Hermenegildo. Resource usage analysis of logic programs via abstract interpretation using sized types. *TPLP*, 14(4-5):739–754, 2014.
24. M. Sinn, F. Zuleger, and H. Veith. A simple and scalable approach to bound analysis and amortized complexity analysis. In *CAV*, volume 8559 of *LNCS*, pages 743–759. Springer, 2014.
25. M. Sinn, F. Zuleger, and H. Veith. Difference constraints: An adequate abstraction for complexity analysis of imperative programs. In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 144–151, 2015.
26. B. Wegbreit. Mechanical Program Analysis. *Communications of the ACM*, 18(9):528–539, September 1975.

27. F. Zuleger, S. Gulwani, M. Sinn, and H. Veith. Bound analysis of imperative programs with the size-change abstraction. In E. Yahav, editor, *Static Analysis*, volume 6887 of *LNCS*, pages 280–297. Springer, 2011.

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 ($\overline{x_s}$). Such bounds are not only valid for $\sum_{j=1}^{\#c_i} |l(\overline{x_j x'_j})|$ but also for any partial sum $\sum_{j=1}^{p} |l(\overline{x_j x'_j})|$ with $p \leq \#c_i$ and they are valid for non-terminating phases where $\overline{x_f}$ is not defined.

In order to compute such a bound, the strategy proceeds as before: Let $\sum_{k=1}^{m} iv_k \leq |l(\overline{xx'})| \in Psums^{c_i}$, it generates a candidate $cd(\overline{x})$ using the constraint set of c_i $\varphi_i(\overline{xx'y})$ and Farkas' Lemma. However, This time the strategy uses the condition $|l'(\overline{xx'})| \leq cd(\overline{x}) - cd(\overline{x'}) \wedge |l'(\overline{xx'})| \leq cd(\overline{x})$. This condition contains the extra conjunction $|l'(\overline{xx'})| \leq cd(\overline{x})$ which allows us to ignore the final value of the candidate $cd(\overline{x_f})$ 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 $cd(\overline{x})$, the following constraints are valid:*

$$\sum_{c_e \in Cntr} cntr_e \leq iv_{cd} + \sum_{c_e \in Ic} smiv_{ic_e} + \sum_{c_e \in Rst} smiv_{rst_e} \ , \qquad iv_{cd} \leq |cd(\overline{x_s})|$$

For each $c_e \in Ic$, the strategy adds the pending constraints $iv_{ic_e} \leq |ic(\overline{xx'})|$ to $Psums^{c_e}$ and for each $c_e \in Rst$, it adds $iv_{rst_e} \leq |rst(\overline{x})|$ 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.

	Condition when $\bowtie$ is $\leq$	Defines
$Cntr$	$(\sum_{k=1}^{m} iv_k \leq \|l'(\overline{xx'})\|) \in Psums^{c_e} \wedge$ $\|l'(\overline{xx'})\| \leq cd(\overline{x}) - cd(\overline{x'}) \wedge \|l'(\overline{xx'})\| \leq cd(\overline{x})$	$cntr_e = \sum_{k=1}^{m} smiv_k$
	Condition when $\bowtie$ is $\leq$ | Condition when $\bowtie$ is $\geq$	Defines
$CntTri$	$(\sum_{k=1}^{m} iv_k \bowtie \|l'(\overline{xx'})\|) \in Psums^{c_e} \wedge$ $(l'(\overline{xx'}) \bowtie cd(\overline{x}) \geq 0 \wedge cd(\overline{x'}) - cd(\overline{x}) \bowtie q_e)$	$cntTri_e = \sum_{k=1}^{m} smiv_k$ $q_e \in \mathbb{Q}$
$NoCnt$	$cd(\overline{x'}) - cd(\overline{x}) = 0$	

Fig. 7. Additional Classes of CE c_e with respect to a candidate $cd(\overline{x})$, their condition and defined term

Program 2	Refined cost relations
1 **for**(**int** x=0;x<n;x++) 2 **for**(**int** y=x;y<n;y++) 3 ;//*tick*(1);	1: $for_1(x,n) = for_2[3^+4](y,n)+ : for_1(x',n)$ $\{y = x, x < n, x' = x + 1\}$ 2: $for_1(x,n) = 0 \quad \{x \geq n\}$ 3: $for_2(y,n) = 1 + for_2(y',n)$ $\{y < n, y' = y + 1\}$ 4: $for_2(y,n) = 0 \quad \{y \geq n\}$ Lower bound=$\mathbf{|n|^2/2 + |n|/2}$

Fig. 8. Program 2 and its refined cost relations

A.2 Triangular Sum Strategy

This strategy represents an alternative to the *Basic Product* strategy for dealing with constraints $\sum_{k=1}^{m} iv_k \bowtie |l(\overline{xx'})| \in Psums^{c_i}$ where $|l(\overline{xx'})|$ 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 $\langle iv_1, \emptyset, \{iv_1 = |n - y|\}\rangle$ which yields a cost of $\langle iv_1, \emptyset, \{iv_1 = |n - x|\}\rangle$ 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: $Psums^1 = \{iv_1 \leq |n - x|, iv_1 \geq |n-x|\}$ and $Pms^1 = \emptyset$. If we apply the basic product strategy to $iv_1 \geq |n-x|$, we would obtain $smiv_1 \geq smiv_{it_1} * \lceil iv \rceil_2$ and later $smiv_{it_1} \geq |n_s - x_s|$ and $\lceil iv \rceil_2 \geq 1$ (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:

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

This expression can be represented with constraints as follows:

$$smiv_1 \geq iv_{p1} - \tfrac{1}{2} iv_{p2} + \tfrac{1}{2} smiv_{it_1} \;\; , \; iv_{p1} \geq iv_{ini} * smiv_{it_1}$$
$$iv_{p2} \leq smiv_{it_1} * smiv_{it_1} \;\; , \qquad iv_{ini} \geq |n_s - x_s|$$

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 $iv_{it_1} \leq 1$ and $iv_{it_1} \geq 1$ (in $Psums^1$), we generate $smiv_{it_1} = |(n_s - x_s) - (n_f - x_f)|$. Once we compute the cost of the complete chain '$[1^+2]$', we will transform $|(n_s - x_s) - (n_f - x_f)|$ into $|n_s - x_s|$ (Because $n_f - x_f$ must be 0 in chain '$[1^+2]$'). If we minimize the cost

of the resulting cost structure, we obtain:

$$\begin{array}{l} |n_s - x_s|^2 - |n_s - x_s|^2/2 + |n_s - x_s|/2 \\ = |n_s - x_s|^2/2 + |n_s - x_s|/2 = \\ =^{(1)} |n_s|^2/2 + |n_s|/2 \quad \text{(because } x_s = 0\text{)} \end{array}$$

In general, given a constraint $\sum_{k=1}^{m} smiv_k \bowtie |l(\overline{xx'})| \in Psums^{c_i}$, the strategy generates a candidate $cd(\overline{x})$ that approximates the cost of one instance of $|l(\overline{xx'})|$, it is positive and it varies by a constant amount $q_i \in \mathbb{Q}$. That is:

$$\varphi_i(\overline{xx'y}) \Rightarrow (l(\overline{xx'}) \bowtie cd(\overline{x}) \geq 0 \wedge cd(\overline{x'}) - cd(\overline{x}) \bowtie q_i)$$

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

Lemma 4. *If we classify every $c_e \in ph$ into $CntTri$ and $NoCnt$ with respect to a candidate $cd(\overline{x})$. Let q be $q := \max_{c_e \in CntTri}(q_e)$ when $\bowtie$ is $\leq$ or $q := \min_{c_e \in CntTri}(q_e)$ when $\bowtie$ is $\geq$. The following constraints are valid:*

$$\sum_{c_e \in CntTri} cntTri_e \bowtie iv_{p1} + \tfrac{q}{2} iv_{p2} - \tfrac{q}{2} iv_{its} \ , \quad iv_{p1} \bowtie iv_{ini} * iv_{its}$$
$$iv_{p2} = iv_{its} * iv_{its}, \quad iv_{its} = \sum_{c_e \in CntTri} smiv_{it_e}, \quad iv_{ini} \bowtie |cd(\overline{x_s})|$$

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 $\#c_e$ of $c_e \in CntTri$. The constraints $iv_{it_e} \leq 1$ and $iv_{it_e} \geq 1$ are added to each $Psums^{c_e}$ such that $c_e \in CntTri$ (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 z. Finally, CE 14 is the exit path of the loop.

We will compute the cost structure of the phase $(10 \vee 11 \vee 12 \vee 13)^+$ based on the cost structures of CEs $10 - 13$. We assume we have the cost structure $\langle 2iv_5, \emptyset, iv_5 \leq |y - y'| \rangle$ for CE 10 and the cost structures of CEs $11 - 13$ are empty. For simplicity, we only consider constraints for upper bounds (with $\leq$).

Program 3	Refined cost relation of the outer loop:

1 **while**(x>0){
2 **if**(∗)
3 **while**(y>0 && ∗)
4 y−−;//*tick*(2);
5 **else if**(∗) y++;
6 **else** y=z;
7 x−−;}
	10: $p3(x,y,z) = wh6[5^{+}4](y,y') + p3(x,y',z)$ $\{x>0, y>y'\geq 0, x'=x-1\}$
	11: $p3(x,y,z) = wh6[4](y,y') + p3(x,y',z)$ $\{x>0, y=y'\geq 0, x'=x-1\}$
	12: $p3(x,y,z) = p3(x',y',z)$ $\{x'=x-1\geq 0, y'=y-1\}$
	13: $p3(x,y,z) = p3(x',y',z)$ $\{x>0, x'=x-1, y=z\}$
	14: $p3(x,y,z) = 0\{x\leq 0\}$
	Upper bound= 2(|y|+max(|x|,|x|*|z|))

Fig. 9. Program 3 and its refined cost relations

The main cost expression of the phase is $2smiv_5$. 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 $Psums^{c_i}$ and Pms^{c_i}.
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 $c_e \in ph$ and the related defined terms $cntr_e$, ic_e, 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 $\langle E, IC, FC(\overline{x})\rangle$ contains all the generated constraints ($NewCs$):

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

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 $\langle E, IC, FC(\overline{x})\rangle$, 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 $\overline{x}$ to the intermediate variables according to the constraints in IC and $FC(\overline{x})$. 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.

State	$Psums^{10} = \{iv_5 \leq \lvert y - y' \rvert\}$
SelConstr	$iv_5 \leq \lvert y - y' \rvert$
Strategy	$ISR:\ cd = y,\ Cntr = \{10\},\ Dc = \{11\},\ Ic = \{12\},\ Rst = \{13\}$
	$cntr_{10} = smiv_5,\ dc_{11} = 0,\ ic_{12} = 1,\ rst_{13} = \lvert z \rvert$
NewCs	$smiv_5 \leq iv_1 + smiv_{ic_{12}} + smiv_{rst_{13}},\quad iv_1 \leq \lvert y \rvert$
State	$Psums^{12} = \{iv_{ic_{12}} \leq 1\}, Psums^{13} = \{iv_{rst_{13}} \leq \lvert z \rvert\}$
SelConstr	$iv_{rst_{13}} \leq \lvert z \rvert$
Strategy	BP
NewCs	$smiv_{rst_{13}} \leq smiv_{it_{13}} * \lceil iv \rceil_2$
State	$Psums^{12} = \{iv_{ic_{12}} \leq 1\}, Psums^{13} = \{iv_{it_{13}} \leq 1\}, Pms^{13} = \{iv_2 \leq \lvert z \rvert\}$
SelConstr	$iv_2 \leq \lvert z \rvert$
Strategy	$MM:\ cd = z,\ Dc = \{10, 11, 12, 13\},\ dc_{10,11,12,13} = 0$
NewCs	$\lceil iv \rceil_2 \leq \lvert z \rvert$
State	$Psums^{12} = \{iv_{ic_{12}} \leq 1\}, Psums^{13} = \{iv_{it_{13}} \leq 1\}$
SelConstr	$iv_{ic_{12}} \leq 1$
Strategy	$ISR:\ cd = x,\ Cntr = \{12, 13\},\ Dc = \{10, 11\}$
	$cntr_{12} = smiv_{ic_{12}},\ cntr_{13} = smiv_{it},\ dc_{10,11} = 1$
NewCs	$smiv_{ic_{12}} + smiv_{it_{13}} \leq \lvert x \rvert$
Done	

Fig. 10. Computation of cost structure of phase $(10 \vee 11 \vee 12 \vee 13)^+$ of program 3.

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: $\langle 1iv_2 + 2iv_6, \{iv_2 = iv_3 * iv_4\}, \{iv_3 + iv_6 = \lvert y + x \rvert, iv_4 = \lvert z \rvert\}\rangle$. 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: $1(iv_3 * |z|) + 2iv_6$ 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:

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

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

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

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

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

As in the previous example, we solve all the constraints with only one variable on the left side first and obtain: $2 * (|y| + smiv_{inc_{12}} + (smiv_{it_{13}} * |z|))$ 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:

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

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: $2 * (|y| + |x| + (|x| * |z|))$.

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 $\langle iv, \emptyset, \{iv \leq |x|, iv \leq |y|\}$, both $|x|$ and $|y|$ are valid upper bounds. These upper bounds are not comparable (we do not know whether x is bigger that 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 consider 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)

UB	1	n	n^2	$\geq n^3$	Failed	# Timeouts	Avg. Time
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

CoFloCo	KoAT	Loopus	Old	Loopus*
better	35	24	8	40
worse	5	416	2	12

Fig. 11. Replication of experimental evaluation of [25]. Complexity, Failed, Timeouts and average Time in seconds

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).

E Soundness Proofs

Theorem 1. *Let $T(\overline{x})$ be a chain, a phase or a CE. Then the cost structure $\langle E_T, IC_T, FC_T(\overline{x})\rangle$ 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 $FC_T(\overline{x})$ 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 φ_{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*, $\lceil iv \rceil$ and $\lfloor iv \rfloor$. 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

E.1 Soundness of Constraints Generated from Final Constraints

An evaluation of a phase $ph = (c_1 \vee \cdots \vee c_n)^+$ 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.

$$\sum_{c_e \in Cnt} cnt_e \bowtie iv_{cd+} - iv_{cd-} + \sum_{c_e \in Ic} smiv_{ic_e} - \sum_{c_e \in Dc} smiv_{dc_e} \ ,$$
$$iv_{cd+} \bowtie |cd(\overline{x_s}) - cd(\overline{x_f})| \ , \quad iv_{cd-}! \bowtie | - cd(\overline{x_s}) + cd(\overline{x_f})|$$

These constraints involve additional intermediate variables iv_{ic_e} and iv_{dc_e} for each $c_e \in Ic$ and $c_e \in Dc$ whose value is defined as $iv_{ic_e} := |ic_e(\overline{xx'})|$ and $iv_{dc_e} := |dc_e(\overline{xx'})|$. The addition of $iv_{ic_e} \bowtie |ic(\overline{xx'})|$ and $iv_{dc_e}! \bowtie |dc(\overline{xx'})|$ 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:

$$\sum_{c_e \in Cnt} cnt_e \bowtie cd(\overline{x_s}) - cd(\overline{x_f}) + \sum_{c_e \in Ic} smiv_{ic_e} - \sum_{c_e \in Dc} smiv_{dc_e}$$

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:

$$\sum_{c_e \in Cnt} \Big(\sum_{smiv_k \in cnt_e} psmiv^{c_e}{}_k[1..n] \Big) \bowtie cd(\overline{x_1}) - cd(\overline{x_n}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) - \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n])$$

We prove that the generalized constraint holds for all n by induction.

Base Case For $n = 1$ we have $0 \bowtie 0$ because the intervals for all $psmiv^{c_e}$ are empty and $cd(\overline{x_1}) - cd(\overline{x_1}) = 0$.

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:

$$\sum_{c_e \in Cnt} (\sum_{smiv_k \in cnt_e} psmiv^{c_e}{}_k[1..n+1]) = \sum_{c_e \in Cnt} \sum_{smiv_k \in cnt_e} psmiv^{c_e}{}_k[1..n] + \sum_{smiv_k \in cnt_i} iv_{kn}$$

  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:

$$\begin{aligned} &cd(\overline{x_1}) - cd(\overline{x_{n+1}}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n+1]) \\ &- \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n+1]) = \\ &cd(\overline{x_1}) - cd(\overline{x_n}) + cd(\overline{x_n}) - cd(\overline{x_{n+1}}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) \\ &- \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n]) \end{aligned}$$

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

$$\sum_{smiv_k \in cnt_i} iv_{kn} \bowtie cd(\overline{x_n}) - cd(\overline{x_{n+1}})$$

  This constraints is directly guaranteed by the classification condition of Cnt: $\sum_{smiv_k \in cnt_i} iv_k \bowtie |l'(\overline{xx'})|) \in Psums^{c_i} \wedge |l'(\overline{xx'})| \bowtie cd(\overline{x}) - cd(\overline{x'})$ which is valid for any evaluation of a $c_e \in Cnt$ and in particular for the evaluation of c_i 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:

$$\sum_{c_e \in Cnt} (\sum_{smiv_k \in cnt_e} psmiv^{c_e}{}_k[1..n+1]) = \sum_{c_e \in Cnt} \sum_{smiv_k \in cnt_e} psmiv^{c_e}{}_k[1..n]$$

  And the right hand side is:

$$\begin{aligned} &cd(\overline{x_1}) - cd(\overline{x_{n+1}}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n+1]) \\ &- \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n+1]) = \\ &cd(\overline{x_1}) - cd(\overline{x_n}) + cd(\overline{x_n}) - cd(\overline{x_{n+1}}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) \\ &- \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n]) - iv_{dc_i n} \end{aligned}$$

Where $iv_{dc_i n}$ is the instance of iv_{dc_i} in the n-th CE evaluation. We have to prove:

$$0 \bowtie cd(\overline{x_n}) - cd(\overline{x_{n+1}}) - iv_{dc_i n}$$

By definition of Dc we have $cd(\overline{x_n}) - cd(\overline{x_{n+1}})$ is positive and $iv_{dc_e n} = |dc_e(\overline{x_n x_{n+1}})| \bowtie cd(\overline{x_n}) - cd(\overline{x_{n+1}})$ 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:

$$\begin{aligned}
&cd(\overline{x_1}) - cd(\overline{x_{n+1}}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n+1]) \\
&- \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n+1]) = \\
&cd(\overline{x_1}) - cd(\overline{x_n}) + cd(\overline{x_n}) - cd(\overline{x_{n+1}}) + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) + iv_{ic_i n} \\
&- \sum_{c_e \in Dc} (psmiv^{c_e}{}_{dc_e}[1..n])
\end{aligned}$$

Therefore, we have to prove:

$$0 \bowtie cd(\overline{x_n}) - cd(\overline{x_{n+1}}) + iv_{ic_i n}$$

This is directly guaranteed by the definition of Ic (given that $|ic_i(\overline{x_n x_{n+1}})| = iv_{ic_i n}$).

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:

$$\sum_{c_e \in Cntr} cntr_e \leq iv_{cd} + \sum_{c_e \in Ic} smiv_{ic_e} + \sum_{c_e \in Rst} smiv_{rst_e} \ , \ iv_{cd} \leq |cd(\overline{x_s})|$$

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 $iv_{ic_e} := |ic_e(\overline{xx'})|$ and $iv_{rst_e} := |rst_e(\overline{x})|$. The addition of $iv_{ic_e} \leq |ic(\overline{xx'})|$ and $iv_{rst_e} \leq |rst(\overline{xx'})|$ to $Psums^{c_e}$ follows directly from these definitions.

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

$$\sum_{c_e \in Cntr} cntr_e \leq |cd(\overline{x_s})| + \sum_{c_e \in Ic} smiv_{ic_e} + \sum_{c_e \in Rst} smiv_{rst_e}$$

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:

$$\begin{aligned}
\sum_{c_e \in Cntr} \Big(\sum_{smiv_k \in cntr_e} psmiv^{c_e}{}_k[1..n] \Big) \leq |cd(\overline{x_1})| - |cd(\overline{x_n})| + \\
+ \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) + \sum_{c_e \in Rst} (psmiv^{c_e}{}_{rst_e}[1..n])
\end{aligned}$$

Note that $|cd(\overline{q_1})| - |cd(\overline{q_n})| \leq |cd(\overline{q_1})|$ 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 $0 \leq |cd(\overline{x_1})| - |cd(\overline{x_1})| = 0$ 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 $n+1$ case to the n case and prove that the additional summands maintain the inequality.

– If $CE?(n) = c_i \in Cntr$, the left hand side of the constraint is:

$$\sum_{c_e \in Cntr} (\sum_{smiv_k \in cntr_e} psmiv^{c_e}{}_k[1..n+1]) = \sum_{c_e \in Cntr} \sum_{smiv_k \in cntr_e} psmiv^{c_e}{}_k[1..n] + \sum_{smiv_k \in cntr_i} iv_{kn}$$

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:

$$|cd(\overline{x_1})| - |cd(\overline{x_{n+1}})| + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n+1]) + \sum_{c_e \in Rst} (psmiv^{c_e}{}_{rst_e}[1..n+1]) = |cd(\overline{x_1})| - |cd(\overline{x_n})| + |cd(\overline{x_n})| - |cd(\overline{x_{n+1}})| + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) + \sum_{c_e \in Rst} (psmiv^{c_e}{}_{rst_e}[1..n])$$

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

$$\sum_{smiv_k \in cntr_i} iv_{kn} \bowtie |cd(\overline{x_n})| - |cd(\overline{x_{n+1}})|$$

According to the definition of $Cntr$, we have:

$$(1) \quad \sum_{smiv_k \in cntr_i} iv_{kn} \leq |l'(\overline{x_n x_{n+1}})| \leq cd(\overline{x_n}) - cd(\overline{x_{n+1}})$$
$$(2) \quad \sum_{smiv_k \in cntr_i} iv_{kn} \leq |l'(\overline{x_n x_{n+1}})| \leq cd(\overline{x_n})$$

The property (2) implies that $cd(\overline{x_n})$ is positive ($cd(\overline{x_n}) = |cd(\overline{x_n})|$).

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

$$\sum_{smiv_k \in cntr_i} iv_{kn} \leq cd(\overline{x_n}) - cd(\overline{x_{n+1}}) = |cd(\overline{x_n})| - |cd(\overline{x_{n+1}})|$$

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

$$\sum_{smiv_k \in cntr_i} iv_{kn} \leq |l'(\overline{x_n x_{n+1}})| \leq cd(\overline{x_n}) \leq |cd(\overline{x_n})| = |cd(\overline{x_n})| - |cd(\overline{x_{n+1}})|$$

– 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:

$$|cd(\overline{x_1})| - |cd(\overline{x_{n+1}})| + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n+1]) + \sum_{c_e \in Rst} (psmiv^{c_e}{}_{rst_e}[1..n+1]) =$$
$$|cd(\overline{x_1})| - |cd(\overline{x_n})| + |cd(\overline{x_n})| - |cd(\overline{x_{n+1}})| + \sum_{c_e \in Ic} (psmiv^{c_e}{}_{ic_e}[1..n]) + \sum_{c_e \in Rst} (psmiv^{c_e}{}_{rst_e}[1..n])$$

Therefore, we have to prove:

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

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

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

- if $cd(\overline{x_{n+1}})$ is positive, $cd(\overline{x_n})$ is also positive and $0 \leq cd(\overline{x_n}) - cd(\overline{x_{n+1}}) =^{(1)} |cd(\overline{x_n})| - |cd(\overline{x_{n+1}})|$.
- if $cd(\overline{x_{n+1}})$ is negative, $|cd(\overline{x_n})| - |cd(\overline{x_{n+1}})| = |cd(\overline{x_n})| \geq 0$ (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:

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

From the definition of Ic, we know:

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

We distinguish cases:

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

– 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:

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

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

Note that the generated constraint does not contain variables from the end of the phase ($\overline{x_f}$). 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 $iv \leq |l(\overline{xx'})| \in Pms^{c_i}$, if we manage to classify all the $c_e \in ph$ into Dc, Ic, and Rst, we generate:

$$\lceil iv \rceil \leq iv_{max} + \sum_{c_e \in Ic} smiv_{ic_e}, \;\; iv_{max} \leq \max_{c_e \in Rst} (\lceil iv \rceil_{rst_e}, iv_{cd}), \;\; iv_{cd} \leq |cd(\overline{x_s})|$$

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

- $iv_j \leq |cd(\overline{x_s})| + \sum_{c_e \in Ic} smiv_{ic_e}$;
- or there is a $c_e \in Rst$ such that: $iv_j \leq \lceil iv \rceil_{rst_e} + \sum_{c_e \in Ic} smiv_{ic_e}$.

If all instances iv_j are smaller or equal than an amount. $\lceil iv \rceil$ (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 $iv_j \leq |l(\overline{x_j x_{j+1}})| \leq |cd(\overline{x_j})|$. We consider the CE evaluations that happen before. Consider the last CE evaluation such that it belongs to Rst $CE?(l) = c_r \in Rst$. 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 $|cd(\overline{x_{k+1}})| \leq |cd(\overline{x_k})|$.
- For each evaluation such that $CE?(k) = c_e \in Ic$, we have $|cd(\overline{x_{k+1}})| \leq |cd(\overline{x_k})| + iv_{ic_e k}$*.

* ($cd(\overline{x_{k+1}}) \leq cd(\overline{x_k}) + ic_e(\overline{x_k x_{k+1}})$ implies $|cd(\overline{x_{k+1}})| \leq |cd(\overline{x_k})| + |ic_e(\overline{x_k x_{k+1}})| = |cd(\overline{x_k})| + iv_{ic_e k}$).

This way, we have:

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

Given that $CE?(l) = c_r \in Rst$, we have that $|cd(\overline{x_{l+1}})| \leq |rst_r(\overline{x_l})| = iv_{rst_r l}$ which by definition is $iv_{rst_r l} \leq \lceil iv \rceil_{rst_r}$. Therefore, we conclude:

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

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

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

The maximum of the different cases for the different $c_e \in Rst$ corresponds to the constraint generated. The proof for $\lfloor iv \rfloor \geq |l(\overline{xx'})| \in Pms^{c_i}$ 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:

$$\sum_{c_e \in CntTri} cntTri_e \bowtie |cd(\overline{x_s})| * iv_{its} + \tfrac{q}{2} iv_{its}^2 - \tfrac{q}{2} iv_{its} \ , \ iv_{its} = \sum_{c_e \in CntTri} smiv_{it}$$

Which is a merged version of the constraints stated in Sec. A.2. Here $q = \max_{c_e \in CntTri}(q_e)$ if $\bowtie$ 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 $cd(\overline{x_n}) \bowtie cd(\overline{x_1}) + q * iv_{its}[1..n]$.*

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 $cd(\overline{x_1}) \bowtie cd(\overline{x_1}) + 0$.
- Inductive case: We assume $cd(\overline{x_n}) \bowtie cd(\overline{x_1}) + q * iv_{its}[1..n]$ and prove it for $n + 1$. We distinguish two cases:
  - If $CE?(n) \in CntTri$, we have:

$$\begin{aligned} cd(\overline{x_{n+1}}) &\bowtie cd(\overline{x_n}) + q \bowtie cd(\overline{x_1}) + q * iv_{its}[1..n] + q \\ &=^{(IH)} cd(\overline{x_1}) + q * (iv_{its}[1..n] + 1) = cd(\overline{x_1}) + q * (iv_{its}[1..n+1]) \end{aligned}$$

  - If $CE?(n) \in NoCnt$, we have:

$$\begin{aligned} cd(\overline{x_{n+1}}) = cd(\overline{x_n}) &=^{(IH)} cd(\overline{x_1}) + q * iv_{its}[1..n] \\ &= cd(\overline{x_1}) + q * iv_{its}[1..n+1] \end{aligned}$$

  According to the definition of $CntTri$, we have:

$$\sum_{c_e \in CntTri} cntTri_e \bowtie \sum_{1 \leq j < f \wedge CE?(j) \in CntTri} cd(\overline{x_j})$$

We prove that the left side of the constraint that we generate is a valid approximation of such constraint:

$$
\begin{aligned}
&\sum_{1\le j<f\wedge CE?(j)\in CntTri} cd(\overline{x_j}) \\
&\bowtie^{(1)} \sum_{1\le j<f\wedge CE?(j)\in CntTri} (cd(\overline{x_1}) + q * iv_{its}[1..j]) \\
&=^{(2)} iv_{its}[1..f] * cd(\overline{x_1}) + \sum_{1\le j<f\wedge CE?(j)\in CntTri} (q * iv_{its}[1..j]) \\
&=^{(3)} iv_{its}[1..f] * cd(\overline{x_1}) + q \sum_{j=0}^{iv_{its}[1..f]} j \\
&=^{(4)} iv_{its} * cd(\overline{x_1}) + \sum_{j=0}^{iv_{its}} j \\
&=^{(5)} cd(\overline{x_1}) * (iv_{its}) + \tfrac{q}{2} * (iv_{its}^2 - iv_{its})
\end{aligned}
$$

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.