Monotone Procedure Summarization via Vector Addition Systems and Inductive Potentials

NIKHIL PIMPALKHARE, Princeton University, USA
ZACHARY KINCAID, Princeton University, USA

This paper presents a technique for summarizing recursive procedures operating on integer variables. The motivation of our work is to create more predictable program analyzers, and in particular to formally guarantee compositionality and monotonicity of procedure summarization. To summarize a procedure, we compute its best abstraction as a vector addition system with resets (VASR) and exactly summarize the executions of this VASR over the context-free language of syntactic paths through the procedure. We then improve this technique by refining the language of paths that the VASR is summarized over by synthesizing linear *potential functions* bounding the number of recursive calls within valid executions of the input program. We implemented our summarization technique in an automated program verification tool; our experimental evaluation demonstrates that our technique computes more precise summaries than existing abstract interpreters and that our tool's verification capabilities are comparable with state-of-the-art software model checkers.

Additional Key Words and Phrases: Invariant Generation, Program Analysis, Formal Methods

1 INTRODUCTION

Program analyzers typically reason about procedures by computing summaries that over-approximate their behavior and using those summaries to interpret procedure calls. Procedure summarization is typically driven by heuristics, which can lead to unpredictable behavior of down-stream analysis tasks. For example, software verifiers built atop heuristic analysis methods can exhibit unintuitive behavior, as seen in Figures 1 and 2. Ultimately, this unpredictability is a symptom of the fact that program analysis techniques generally do not provide any guarantees on their behavior beyond soundness (and in some cases, termination). This raises the question: *what behavioral guarantees are attainable while retaining state-of-the-art precision and performance?*

```
int add(int m, int n) {
   if (n == 0) { return m; }
   if (n > 0) { return add(m + 1, n − 1); }
   if (n < 0) { return add(m − 1, n + 1); }}
```

Fig. 1. SOTA verifiers UAutomizer [Heizmann et al. 2013] and Korn [Ernst 2020] verify $add(m, n) == m + n$ but cannot verify $m1 > m2 \implies add(m1, n) > add(m2, n)$ even though the former property implies the latter.

```
int nodes = 0; int leaves = 0;
void tc1(int n) {
   if (*) { leaves += 1; }
   else { nodes += 1;
      tc1((n − 1) / 2); tc1((n − 1) / 2); }
```

```
int nodes = 0; int leaves = 0;
void tc2(int n) {
   if (n <= 1) { leaves += 1; }
   else { nodes += 1;
      tc2((n − 1) / 2); tc2((n − 1) / 2); }
```

Fig. 2. The above programs differ only at the conditional. UAutomizer and Korn verify $nodes + 1 == leaves \vee nodes == n + 46$ after running `tc1` but cannot verify it holds after running `tc2`, even though the behavior of the latter program is a subset of the former.

This paper describes a summarization algorithm for recursive procedures operating over integers that is *compositional* and *monotone*. A compositional analyzer summarizes a composite program by combining summaries of its components. In Figure 1, a compositional analyzer verifies the latter property for `add` if it verifies the former. If program A's input-output behavior is a subset of program B's, a monotone analyzer will compute at least as precise of a summary for A as it does for B. In Figure 2, a monotone analyzer verifies any property for `tc2` that it does for `tc1`. The primary barrier to monotonicity is that many of the foundational tools that are used to

Authors' addresses: Nikhil Pimpalkhare, Princeton University, Princeton, USA, np6641@princeton.edu; Zachary Kincaid, Princeton University, Princeton, USA, zkincaid@princeton.edu.

compute summaries, in particular widening and Craig interpolation, are non-monotone.

A successful recipe for developing monotone loop analyses has been to compute an abstract model of a loop and to analyze the exact dynamics of that model [Kincaid 2018; Silverman and Kincaid 2019; Zhu and Kincaid 2021a]. In particular, Silverman and Kincaid [2019] computes loop summaries by (1) modelling loops as Rational Vector Addition Systems with Resets (VASR) and (2) leveraging the reachability relation of the VASR as an over-approximation of the transitive closure of the loop. The key factors making this approach monotone are that the computed abstraction is *best*, in the sense that it is at least as precise as any other abstraction in its class, and that one can precisely summarize the reachability relation of a VASR in a loop. However, there is no obvious way to translate a technique for loop summarization into one for recursive procedures.

This paper extends the best abstraction paradigm of monotone loop analyses to compute monotone summaries of recursive procedures. We compute the best VASR abstraction of our input procedure and then compute an *exact* summary of the executions of the VASR along the context-free language of paths through the procedure. This formula serves as an over-approximate summary of the input procedure. To improve the precision of this technique, we present two refinements. First, we extend our method to Lossy VASRs, a strictly more powerful abstract domain. Second, we perform an auxiliary static analysis that synthesizes potential functions bounding the number of procedure calls within any valid program execution as a function of the input state; we then only encode the executions of the VASR on paths meeting these bounds into our summary. The end-to-end summarization procedure of this paper is compositional, in that it computes a summary for each procedure separately, and monotone. Our evaluation of this technique within a software verifier empirically shows that it computes more precise summaries than existing abstract interpreters and that its verification capabilities are comparable with state-of-art model checkers.

Contributions. (1) Algorithm computing the best VASR abstraction of an input program. (Section 5). (2) Polynomially-sized encoding of the reachability relation of a VASR along a context-free language. (Section 4). (3) Extension of above contributions to more expressive domain of Lossy VASRs (Section 6). (4) Monotone technique to synthesize linear potential functions of the input state upper bounding procedure invocations within executions of the input program. (Section 7). (5) A benchmark suite of recursive integer programs. (Section 9).

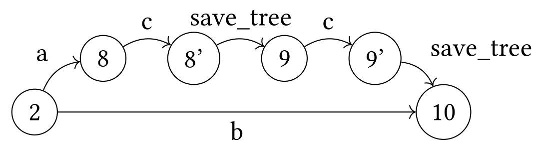
2 OVERVIEW

The objective of our technique is to compute a *transition formula* that over-approximates the dynamics of a procedure. A transition formula F is a logical formula over a set of program variables X and primed copies X' respectively representing program state before and after some computation. For two states $\rho, \rho' \in \mathbb{Q}^X$, we say ρ can transition to ρ' according to F if F holds when each x in X is replaced with $\rho(x)$ and each x' in X' is replaced with $\rho'(x)$. We aim to compute a summary formula F such that if an execution of a procedure with input state ρ terminates with state ρ' then ρ can transition to ρ' according to F.

Figure 3 displays a procedure `save_tree` that traverses a binary tree, saving the value of each internal node to an intermediate buffer and emptying the buffer to disk at any leaf. The variable *mem_ops* counts the number of integers written to disk, *buf* represents the length of the buffer, and *size* represents the size of the binary tree. The source code for `save_tree` is pictured alongside its representation as a *program graph* and *transition assignment*. The nodes of the program graph represent lines of the code and the edges represent execution paths between those lines, with recursive calls labeled by the name of the called functions. The transition assignment $t\!f$ corresponds

```c
int mem_ops, buf;
void save_tree(int size) {
    buf += 1;
    if (size <= 1) {
        mem_ops += buf;
        buf = 0;
    } else {
        save_tree((size - 1) / 2);
        save_tree((size - 1) / 2);
    }
}
```

```c
void main() {
    mem_ops = 0; buf = 0;
    int size = nondet_int();
    assume(size >= 1);
    save_tree(size);
    assert(mem_ops <= size + 1);
}
```



Globals: $\{mem\_ops, buf, param0\}$ Locals: $\{size\}$

$$\mathcal{T}(a) = \begin{pmatrix} size' = param0 \wedge\ size' > 1 \\ \wedge\ param0' = param0 \wedge\ buf' = buf + 1 \\ \wedge\ mem\_ops' = mem\_ops \end{pmatrix}$$

$$\mathcal{T}(b) = \begin{pmatrix} size' = param0 \wedge\ size' \leq 1 \\ \wedge\ buf' = 0 \wedge\ param0' = param0 \\ \wedge\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix}$$

$$\mathcal{T}(c) = \begin{pmatrix} param0' = (size - 1)/2 \wedge buf' = buf \\ \wedge mem\_ops' = mem\_ops \wedge size' = size \end{pmatrix}$$

Fig. 3. On the left, a program in C (top) and verification task (bottom). On the right, its representation as a program graph (top) representing the trajectories (syntactic paths) through the program and a transition assignment (bottom) representing its semantics.

$$f(\rho)(y_1) = \rho(buf)$$
$$f(\rho)(y_2) = \rho(mem\_ops + buf)$$

$$\mathcal{V}(a) \triangleq y_1' = y_1 + 1 \wedge y_2' = y_2 + 1$$
$$\mathcal{V}(b) \triangleq y_1' = 0 \wedge y_2' = y_2 + 1$$
$$\mathcal{V}(c) \triangleq y_1' = y_1 \wedge y_2' = y_2$$

Fig. 4. Best VASR abstraction of transition assignment in Figure 3. If ρ can transition to ρ' according to $\mathcal{T}(s)$ then $f(\rho)$ can transition to $f(\rho')$ according to $\mathcal{V}(s)$ for any s in $\{a, b, c\}$.

each non-call edge with a transition formula representing the semantics of the corresponding execution path. Parameter passing is modeled via the global variable $param0$.

The program graph represents a language of *trajectories* through the procedure `save_tree`, the members of which are paths from the input vertex (2) to the output vertex (10) in which every recursive call is replaced with a trajectory through the called procedure. For example, $acbcacbcb$ is a trajectory through `save_tree`. This language is context-free; the grammar generating this language has one nonterminal `save_tree`, terminals $\{a, b, c\}$, and production rules $\{\texttt{save\_tree} \implies b, \texttt{save\_tree} \implies ac\texttt{save\_tree}c\texttt{save\_tree}\}$.

Silverman and Kincaid [2019] computed over-approximate loop summaries by computing the best Rational Vector Addition System with Resets (VASR) that over-approximates a single transition formula describing the body of a loop and computing the reachability of this VASR over any number of iterations of the loop. We generalize this approach by viewing procedures as context-free languages interpreted with a transition assignment; we compute the best VASR that over-approximates the transition assignment and compute its reachability constrained to sequences in the context-free language of trajectories through the procedure. The following paragraphs elaborate on each step of this technique in turn.

Best VASR abstractions. A labeled Rational Vector Addition System with Resets (VASR)[1] $\mathcal{V}$ over variables Y is a transition assignment in which each transition formula is of the form $\bigwedge_{y \in Y} y' = r_y \cdot y + a_y$ where $r_y \in \{0, 1\}$ and $a_y \in \mathbb{Q}$ for all $y \in Y$. Each variable of a VASR can be thought of as an independent counter; for each counter, a VASR transition either resets the counter to zero or leaves its value unchanged, and then adds some rational constant to it.

Most programs (including `save_tree`) are not VASRs, but we show in Section 5 that every program has a *best* VASR that over-approximates it, which is called its *reflection*. A VASR reflection $\mathcal{V}$ of $\mathcal{tf}$ is depicted in Figure 4. $\mathcal{V}$ is defined over two variables y_1 and y_2, which correspond to the terms buf and $mem\_ops{+}buf$, respectively. The *linear simulation* f captures this correspondence: for each character s, if state ρ can transition to state ρ' according to $\mathcal{tf}(s)$, then state $f(\rho)$ can transition to state $f(\rho')$ according to $\mathcal{V}(s)$. The simulation f ensures that each VASR transition $\mathcal{V}(s)$ is an over-approximation of transition formula $\mathcal{tf}(s)$, and thus by transitivity, that the composition of VASR transitions according to $\mathcal{V}$ along a trajectory is an over-approximation of the composition of transition formulas according to $\mathcal{tf}$ along that trajectory. In this sense, we say $\mathcal{V}$ simulates $\mathcal{tf}$.

The VASR reflection is best because it simulates any VASR that simulates $\mathcal{tf}$, and therefore contains at least much information about the semantics of $\mathcal{tf}$ as any other VASR. The computed VASR being best of its class is the critical property that makes our summarization method monotone.

Our method for computing VASR reflections is an extension of Silverman and Kincaid [2019] to the more general setting of summarizing procedures; we go beyond this work by (1) computing VASR reflections of transition assignments expressed in linear integer real arithmetic (LIRA), not just linear real arithmetic (LRA) and (2) presenting a new coordinate-free theory of VASR abstractions, allowing us to easily extend our best abstraction strategy to extensions of the VASR model.

CFL-reachability for VASR. For a context free grammar G and VASR $\mathcal{V}$, Section 4 describes a method for computing a transition formula F such that state ρ can transition to state ρ' according to F if and only if ρ can transition to ρ' according the composition of VASR transformations of $\mathcal{V}$ along some member of the language of G. In our context, this is used to summarize all executions of a VASR along the trajectories through a procedure. Haase and Halfon [2014] showed that the reachability relation for VASR restricted to sequences in a regular language (equivalently, the reachability of VASR with states) can be defined in LIRA formulas; however, their approach does not generalize to context-free languages. We present a fresh approach which defines the CFL-restricted reachability relation for VASR (equivalently, reachabiliy for VASR with a stack).

The reachability of the VASR $\mathcal{V}$ shown in Figure 4 over the language of trajectories through the program graph shown in Figure 3 can by described by the formula $y_1' = 0 \land \exists k \geq 0. y_2' = y_2 + k + 1$. By applying the variable substitution corresponding to the simulation f shown in Figure 4, we obtain an over-approximate summary of the executions of $\mathcal{tf}$ on the language of paths through the program graph: $buf' = 0 \land \exists k \geq 0. mem\_ops' + buf' = mem\_ops + buf + k + 1$. Note that k symbolically represents the number of child invocations to the `save_tree` procedure.

Extension to Lossy VASRs. A labeled Lossy VASR over variables Y is a transition assignment in which each transition formula is of the form $\bigwedge_{y \in Y} y' \leq r_y \cdot y + a_y$ where $r_y \in \{0, 1\}$ and $a_y \in \mathbb{Q}$ for all $y \in Y$. Section 6 describes how our method for computing best VASR abstractions can be extended to compute best Lossy VASR abstractions. CFL-reachability of Lossy VASR is a trivial extension of CFL-reachability of VASR.

[1] Classically, the states of vector addition systems are vectors of natural numbers. In this paper, states are vectors of rational numbers (essentially equivalent to the $\mathbb{Z}$-VASR model of Haase and Halfon [2014]). The relaxation to rationals both (1) allows VASR to model quantities that may be negative (e.g., signed integers in C) and (2) enables the reachability relation of a VASR to be defined in linear integer/real arithmetic.

$$\begin{aligned} f(\rho)(z_1) &= \rho(buf) \\ f(\rho)(z_2) &= \rho(-buf) \\ f(\rho)(z_3) &= \rho(mem\_ops + buf) \\ f(\rho)(z_4) &= \rho(-mem\_ops - buf) \end{aligned}$$

$$\begin{aligned} \mathcal{LV}(a) &\triangleq \begin{array}{c} z_1' \leq z_1 + 1 \land z_2' \leq z_2 - 1 \land \\ z_3' \leq z_3 + 1 \land z_4' \leq z_4 - 1 \end{array} \\ \mathcal{LV}(b) &\triangleq \begin{array}{c} z_1' \leq 0 \land z_2' \leq 0 \land \\ z_3' \leq z_3 + 1 \land z_4; \leq z_4 - 1 \end{array} \\ \mathcal{LV}(c) &\triangleq \begin{array}{c} z_1' \leq z_1 \land z_2 \leq z_2 \land \\ z_3' \leq z_3 \land z_4' \leq z_4 \end{array} \end{aligned}$$

Fig. 5. Best Lossy VASR abstraction of transition assignment in Figure 3.

A Lossy VASR reflection of $\mathcal{T\!F}$ is depicted in Figure 5. The summary computed by taking the CFL reachability of $\mathcal{LV}$ through the program graph and applying the variable substitution of f is:

$$\exists k \geq 0. \begin{pmatrix} buf' \leq 0 \land -buf' \leq 0 \land \\ mem\_ops' + buf' \leq k + 1 \land -mem\_ops' - buf' \leq -k - 1 \end{pmatrix}$$

Note that this summary is logically equivalent to the summary produced by VASR summarization. The Lossy VASR variables z_1 and z_2 capture the same information as VASR variable y_1 in $\mathcal{V}$; the same holds for z_3, z_4 and y_2. In this way, Lossy VASR summaries are always at least as precise as VASR summaries. Additionally, since in some cases program behavior can be modeled by a Lossy VASR but not by a VASR, Lossy VASRs are a strictly more powerful domain than VASRs.

Constraining call count with potentials. The language of trajectories through the program graph includes all syntactic trajectories, including those that do not represent valid executions of the input program. The above summary is not precise enough to conclude that $mem\_ops' \leq size + 1$ because k can be arbitrarily large. Observe that any call to `save_tree` from input state ρ can produce at most $\max(0, \rho(param0))$ child calls; if our summary bounded k to be below this term, it would be precise enough to prove the desired assertion. We improve the precision of our summary by synthesizing potential functions [Tarjan 1985] bounding the number of times each procedure can be invoked as a function of the initial state and summarizing VASR executions only over the subset of trajectories meeting these bounds. A *potential function* ν bounding calls to p maps each procedure q and valuation ρ to an integer $\nu(q, \rho)$ such that every (terminating) execution of q with initial state ρ may call p no more than $\nu(q, \rho)$ times.

The key point making our refinement technique monotone is that we synthesize *all* potential functions meeting an intra-procedurally checkable *inductiveness* condition and matching our template: $\nu(q, \rho) = \max(0, \theta_q(\rho))$, where all θ_q are linear. Our method generates a set of necessary constraints for valid inductive potentials via an intraprocedural analysis and incorporates a representation of all solutions into our transition formula summary. The over-approximate summary of the program in Figure 3 after adding this refinement is:

$$\exists k \geq 0. \begin{pmatrix} buf' \leq 0 \land -buf' \leq 0 \land \\ mem\_ops' + buf' \leq k + 1 \land -mem\_ops' - buf' \leq -k - 1 \end{pmatrix} \land k \leq \max(0, param0)$$

Observe that this summary is strong enough to prove that $mem\_ops \leq size + 1$, as desired.

3 BACKGROUND

Let $[n]$ denote the set $\{1, \ldots, n\}$. Given a function $f : A \to B$ and a subset of the domain $C \subseteq A$, $f|_C : C \to B$ is the restriction of f to the domain C. Within this paper, we use "vector space" to mean a vector space over the rational numbers. For a finite set of variables X, let $Lin(X) \triangleq \{\sum_{x \in X} \alpha_x x : \alpha_x \in \mathbb{Q}\}$ be the vector space of linear terms over X. Given a valuation $\rho : \mathbb{Q}^X$ and a term t over X, let $\rho(t)$ be the evaluation of t.

A **convex polyhedron** P in vector space V is a set of the form

$$P = \left\{ \sum_{i=1}^{n} \lambda_i v_i + \sum_{j=1}^{m} \alpha_j r_j : \lambda_i \geq 0, \alpha_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1 \right\}$$

where $v_1 \dots v_n, r_1 \dots r_m \in V$. The sets $\{v_1, \dots, v_n\}$ and $\{r_1, \dots, r_m\}$ are the *vertices* and *rays* of P. If the only vertex of P is zero, P is a **convex cone** [Schrijver 1986].

3.1 Transition Systems

A **labeled transition system**[2] over a finite set of variables X and a finite alphabet Σ is a pair $T = \langle \mathbb{Q}^X, \to_T \rangle$ where $\mathbb{Q}^X$ is a state space and $\to_T \subseteq \mathbb{Q}^X \times \Sigma \times \mathbb{Q}^X$ is a labeled transition relation. We use $T|_{\Sigma'}$ to denote the restriction of transition system T to the alphabet $\Sigma' \subseteq \Sigma$. We use the following notation for transition systems:

- For character $s \in \Sigma$, $\rho \xrightarrow{s}_T \rho'$ denotes that $\langle \rho, s, \rho' \rangle$ belongs to transition relation $\to_T$
- For word $s_1 \dots s_n \in \Sigma^*$, $\rho \xrightarrow{s_1 \dots s_n} \rho'$ denotes there exist states $\rho_0 \dots \rho_n$ such that $\rho = \rho_0 \xrightarrow{s_1}_T \rho_1 \xrightarrow{s_2}_T \dots \xrightarrow{s_n}_T \rho_n = \rho'$. The sequence $\rho_0 \dots \rho_n$ is called a **trace** of the word $s_1 \dots s_n$ in T.
- For language $L \subseteq \Sigma^*$, $\rho \xrightarrow{L}_T \rho'$ denotes that $\rho \xrightarrow{w}_T \rho'$ for some $w \in L$

A **linear simulation** between labeled transition systems T over variables X and U over variables Y over the same alphabet Σ is a linear map $f : \mathbb{Q}^X \to \mathbb{Q}^Y$ such that for all $s \in \Sigma$, if $\rho \xrightarrow{s}_T \rho'$ then $f(\rho) \xrightarrow{s}_U f(\rho')$. The image of T under a function $f : \mathbb{Q}^X \to \mathbb{Q}^Y$ is the labeled transition system $image(T, f) = \langle \mathbb{Q}^Y, \to_U \rangle$ where $\sigma \xrightarrow{s}_U \sigma'$ if and only if there is some $\rho \xrightarrow{s}_T \rho'$ with $\sigma = f(\rho)$ and $\sigma' = f(\rho')$. Each linear map $f : \mathbb{Q}^X \to \mathbb{Q}^Y$ corresponds uniquely to a variable substitution $\text{SUB}_f : Y \to Lin(X)$ such that $f(\rho)(y) = \rho(\text{SUB}_f(y))$; SUB_f extends uniquely to a linear map in $Lin(Y) \to Lin(X)$ which is the dual of f. For example, the substitution SUB_f of the linear simulation displayed in Figure 4 sends y_1 to buf and y_2 to $mem\_ops + buf$.

A **transition formula** over a set of variables X is a formula F in linear integer/real arithmetic (LIRA) whose free variables range over variables X and primed copies X'. For two valuations $\rho, \rho' \in \mathbb{Q}^X$, we write $[\rho, \rho'] \models F$ if and only if F holds when every occurrence of $x \in X$ and $x' \in X'$ are replaced with $\rho(x)$ and $\rho'(x)$ respectively. We refer to the set of all transition formulas over X as $TF(X)$. A transition assignment $\mathit{tf} : \Sigma \to TF(X)$ defines a labeled transition system $\langle \mathbb{Q}^X, \to_{\mathit{tf}} \rangle$ where $\rho \xrightarrow{s}_{\mathit{tf}} \rho'$ if and only if $[\rho, \rho'] \models \mathit{tf}(s)$.

A **VASR transition** over variables Y is a transition formula in $TF(Y)$ of the form $\bigwedge_{y \in Y} y' = r_y y + a_y$, where r_y is either 0 or 1 and $a_y \in \mathbb{Q}$ for all y. A Rational Vector Addition System with Resets (**VASR**) over a set of variables Y is a transition assignment $\mathcal{V} : \Sigma \to TF(Y)$ in which $\mathcal{V}(s)$ is a VASR transition for all $s \in \Sigma$. We define $Reset(\mathcal{V}, y)$ to be the set of symbols $s \in \Sigma$ such that $r_y = 0$ in $\mathcal{V}(s)$ and $Offset(\mathcal{V}, s, y)$ to be the rational a_y in $\mathcal{V}(s)$. **Lossy-VASRs** and **Lossy-VASR transitions** are defined in the same way except with "$\leq$" in place of "=".

3.2 Language Formalisms

A **program graph** $M = \langle V, \Sigma, P, E, in, out \rangle$ consists of a finite set of vertices V, a finite alphabet Σ, a finite set of procedure identifiers P (disjoint from Σ), a set of labeled edges $E \subseteq V \times (\Sigma \cup P) \times V$, and two functions $in, out : P \to V$ mapping each procedure to its input and output vertex respectively.

[2]We restrict our attention to transition systems with state spaces which are finite-dimensional linear spaces over the rationals.

A **trajectory** over Σ is a word in Σ^*. A **nested trajectory** over Σ and P is a sequence $\tau_1 \dots \tau_n$ such that each τ_i is either a character $s \in \Sigma$ or a pair $\langle p, \tau \rangle$ such that $p \in P$ and τ is a nested trajectory. We denote the set of all nested trajectories over Σ and P to be $\mathcal{N}(\Sigma, P)$. For program graph M, we define the set of nested trajectories $\mathcal{T}_M(u, v)$ between vertices u and v to be the least set such that:

- if $(u, s, v) \in E$ with $s \in \Sigma$, then $s \in \mathcal{T}_M(u, v)$
- if $(u, p, v) \in E$ with $p \in P$ and $\tau \in \mathcal{T}_M(in(p), out(p))$, then $\langle p, \tau \rangle \in \mathcal{T}_M(u, v)$
- if $\tau_1 \in \mathcal{T}_M(u, w)$ and $\tau_2 \in \mathcal{T}_M(w, v)$, then $\tau_1 \tau_2 \in \mathcal{T}_M(u, v)$

We will use $\mathcal{T}_M(p)$ as shorthand for $\mathcal{T}_M(in(p), out(p))$.

Suppose variables X are partitioned into locals X_L and globals X_G. For transition assignment $\mathit{tf} : \Sigma \to TF(X)$ and nested trajectory $\tau_1 \dots \tau_n \in \mathcal{N}(\Sigma, P)$, we write $\rho_0 \xrightarrow{\tau_1 \dots \tau_n}_{\mathit{tf}} \rho_n$ to denote there exists a trace $\rho_0 \dots \rho_n$ such that:

- *(Local transition)* if $\tau_i = s$ then $\rho_{i-1} \xrightarrow{s}_{\mathit{tf}} \rho_i$
- *(Procedure call)* if $\tau_i = \langle p, \tau \rangle$ then $\rho_{i-1}|_{X_L} = \rho_i|_{X_L}$ and there exists $\bar{\rho}_{i-1}, \bar{\rho}_i$ such that $\bar{\rho}_{i-1} \xrightarrow{\tau}_{\mathit{tf}} \bar{\rho}_i$ and $\bar{\rho}_i|_{X_G} = \rho_i|_{X_G}$ and $\bar{\rho}_{i+1}|_{X_G} = \rho_{i+1}|_{X_G}$

Nested trajectories and their transition relations describe semantics that are representative of the local variable behavior of programming languages with lexical scope. Parameter passing and returns can be modeled by introducing auxiliary global variables $param0, param1, \dots$ and ret.

Define *flat* to be the flattening function mapping a nested trajectory to its corresponding trajectory. Formally, *flat* is the homomorphism that sends $\tau_i = s$ to s and $\tau_i = \langle p, \tau \rangle$ to $\mathit{flat}(\tau)$. The **language of trajectories through a procedure** p is defined as $\mathcal{L}_M(p) = \{\mathit{flat}(\tau) : \tau \in \mathcal{T}_M(p)\}$.

A **context-free grammar** $G = \langle N, \Sigma, R, n_0 \rangle$ consists of a finite set of nonterminals N, a finite alphabet Σ, a set of production rules $R \subseteq N \times (\Sigma \cup N)^*$, and a designated start symbol $n_0 \in N$. We denote production rule $(\alpha, \beta) \in R$ as $\alpha \Rightarrow \beta$. An application of production rule $\alpha \Rightarrow \beta$ replaces a single occurrence of α with β: $w_1 \alpha w_2 \to w_1 \beta w_2$. The language corresponding to a nonterminal $\mathcal{L}_G(n)$ is the set $\{w \in \Sigma^* : n \to^* w\}$. The language of the grammar $\mathcal{L}(G)$ is the language of its start symbol, $\mathcal{L}_G(n_0)$. A grammar is in Chomsky Normal Form if all of its productions rules are of the form $n_1 \Rightarrow s$, $n_1 \Rightarrow n_2 n_3$, or $n_0 \Rightarrow \epsilon$ where $n_1, n_2, n_3 \in N$ and $s \in \Sigma$. There is a quadratic-time procedure to convert any context-free grammar into a grammar in Chomsky Normal Form that recognizes the same language [Chomsky 1959]. Given a program graph M, it is straightforward to construct a grammar $\mathcal{G}(M, p)$ such that $\mathcal{L}_M(p) = \mathcal{L}(\mathcal{G}(M, p))$.

The **Parikh image** [Parikh 1966] of a word $w \in \Sigma^*$ is a function $\pi(w) : \Sigma \to \mathbb{N}$ mapping each symbol $s \in \Sigma$ to the number of occurrences of s in w. The Parikh image of a language L is defined as $\pi(L) \triangleq \{\pi(w) : w \in L\}$. For any context-free grammar $G = \langle N, \Sigma, R, n_0 \rangle$, there is a LIRA formula $Parikh(G)$ that represents the Parikh image of $\mathcal{L}(G)$; its free variables are $\{c_s : s \in \Sigma\}$ and $Parikh(G)[c_s \mapsto m(s)]$ holds if and only if m is the Parikh image of some word $w \in \mathcal{L}(G)$. There is a polynomial-time procedure (quadratic in the number of production rules) to compute $Parikh(G)$ from any context-free grammar [Verma et al. 2005].

4 CONTEXT-FREE REACHABILITY OF VASRS

The central pillar of our summarization procedure is a method to compute (in polynomial time) a transition formula that precisely encodes the reachability relation of a VASR over a context-free language. We leverage this result to compute a logical summary of the executions of a VASR abstraction of our input program over the syntactic paths through our program graph. Given a VASR $\mathcal{V}$ over variables Y and alphabet Σ and a context-free grammar G with terminal alphabet Σ, our goal is to compute a transition formula $Reach(\mathcal{V}, G) \in TF(Y)$ such that $[\rho, \rho'] \models Reach(\mathcal{V}, G)$ if and only if $\rho \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}} \rho'$. The following example lends intuition to our approach.

Example 4.1. For the VASR $\mathcal{V}$ in Figure 4, consider the task of computing a formula F such that $[\rho, \rho'] \models F$ if and only if $\rho \xrightarrow{ababa}_{\mathcal{V}} \rho'$. We can consider each variable independently. For the second variable of the VASR, y_2, the composition of $\mathcal{V}(a)$ and $\mathcal{V}(b)$ along *ababa* can be computed from the character count of a and b within the trajectory, as all VASR transitions increment y_2 and therefore commute. Since there are 3 occurrences of a and 2 occurrences of b, $y_2' = y_2 + 3(1) + 2(1)$.

The transitions along *ababa* do not commute with respect to y_1 due to the reset incurred by $\mathcal{V}(b)$, so we cannot compute their composition from the Parikh image of *ababa*. For example, *aaabb* has the same Parikh image as *ababa* but the former resets y_1 to 0 and the latter resets y_1 to 1. Haase and Halfon [2014] observed that it is sufficient to identify the final reset of y_2 from left to right and the Parikh image of the sub-word after it; the final reset nullifies the effects of the transitions before it and all transitions after increment the variable and therefore commute.

To formalize what we wish to compute, observe that any trajectory $w \in \{a, b, c\}^*$ can be decomposed as $w = w_1 w_2 w_3$ where $w_3 \in \{a, c\}^*$, w_2 is either ϵ or b, and w_1 is in $\{a, c\}^*$ if w_2 is ϵ and is in $\{a, b, c\}^*$ otherwise. Intuitively, w_2 identifies the final reset of y_2 from left to right. The transition relation $\xrightarrow{w}_{\mathcal{V}}$ is uniquely determined by the Parikh images of w_1, w_2 and w_3. ⌟

As seen in Example 4.1, our goal is to compute a variation of the Parikh image of the language of G which identifies the final time each variable is reset from left to right. Haase and Halfon [2014] introduced such a variation, the *generalized Parikh image*, which computes the final occurrence of each character in a monitored alphabet; they used it to compute reachability of VASR with states, or regular-reachability in our terminology. Their approach is based on an explicit encoding over the states of the automaton generating a regular language, and does not generalize to context-free languages because there is no finite-state automaton recognizing context-free languages.

Our approach computes abstract trajectories, a more general abstraction of context free languages which identifies an arbitrary number of symbols in each word and captures the Parikh images of the subwords in between. Any particular trajectory has many abstract trajectories that represent it, but at least one identifies the final reset of each variable. We conjoin additional formulas symbolically ensuring that the abstract trajectories we compute over a context-free language capture enough information to compute the corresponding VASR transition. This division of tasks is our key insight that we leverage to compute the CFL-reachability of VASR.

Definition 4.2. A d-marked **abstract trajectory** is a function $n : (\Sigma \times [2d+1]) \to \mathbb{N}$ such that for all even i we have $\sum_{s \in \Sigma} n(s, i) \leq 1$.

For a trajectory $w \in \Sigma^*$ and d-marked abstract trajectory n, write $w \Vdash n$ if there exists a decomposition $w = w_1 \dots w_{2d+1}$ such that $n(s, i) = \pi(w_i)(s)$ for all i and s. For any even i, there is at most one nonzero $n(s, i)$ so we can determine all even-indexed words of the decomposition from n; additionally, n captures the Parikh image of all odd-indexed words. Then, a d-marked abstract trajectory n such that $w \Vdash n$ identifies up to d characters in w in order, and also contains the Parikh images of the subwords between the identified characters.

We say that an abstract trajectory is **well-formed** according to a VASR if its identified characters mark the final reset of each variable. Formally, a $|Y|$-marked abstract trajectory is well-formed with respect to VASR $\mathcal{V}$ over variables Y if for all $y \in Y$ and all odd i:

$$\begin{pmatrix}\text{there exists } s \in Reset(\mathcal{V}, y) \\ \text{with } n(s, i) > 0\end{pmatrix} \implies \begin{pmatrix}\text{there exists even } j > i,\ s' \in Reset(\mathcal{V}, y) \\ \text{with } n(s', j) > 0\end{pmatrix}$$

We show how to compute a formula $Transition(\mathcal{V})$ corresponding a well-formed abstract trajectory to its associated VASR transformation, a formula $AT(\mathcal{V}, G)$ that defines the set of abstract trajectories of trajectories in $\mathcal{L}(G)$, and a formula $WF(\mathcal{V})$ that constrains abstract trajectories to be well-formed. These formulae are conjoined to form $Reach(\mathcal{V}, G)$.

4.1 Transitions of Abstract Trajectories

This subsection defines the formula $Transition(\mathcal{V})$ for a VASR $\mathcal{V}$ over variables Y over alphabet Σ. The free variables of this formula are rational variables Y and Y' representing the pre and post states of the VASR and integer variables $c_{s,k}$ for all $s \in \Sigma$ and $k \in [2|Y|+1]$ representing a $|Y|$-marked abstract trajectory. Its behavior fulfills the following theorem.

Theorem 4.3. *Let $\mathcal{V}$ be a VASR over variables Y, w be a trajectory over Σ, and n be a $|Y|$-marked abstract trajectory well-formed according to $\mathcal{V}$ such that $w \Vdash n$. For all states ρ, ρ':*

$$[\rho, \rho'] \models Transition(\mathcal{V})[c_{s,i} \mapsto n(s,i)] \iff \rho \xrightarrow{w}_{\mathcal{V}} \rho'$$

We define the following helper formulae; $FR(\mathcal{V}, y, j)$ holds if the final reset of y occurs at index j and $After(y, j)$ is the sum of the character counts after index j weighted by the offsets of y.

$$FR(\mathcal{V}, y, j) \triangleq \left(\bigvee_{s \in Reset(\mathcal{V}, y)} c_{s,j} > 0 \right) \wedge \left(\bigwedge_{\substack{s \in Reset(\mathcal{V}, y) \\ j < k \leq 2|Y|+1}} c_{s,k} = 0 \right) \quad After(y, j) \triangleq \sum_{\substack{s \in \Sigma \\ k \geq j}} \mathit{Offset}(\mathcal{V}, s, y) \cdot c_{s,k}$$

$Transition(\mathcal{V})$ is defined as follows:

$$Transition(\mathcal{V}) \triangleq \bigwedge_{y \in Y} \left(\begin{array}{c} \bigvee_{j=1}^{|Y|} (FR(\mathcal{V}, y, 2j) \wedge y' = After(y, 2j)) \\ \vee \left(\bigwedge_{k=1}^{2|Y|+1} \bigwedge_{s \in Reset(\mathcal{V}, y)} c_{s,k} = 0 \wedge y' = y + After(y, 1) \right) \end{array} \right)$$

4.2 Abstract Trajectories of CFLs

The aim of this section is to compute, given a context-free grammar $G\langle N, \Sigma, R, s_0 \rangle$ and a VASR $\mathcal{V}$ over Y, a formula $AT(\mathcal{V}, G)$ that represents the set of $|Y|$-marked abstract trajectories n such that $w \Vdash n$ for some trajectory w in $\mathcal{L}(G)$. This formula has free variables $c_{s,i}$ for $s \in \Sigma$ and $i \in [2|Y|+1]$. It meets the condition that $AT(\mathcal{V}, G)[c_{s,i} \mapsto n(s,i)]$ holds if and only if there exists some $w \in \mathcal{L}(G)$ such that $w \Vdash n$. This section assumes G is in Chomsky Normal Form.

Consider the following regular language, in which each $\Sigma_i \triangleq \{\langle s, i \rangle : s \in \Sigma\}$ is a copy of Σ:

$$O \triangleq \Sigma_1^*(\Sigma_2 + \epsilon)\Sigma_3^* \ldots \Sigma_{2|Y|-1}^*(\Sigma_{2|Y|} + \epsilon)\Sigma_{2|Y|+1}^*$$

Observe that the Parikh image of O is equal to the set of all abstract trajectories over Σ. Defining $h : (\Sigma \times [2|Y|+1])^* \to \Sigma^*$ to be the homomorphism that maps $\langle a, i \rangle \mapsto a$ for all i, one can additionally observe that $n \in \pi(h^{-1}(\mathcal{L}(G)) \cap O)$ if and only if there exists some trajectory $w \in \mathcal{L}(G)$ such that $w \Vdash n$. Since context-free languages are closed under inverse homomorphism and intersection with regular languages, $h^{-1}(\mathcal{L}(G)) \cap O$ is context-free. We may construct a grammar $\mathcal{I}(G, Y) \triangleq \langle N_{[]}, \Sigma \times [2|Y|+1], R_{[]}, s_{[1,2|Y|+1]} \rangle$ that recognizes the language $h^{-1}(\mathcal{L}(G)) \cap O$ as follows; its Parikh image formula $Parikh(I(G, Y))$ will be $AT(\mathcal{V}, G)$.

- The non-terminal symbols are defined to be

$$N_{[]} \triangleq \{n_{[2i+1:2j+1]} : n \in N, 0 \leq i \leq j \leq |Y|\}$$

The intention of the grammar design is that the set of words derivable from $n_{[i:j]}$ is $\mathcal{L}_{I(G,Y)}(n_{[i:j]}) = h^{-1}(\mathcal{L}_G(n)) \cap (\Sigma_i^*(\Sigma_{i+1} + \epsilon) \ldots (\Sigma_{j-1} + \epsilon)\Sigma_j^*)$.

- The productions are defined to be

$$R_{[]} \triangleq \begin{array}{l} \left\{A_{[2i+1:2j+1]} \Rightarrow B_{[2i+1:2k+1]}C_{[2k+1:2j+1]} : A \Rightarrow BC \in R, 0 \le i \le k \le j \le |Y|\right\} \\ \cup\left\{A_{[2i+1:2j+1]} \Rightarrow \langle a, k\rangle : A \Rightarrow a \in R, 2i+1 \le k \le 2j+1, 0 \le i \le j \le |Y|\right\} \\ \cup\left\{s_{[1:2d+1]} \Rightarrow \epsilon : s \Rightarrow \epsilon \in R\right\} \end{array}$$

The design of the production rules maintains the invariant throughout the derivation of any word that for any even k, there is at most one $n_{[i:j]}$ capable of producing a terminal symbol in Σ_k. This ensures that the output of the grammar is in O; all derived words are additionally in $h^{-1}(\mathcal{L}_G(n))$ because all production rules are structurally identical to those of G.

Theorem 4.4. *For any grammar $G = (N, \Sigma, R, s_0)$ (in Chomsky Normal Form), we have*

$$\mathcal{L}(\mathcal{I}(G, Y)) = h^{-1}(\mathcal{L}(G)) \cap O$$

Moreover, observe that $\mathcal{I}(G, Y)$ has $O(|Y||\Sigma|)$ terminals, $O(|Y|^2|N|)$ nonterminals, $O(|Y|^3|R|)$ production rules, and can be constructed in polynomial time.

Theorem 4.5. *Let G be a context-free grammar and $\mathcal{V}$ be a VASR over Y. Define $AT(\mathcal{V}, G) \triangleq Parikh(\mathcal{I}(G, Y))$. Then for all $|Y|$-marked abstract trajectories n:*

$$AT(\mathcal{V}, G)[c_{s,i} \mapsto n(s, i)] \iff w \Vdash n \text{ for some } w \in \mathcal{L}(G)$$

4.3 Well-Formedness and Reachability Formula

Finally, we are ready to use the formulas described in the previous subsections to produce $Reach(\mathcal{V}, G)$, a formula encoding the relation $\xrightarrow{\mathcal{L}(G)}_{\mathcal{V}}$. $Transition(\mathcal{V})$ describes the composition of VASR transitions associated with a well-formed abstract trajectory and $AT(\mathcal{V}, G)$ defines the set of all abstract trajectories such that $w \Vdash n$ for some $w \in \mathcal{L}(G)$. To bridge the gap between these formulas, we define the following formula $WF(\mathcal{V})$ to ensure the abstract trajectory is well-formed.

$$WF(\mathcal{V}) \triangleq \bigwedge_{y \in Y} \left(\bigvee_{j=1}^{|Y|} FR(\mathcal{V}, y, 2j) \vee \bigwedge_{\substack{s \in Reset(\mathcal{V}, i) \\ k \in [2|Y|+1]}} c_{s,k} = 0 \right)$$

With $C \triangleq \left\{c_{s,i} : s \in \Sigma, i \in [2|Y|+1]\right\}$, define:

$$Reach(\mathcal{V}, G) \triangleq \exists C. AT(\mathcal{V}, G) \wedge WF(\mathcal{V}) \wedge Transition(\mathcal{V})$$

Theorem 4.6. *There is a polynomial-time procedure which, given a VASR $\mathcal{V}$ over alphabet Σ and a grammar G over the same alphabet, computes a formula $Reach(\mathcal{V}, G)$ such that:*

$$[\rho, \rho'] \models Reach(\mathcal{V}, G) \iff \rho \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}} \rho'$$

5 BEST VASR ABSTRACTIONS OF TRANSITION FORMULA SYSTEMS

This section describes a procedure for computing the best VASR abstraction $\langle f, \mathcal{V}\rangle$ of a transition assignment tf using a divide-and-conquer approach. A VASR abstraction of tf is a pair composed of a VASR $\mathcal{V}$ and a linear simulation f from tf to $\mathcal{V}$. Using Section 4, we can over-approximate the context-free reachability of tf from a VASR abstraction via the following:

$$\left(\rho \xrightarrow{\mathcal{L}(G)}_{tf} \rho'\right) \implies \left(f(\rho) \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}} f(\rho')\right) \iff \begin{pmatrix} [f(\rho), f(\rho')] \models \\ Reach(\mathcal{V}, G) \end{pmatrix} \iff \begin{pmatrix} [\rho, \rho'] \models \\ Reach(\mathcal{V}, G)[y \mapsto \mathrm{SUB}_f(y)] \end{pmatrix}$$

A VASR abstraction $\langle f, \mathcal{V}\rangle$ is *best* if for any other VASR abstraction $\langle f', \mathcal{V}'\rangle$ there is a linear simulation f^* from $\mathcal{V}$ to $\mathcal{V}'$ such that $f^* \circ f = f'$. We refer to a best VASR abstraction as a *VASR* *reflection*. The over-approximation of the context-free reachability of tf induced by a VASR reflection is at least as precise as that of any other VASR abstraction:

$$\left(\substack{[\rho,\rho'] \models \\ Reach(\mathcal{V},G)[y \mapsto \mathrm{SUB}_f(y)]}\right) \iff \left(f(\rho) \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}} f(\rho')\right) \implies \left(f^*(f(\rho)) \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}'} f^*(f(\rho))\right) \iff \left(\substack{[\rho,\rho'] \models \\ Reach(\mathcal{V}',G)[y \mapsto \mathrm{SUB}_{f'}(y)]}\right)$$

First, we show how to compute a VASR reflection of tf in the special case that Σ is a singleton—in other words, we show that every transition formula has a best abstraction as a VASR transition. Second, we show how to combine reflections across disjoint alphabets—that is, if Σ_1, Σ_2 is a partition of Σ, we may calculate a VASR reflection of $tf : \Sigma \to TF(X)$ from the VASR reflections of $tf|_{\Sigma_1}$ and $tf|_{\Sigma_2}$. By combining these two results, we obtain a procedure for computing a VASR reflection of transition formula systems over finite alphabets.

We restrict our attention to VASR abstractions which operate over global variables of the input program, as we wish to summarize the effect of our programs on global variables; in other words, we require the simulation f to be in $\mathbb{Q}^{X_G} \to \mathbb{Q}^Y$ where X_G is the global subset of the variables X of tf and Y are the variables of $\mathcal{V}$.

5.1 Best VASR Abstractions of Transition Formulas

This subsection describes how to compute the VASR reflection of $tf|_{\{s\}}$. Let X_G denote the global variables of X. Define the spaces of *reset terms* and *incremented terms* implied by a transition formula $F \in TF(X)$ to be:

$$Res(F) \triangleq \{\langle t, a\rangle \in Lin(X_G) \times \mathbb{Q} : F \models t' = a\}$$
$$Add(F) \triangleq \{\langle t, b\rangle \in Lin(X_G) \times \mathbb{Q} : F \models t' = t + b\}$$

$Res(F)$ and $Add(F)$ are linear spaces and can be computed via Reps et al. [2004]. Let $\{\langle t_1, a_1\rangle, \dots, \langle t_n, a_n\rangle\}$ and $\{\langle \hat{t}_1, b_1\rangle, \dots, \langle \hat{t}_m, b_m\rangle\}$ be bases of $Res(tf(s))$ and $Add(tf(s))$ respectively. Then, the VASR reflection $\langle f_s, \mathcal{V}_s\rangle$ of $tf|_{\{s\}}$ can be defined as the following:

$$\mathcal{V}_s(s) \triangleq \left(\bigwedge_{i=1}^{n} y_i' = a_i\right) \wedge \left(\bigwedge_{i=1}^{m} z_i' = z_i + b_i\right)$$
$$\mathrm{SUB}_{f_s}(y_i) = t_i \qquad \mathrm{SUB}_{f_s}(z_i) = \hat{t}_i$$

Lemma 5.1. *For $tf : \Sigma \to TF(X)$ and any $s \in \Sigma$, $\langle f_s, \mathcal{V}_s\rangle$ is a VASR reflection of $tf|_{\{s\}}$.*

Example 5.2. Consider $tf(b)$ from Figure 3. The bases of $Res(tf(b))$ and $Add(tf(b))$ are respectively $\langle buf, 0\rangle$ and $\langle mem\_ops + buf, 1\rangle, \langle param0, 0\rangle$. Then, the VASR reflection of $tf|_{\{b\}}$ is:

$$f_b(\rho)(y_1) = \rho(buf)$$
$$f_b(\rho)(z_1) = \rho(mem\_ops + buf) \qquad \mathcal{V}_b(b) \triangleq y_1' = 0 \wedge z_1' = z_1 + 1 \wedge z_2' = z_2$$
$$f_b(\rho)(z_2) = \rho(param0)$$

⌟

5.2 Combining Best VASR Abstractions over Disjoint Alphabets

For a bipartition Σ_1, Σ_2 of an alphabet Σ, this subsection shows how to combine VASR reflections of $tf|_{\Sigma_1}$ and $tf|_{\Sigma_2}$ into a VASR reflection of $tf : \Sigma \to TF(X)$. Our technique is an adaptation of [Silverman and Kincaid 2019, Algorithm 2] to the setting of labeled transition systems; however, we prove stronger guarantees about the algorithm in the labeled setting (we can compute best abstractions of LIRA transition assignments, not just LRA), and we present a new coordinate-free theory of VASR abstractions, revealing insight about the abstract structure of VASRs without relying upon their matrix representations. This theory provides a foundation for computing reflections of extensions of the VASR model; we discuss one extension (Lossy-VASRs) in Section 6.

The following example refers to the assignment $\mathcal{A}$ of Figure 3 and motivates our approach.

Example 5.3. Consider VASR reflections $\langle f_a, \mathcal{V}_a \rangle$ of $\mathcal{A}|_{\{a\}}$, $\langle f_b, \mathcal{V}_b \rangle$ of $\mathcal{A}|_{\{b\}}$, and $\langle f, \mathcal{V} \rangle$ of $\mathcal{A}|_{\{a,b\}}$:

$$\begin{aligned} f_a(\rho)(y_1) &= \rho(mem\_ops) \\ f_a(\rho)(y_2) &= \rho(buf) \\ f_a(\rho)(y_3) &= \rho(param0) \end{aligned} \qquad \mathcal{V}_a(a) \triangleq y_1' = y_1 \land y_2' = y_2 + 1 \land y_3' = y_3$$

$$\begin{aligned} f_b(\rho)(y_4) &= \rho(buf) \\ f_b(\rho)(y_5) &= \rho(mem\_ops + buf) \\ f_a(\rho)(y_6) &= \rho(param0) \end{aligned} \qquad \mathcal{V}_b(b) \triangleq y_4' = 0 \land y_5' = y_5 + 1 \land y_6' = y_6$$

$$\begin{aligned} f(\rho)(z_1) &= \rho(buf) \\ f(\rho)(z_2) &= \rho(mem\_ops + buf) \\ f(\rho)(z_3) &= \rho(param0) \end{aligned} \qquad \begin{aligned} \mathcal{V}(a) &\triangleq z_1' = z_1 + 1 \land z_2' = z_2 + 1 \land z_3' = z_3 \\ \mathcal{V}(b) &\triangleq z_1' = 0 \land z_2' = z_2 + 1 \land z_3' = z_3 \end{aligned}$$

Since $\langle f_a, \mathcal{V}_a \rangle$ and $\langle f_b, \mathcal{V}_b \rangle$ are reflections, there exist simulations g_a from $\mathcal{V}_a$ to $\mathcal{V}|_a$ and g_b from $\mathcal{V}_b$ to $\mathcal{V}|_b$ such that $g_a \circ f_a = f = g_b \circ f_b$. These are:

$$\begin{aligned} g_a(\rho)(z_1) &= \rho(y_2) \\ g_a(\rho)(z_2) &= \rho(y_1 + y_2) \\ g_a(\rho)(z_3) &= \rho(y_3) \end{aligned} \qquad \begin{aligned} g_b(\rho)(z_1) &= \rho(y_4) \\ g_b(\rho)(z_2) &= \rho(y_5) \\ g_b(\rho)(z_3) &= \rho(y_6) \end{aligned}$$

⌟

In the above example, observe that $\langle f, \mathcal{V} \rangle$ is fully determined by the simulations g_a, g_b and the VASRs $\mathcal{V}_a, \mathcal{V}_b$: $\mathcal{V}|_a = image(\mathcal{V}_a, g_a)$, $\mathcal{V}|_b = image(\mathcal{V}_b, g_b)$, and $f = g_a \circ f_a = g_b \circ f_b$. Thus, our strategy is to find a "best" solution to the equation $g_a \circ f_a = g_b \circ f_b$ subject to the constraint that the image of $\mathcal{V}_a$ under g_a and $\mathcal{V}_b$ under g_b are VASRs. We first investigate the conditions on f under which $image(\mathcal{V}, f)$ is a VASR: we associate with any VASR a *separated space*, a linear space with a canonical decomposition as a direct sum, and show that any simulation between VASRs must be *coherent* in the sense that it preserves the direct sum decomposition. We then show how to compute the "best" coherent solution to the equation $g_a \circ f_a = g_b \circ f_b$.

We say that a VASR transition $F \triangleq \bigwedge_{y \in Y} y' = r_y \cdot y + a_y$ resets a state ρ if $[\rho, \rho'] \models F$ when $\rho'(y) = a_y$ for all y. Likewise, F increments ρ if $[\rho, \rho'] \models F$ when $\rho'(y) = \rho(y) + a_y$ for all y. For a VASR transition over Y, the set of states that are reset and the set of states that are incremented are each linear spaces whose direct sum is $\mathbb{Q}^Y$. A **coherence class** of $\mathcal{V}$ is a linear subspace of $\mathbb{Q}^Y$ of the form $\bigcap_{s \in \Sigma} R_s$ where for each s, R_s is either the space of states reset or the space of states incremented by $\mathcal{V}(s)$. Any two coherence classes only intersect at the zero state, and the direct sum of all coherence classes is $\mathbb{Q}^Y$. A necessary condition for any simulation f from VASR $\mathcal{V}$ to VASR $\mathcal{V}'$ is that if ρ is reset by $\mathcal{V}(s)$, then $f(\rho)$ must be reset by $\mathcal{V}'(s)$; if ρ is incremented by $\mathcal{V}(s)$, then $f(\rho)$ must be incremented by $\mathcal{V}'(s)$. Then, for all coherence classes C' of $\mathcal{V}'$, the set $\{\rho : f(\rho) \in C'\}$ must be contained in a single coherence class C of $\mathcal{V}$. We introduce the following definitions to formalize the abstract structure of VASRs and the linear simulations between them.

A **separated space** S is a pair $\langle V_S, D_S \rangle$ where V_S is a vector space and $D_S = \{C_1, \ldots, C_n\}$ is a finite set of nonzero disjoint subspaces of V_S such that each $v \in V_S$ can be uniquely written as $v = \sum_{i=1}^{n} v_i$ where $v_i \in C_i$. We call v_i the **orthogonal projection** of v onto C_i. For each $C \in D_S$, define $proj_C : V_S \to C$ to be the linear map sending each element of V_S to its orthogonal projection onto C. Define the separated space of a VASR $\mathcal{V}$ over Y to be $S(\mathcal{V}) = \left\langle \mathbb{Q}^Y, D_{S(\mathcal{V})} \right\rangle$ where $D_{S(\mathcal{V})} = \{C_1, \ldots, C_n\}$ are the coherence classes of $\mathcal{V}$.

A **coherent linear map** from separated spaces S to S' is a pair $\langle f, w_f \rangle$ where $f : V_S \to V_{S'}$ is a linear map and $w_f : D_{S'} \to D_S$ is a *witness function* such that for all $C \in D_S$ and $C' \in D_{S'}$, if $C \neq w_f(C')$ then $proj_{C'}(f(proj_C(v))) = 0$ for all $v \in V_S$. Put into words, a coherent linear map f maintains that each orthogonal projection of $f(v)$ is only dependent on one of the orthogonal projections of v; in other terms, $f(v) = \sum_{C \in D'_S} proj_C(f(proj_{w_f(C)}(v)))$. Composition of coherent linear maps can be understood as $\langle f, w_f \rangle \circ \langle g, w_g \rangle \triangleq \langle f \circ g, w_g \circ w_f \rangle$.

Theorem 5.4. *Let $\mathcal{V}$ and $\mathcal{V}^*$ be VASRs. If f is a linear simulation from $\mathcal{V}$ to $\mathcal{V}^*$, then there exists a function $w_f : D_{S(\mathcal{V}^*)} \to D_{S(\mathcal{V})}$ such that $\langle f, w_f \rangle$ is a coherent linear map from $S(\mathcal{V})$ to $S(\mathcal{V}^*)$.*

With this theorem in hand, we can formalize our approach to combining VASR reflections over disjoint alphabets. Let $\tilde{w}$ refer to the dummy witness function sending every input to $\mathbb{Q}^X$ and let $\tilde{S} \triangleq \langle \mathbb{Q}^X, \{\mathbb{Q}^X\} \rangle$ be the separated space of transition assignment tf over variables X. Given two VASR reflections $\langle f_{\Sigma_1}, \mathcal{V}_{\Sigma_1} \rangle$ and $\langle f_{\Sigma_2}, \mathcal{V}_{\Sigma_2} \rangle$ of $tf|_{\Sigma_1}$ and $tf|_{\Sigma_2}$, one can observe that $\langle f_{\Sigma_1}, \tilde{w} \rangle$ is a coherent linear map from $\tilde{S}$ to $S_{\mathcal{V}_{\Sigma_1}}$ and $\langle f_{\Sigma_2}, \tilde{w} \rangle$ is a coherent linear map from $\tilde{S}$ to $S_{\mathcal{V}_{\Sigma_2}}$. We aim to compute a separated space $S_{\mathcal{V}}$ and coherent linear maps $\langle g_{\Sigma_1}, w_{\Sigma_1} \rangle$ from $S_{\mathcal{V}_{\Sigma_1}}$ to $S_{\mathcal{V}}$ and $\langle g_{\Sigma_2}, w_{\Sigma_2} \rangle$ from $S_{\mathcal{V}_{\Sigma_2}}$ to $S_{\mathcal{V}}$ such that:

- $\langle g_{\Sigma_1}, w_{\Sigma_1} \rangle \circ \langle f_{\Sigma_1}, \tilde{w} \rangle = \langle g_{\Sigma_2}, w_{\Sigma_2} \rangle \circ \langle f_{\Sigma_2}, \tilde{w} \rangle$
- for any other separated space $S_{\mathcal{V}'}$ and coherent linear maps $\left\langle g'_{\Sigma_1}, w'_{\Sigma_1} \right\rangle$ from $S_{\mathcal{V}_{\Sigma_1}}$ to $S_{\mathcal{V}'}$ and $\left\langle g'_{\Sigma_2}, w'_{\Sigma_2} \right\rangle$ from $S_{\mathcal{V}_{\Sigma_1}}$ to $S_{\mathcal{V}'}$ such that $\left\langle g'_{\Sigma_1}, w'_{\Sigma_1} \right\rangle \circ \langle f_{\Sigma_1}, \tilde{w} \rangle = \left\langle g'_{\Sigma_2}, w'_{\Sigma_2} \right\rangle \circ \langle f_{\Sigma_2}, \tilde{w} \rangle$, there exists a coherent linear map $\langle u, w_u \rangle$ from $S_{\mathcal{V}}$ to $S_{\mathcal{V}'}$ such that $\langle u, w_u \rangle \circ \langle g_{\Sigma_1}, w_{\Sigma_1} \rangle = \left\langle g'_{\Sigma_1}, w'_{\Sigma_1} \right\rangle$ and $\langle u, w_u \rangle \circ \langle g_{\Sigma_2}, w_{\Sigma_2} \rangle = \left\langle g'_{\Sigma_1}, w'_{\Sigma_1} \right\rangle$

The first condition ensures that the linear simulation $g_{\Sigma_1} \circ f_{\Sigma_1}$ from $tf|_{\Sigma_1}$ to $image(\mathcal{V}_{\Sigma_1}, g_{\Sigma_1})$ is equal to the linear simulation $g_{\Sigma_2} \circ f_{\Sigma_2}$ from $tf|_{\Sigma_2}$ to $image(\mathcal{V}_{\Sigma_2}, g_{\Sigma_2})$ and thus that $\langle g_{\Sigma_1} \circ f_{\Sigma_1}, image(\mathcal{V}_{\Sigma_1}, g_{\Sigma_1}) \uplus image(\mathcal{V}_{\Sigma_2}, g_{\Sigma_2}) \rangle$ is a VASR abstraction of tf. The second condition ensures that this abstraction is best; consider any other abstraction $\langle f', \mathcal{V}' \rangle$ of tf. Since $\langle f_{\Sigma_1}, \mathcal{V}_{\Sigma_1} \rangle$ and $\langle f_{\Sigma_2}, \mathcal{V}_{\Sigma_2} \rangle$ are reflections, there exist simulations g'_{Σ_1} from $\mathcal{V}_{\Sigma_1}$ to $\mathcal{V}'|_{\Sigma_1}$ and g'_{Σ_2} from $\mathcal{V}_{\Sigma_2}$ to $\mathcal{V}'|_{\Sigma_2}$. By Theorem 5.4, there are coherent linear maps $\left\langle g'_{\Sigma_1}, w'_{\Sigma_1} \right\rangle$ from $S_{\mathcal{V}_{\Sigma_1}}$ to $S_{\mathcal{V}'|_{\Sigma_1}}$ and $\left\langle g'_{\Sigma_2}, w'_{\Sigma_2} \right\rangle$ from $S_{\mathcal{V}_{\Sigma_2}}$ to $S_{\mathcal{V}'|_{\Sigma_2}}$. Let w_1 and w_2 be the functions sending each coherence class of $\mathcal{V}'$ respectively to the coherence class of $\mathcal{V}'|_{\Sigma_1}$ or of $\mathcal{V}'|_{\Sigma_2}$ which contains it. There are coherent linear maps $\left\langle g'_{\Sigma_1}, w'_{\Sigma_1} \circ w_1 \right\rangle$ and $\left\langle g'_{\Sigma_2}, w'_{\Sigma_2} \circ w_2 \right\rangle$ from respectively $S_{\mathcal{V}_{\Sigma_1}}$ and $S_{\mathcal{V}_{\Sigma_2}}$ to $S_{\mathcal{V}'}$. If the second condition above holds, there exists a coherent linear map $\langle u, w_u \rangle$ from $S_{\mathcal{V}}$ to $S_{\mathcal{V}'}$ such that $u \circ g_{\Sigma_1} = g'_{\Sigma_1}$ and $u \circ g_{\Sigma_2} = g'_{\Sigma_2}$. Since VASRs are deterministic, we can conclude that u is a simulation from $image(\mathcal{V}_{\Sigma_1}, g_{\Sigma_1}) \cup image(\mathcal{V}_{\Sigma_2}, g_{\Sigma_2})$ to $\mathcal{V}'$.

From a category theoretic view, these conditions correspond to the *pushout*[3] in the category in which the objects are separated spaces and the arrows are coherent linear maps, which we refer to as **Sep**. For this reason, we refer to our procedure for computing $\langle g_{\Sigma_1}, w_{\Sigma_1} \rangle$ and $\langle g_{\Sigma_2}, w_{\Sigma_2} \rangle$ as $pushout_{\mathbf{Sep}}$. For readability, we present this procedure via the following example; the technical definition of $pushout_{\mathbf{Sep}}$ can be found in the proof of Lemma 5.6.

[3]The standard definition of pushouts requires that the coherent linear map $\langle u, w_u \rangle$ is unique, but this uniqueness is not necessary to produce monotone summaries. For the remainder of the paper, we use a weakened definition of pushouts without the uniqueness condition.

Example 5.5. A key technical tool used in $pushout_{\mathbf{Sep}}$ is that we can compute the pushout in the category of rational vector spaces. Given two linear functions $f_1 : A \to B$ and $f_2 : A \to C$ where A, B, C are rational vector spaces, the pushout of f_1 and f_2 is a rational vector space D and two functions $g_1 : B \to D$ and $g_2 : C \to D$ such that:

(1) $g_1 \circ f_1 = g_2 \circ f_2$
(2) For any rational vector space D' and linear $g'_1 : B \to D', g'_2 : C \to D'$ such that $g'_1 \circ f_1 = g'_2 \circ f_2$, there exists a unique function $u : D \to D'$ such that $u \circ g_1 = g'_1$ and $u \circ g_2 = g'_2$

Consider the VASR reflections of $t\!f|_{\{a\}}$ and $t\!f|_{\{b\}}$ shown in Example 5.3. We are searching for coherent linear maps $\langle g_a, w_a \rangle$ and $\langle g_b, w_b \rangle$ meeting the aforementioned properties. We will compute these coherent linear maps by considering each pair of coherence classes C_a, C_b of the VASRs, computing the pushout in the category of rational vector spaces of $proj_{C_a} \circ f_a$ and $proj_b \circ f_b$, and "stacking" the outputs to form the final coherent linear maps. This approach compositionally leverages the properties of the pushout in the category of rational vector spaces to produce the pushout in the category of separated spaces.

$\mathcal{V}_a$ has one coherence class $C_a \triangleq \mathbb{Q}^{\{y_1, y_2, y_3\}}$ and $\mathcal{V}_b$ has two coherence classes $C_b \triangleq \left\{\rho \in \mathbb{Q}^{\{y_4, y_5, y_6\}} : \rho(y_5) = 0, \rho(y_6) = 0\right\}$ and $C'_b \triangleq \left\{\rho \in \mathbb{Q}^{\{y_4, y_5, y_6\}} : \rho(y_4) = 0\right\}$. The orthogonal projection functions onto these classes are:

$$\begin{array}{lll}
proj_{C_a}(\rho)(y_1) = \rho(y_1) & proj_{C_b}(\rho)(y_4) = \rho(y_4) & proj_{C'_b}(\rho)(y_4) = 0 \\
proj_{C_a}(\rho)(y_2) = \rho(y_2) & proj_{C_b}(\rho)(y_5) = 0 & proj_{C'_b}(\rho)(y_5) = \rho(y_5) \\
proj_{C_a}(\rho)(y_3) = \rho(y_3) & proj_{C_b}(\rho)(y_6) = 0 & proj_{C'_b}(\rho)(y_6) = \rho(y_6)
\end{array}$$

The functions g_1, g_2 of the pushout of $proj_{C_a} \circ f_a$ and $proj_{C_b} \circ f_b$ are:

$$g_1(\rho)(z_1) = \rho(y_2) \qquad g_2(\rho)(z_1) = \rho(y_4)$$

The functions g_3, g_4 of the pushout of $proj_{C_a} \circ f_a$ and $proj_{C'_b} \circ f_b$ are:

$$g_3(\rho)(z_2) = \rho(y_1 + y_2) \qquad g_4(\rho)(z_2) = \rho(y_5)$$

$$g_3(\rho)(z_3) = \rho(y_3) \qquad g_4(\rho)(z_3) = \rho(y_6)$$

The pushout of $\langle f_a, \tilde{w} \rangle$ and $\langle f_b, \tilde{w} \rangle$ in the category **Sep** is the separated space $\left\langle \mathbb{Q}^{\{z_1, z_2, z_3\}}, \left\{\hat{C}, \hat{C}'\right\} \right\rangle$ and coherent maps $\langle g_a, w_a \rangle$ and $\langle g_b, w_b \rangle$, where:

$$\begin{array}{ll}
g_a(\rho)(z_1) = \rho(y_2) & g_b(\rho)(z_1) = \rho(y_4) \\
g_a(\rho)(z_2) = \rho(y_1 + y_2) & g_b(\rho)(z_2) = \rho(y_5) \\
g_a(\rho)(z_3) = \rho(y_3) & g_b(\rho)(z_3) = \rho(y_6)
\end{array}$$

$$\hat{C} \triangleq \left\{\rho \in \mathbb{Q}^{\{z_1, z_2, z_3\}} : \rho(z_2) = 0, \rho(z_3) = 0\right\} \qquad \hat{C}' = \left\{\rho \in \mathbb{Q}^{\{z_1, z_2, z_3\}} : \rho(z_1) = 0\right\}$$

Witness function w_a sends C and C' to C_a; w_b sends C to C_b and C' to C'_b. ⌟

Lemma 5.6. *The category* **Sep** *has pushouts.*

Theorem 5.7. *Consider a transition assignment $t\!f : \Sigma \to TF(X)$ and a partition Σ_1, Σ_2 of Σ. Let $\left\langle f_{\Sigma_1}, \mathcal{V}_{\Sigma_1} \right\rangle$ and $\left\langle f_{\Sigma_2}, \mathcal{V}_{\Sigma_2} \right\rangle$ be VASR reflections of $t\!f|_{\Sigma_1}$ and $t\!f|_{\Sigma_2}$ respectively, and let*

$$\langle S(\mathcal{V}), \langle a, w_a \rangle, \langle b, w_b \rangle \rangle = pushout_{\mathbf{Sep}}(\left\langle f_{\Sigma_1}, \tilde{w} \right\rangle, \left\langle f_{\Sigma_2}, \tilde{w} \right\rangle)$$

Then, $\left\langle a \circ f_{\Sigma_1}, \mathcal{V} \right\rangle$ is a VASR reflection of $t\!f$, where $\mathcal{V}|_{\Sigma_1} = image(\mathcal{V}_{\Sigma_1}, a)$ and $\mathcal{V}|_{\Sigma_2} = image(\mathcal{V}_{\Sigma_2}, b)$.

Lemma 5.1 describes how to compute the best VASR abstraction of a single transition formula and Theorem 5.7 describes how to combine reflections. Together, they describe a divide-and-conquer approach to computing the VASR reflection of any transition formula system. We describe an efficient algorithmic implementation of this approach in the next section.

5.3 An Efficient Algorithm for Computing Best VASR Abstractions

We can compute the VASR reflection of any transition assignment $t\!f$ via the following algorithm. At a high level, Algorithm 1, *GenVASR*, iteratively applies the combination step described in Theorem 5.7 to combine singleton VASR reflections generated via Lemma 5.1. It does so by computing the $|\Sigma|$-way **Sep** pushout in a forward pass, then computing the images of the VASRs in a backwards pass. The complexity of this algorithm is linear in its calls to $pushout_{\mathbf{Sep}}$, which can cause an exponential blowup in the state space of the resulting reflection as a function of $|\Sigma|$

Input: Transition assignment $t\!f : \Sigma \to TF(X)$
Output: VASR-reflection $\langle f, \mathcal{V}\rangle$

$\left\langle f_{s_i}, \mathcal{V}_{s_i}\right\rangle \leftarrow$ VASR-reflection of $t\!f|_{s_i}$ (Section 5.1) for all $s_i \in \Sigma$
$curr \leftarrow f_{s_1}$
for $i \in [2, \dots, |\Sigma|]$ **do**
  $\langle S_i, \langle a_i, \_\rangle, \langle b_i, \_\rangle\rangle \leftarrow pushout_{\mathbf{Sep}}(curr, f_{s_i})$; /* *invariant:$curr = a_{i-1} \circ \cdots \circ a_2 \circ f_{s_1}$* */
  $curr \leftarrow a_i \circ curr$
$r \leftarrow \mathbb{I}$; /* *identity* */
for $i \in [|\Sigma| \dots 2]$ **do**
  $\mathcal{V}|_{\{s_i\}} \leftarrow image(\mathcal{V}_{s_i}, r \circ b_i)$; /* *invariant: $r = a_{|\Sigma|} \circ \cdots \circ a_{i+1}$* */
  $r \leftarrow r \circ a_i$
$\mathcal{V}|_{s_1} \leftarrow image(\mathcal{V}_{s_1}, r)$
return $\left\langle b_{|\Sigma|} \circ f_{|\Sigma|}, \mathcal{V}\right\rangle$

Algorithm 1: GenVASR: Calculate a VASR-reflection of a transition assignment

Theorem 5.8. *For any transition assignment $t\!f : \Sigma \to TF(X_G)$, $\langle f, \mathcal{V}\rangle = GenVASR(t\!f)$ is a VASR-reflection of $t\!f$.*

6 EXTENSION TO LOSSY VASRS

This subsection briefly discusses an extension of our summarization procedure to Lossy-VASRs, highlighting the extensibility of the theory built in Section 5. At a high level, we adapt the divide-and-conquer strategy used to compute VASR reflections to compute Lossy-VASR reflections. Then, the reachability formula of Section 4 is immediately adaptable to Lossy-VASRs; we simply replace the equalities in $Transition(\mathcal{V})$ with inequalities.

Lossy-VASR transitions are a relaxation of VASR transitions to inequalities, but counter-intuitively this means that Lossy-VASRs abstractions are strictly more powerful than VASR abstractions. This is because every VASR is precisely simulated by a Lossy-VASR: for any VASR $\mathcal{V}$ over Y, we can construct an LVASR $\mathcal{LV}$ over a set of variables $\left\{lo_y : y \in Y\right\} \cup \left\{hi_y : y \in Y\right\}$ and a simulation $f : \mathcal{V} \to \mathcal{LV}$ such that $\rho \xrightarrow{s}_{\mathcal{V}} \rho'$ if and only if $f(\rho) \xrightarrow{s}_{\mathcal{LV}} f(\rho')$.

$$\begin{aligned} f(\rho)(lo_y) &\triangleq \rho(y) \\ f(\rho)(hi_y) &\triangleq \rho(-y) \end{aligned} \qquad \mathcal{LV}(s) \triangleq \begin{array}{c} \bigwedge_{y\in Y}(lo'_y \le r_y lo_y + a_y) \\ \wedge\bigwedge_{y\in Y \in |Q|}(hi'_y \le r_y hi_y - a_y) \end{array} \quad \text{where } \mathcal{V}(s) \triangleq \bigwedge_{y\in Y}(y' = r_y y + a_y)$$

Then, a LVASR reflection of a program is guaranteed to capture all information that a VASR reflection does. Conditionals in programs, particularly those involving inequalities, frequently cannot be modeled by VASR transitions but can be modeled by LVASR transitions.

We proceed by computing LVASR reflections of singleton transition assignments and then modify the pushout procedure to account for the new model. Consider a transition formula F. Like before, define the space of resets and increments to be:

$$\text{L-Res}(F) \triangleq \{\langle t, b\rangle \in Lin(X_G) \times \mathbb{Q} : F \models t' \leq b\}$$
$$\text{L-Add}(F) \triangleq \{\langle t, b\rangle \in Lin(X_G) \times \mathbb{Q} : F \models t' \leq t + b\}$$

L-Res(F) and L-Add(F) are convex cones; we define the LVASR reflection of a singleton transition assignment using the generator representations of L-Res$(\mathit{tf}(s))$ and L-Add$(\mathit{tf}(s))$ as in Section 5.1.

Following the pattern developed in Section 5.2, we may reduce the problem of merging two LVASR abstractions over disjoint alphabets to computing a pushout in an appropriate category. Intuitively, the additional constraint (besides coherence) that must be satisfied by a linear simulation f from $\mathcal{LV}$ to $\mathcal{LV}'$ is that it is *non-negative*: for each variable z of $\mathcal{LV}'$, we have $f(\rho)(z) = \rho(a_1y_1 + \ldots a_ny_n)$ when each $a_i \geq 0$. We augment the category **Sep** from Section 5 as follows.

An *ordered* vector space is a vector space V equipped with a partial order $\leq_V$ on V. A *positive map* f between ordered spaces V and V' is linear map that is monotone with respect to this order: $u \leq_V v$ implies that $f(u) \leq_{V'} f(v)$. A *separated ordered space* is a separated space $S = \langle V_S, D_S\rangle$ in which V_S equipped with an partial order under which V_S and each space in D_S is an ordered vector space. Let $\mathbf{Sep}^{\leq}$ be the category in which the objects are separated ordered spaces and the arrows are positive coherent linear maps. The category $\mathbf{Sep}^{\leq}$ has pushouts, following a construction analogous to that in the category **Sep** (substituting pushouts in the category of rational vector spaces with weak pushouts in the category of *ordered* rational vector spaces, explained in the Appendix). Following similar reasoning to Section 5, we obtain an LVASR reflection of a transition assignment $\mathit{tf} : \Sigma \to TF(X)$ by splitting the alphabet Σ into two $\Sigma = \Sigma_1 \uplus \Sigma_2$, computing $\mathcal{LV}$ reflections $\left\langle f_{\Sigma_1}, \mathcal{LV}_{\Sigma_1}\right\rangle$ and $\left\langle f_{\Sigma_2}, \mathcal{LV}_{\Sigma_2}\right\rangle$ of $\mathit{tf}|_{\Sigma_1}$ and $\mathit{tf}|_{\Sigma_2}$ respectively, computing the pushout g_1 and g_2 of f_{Σ_1} and f_{Σ_2} in $\mathbf{Sep}^{\leq}$, and then taking the reflection to be $\left\langle g_1 \circ f_{\Sigma_1}, image(g_1, \mathcal{LV}_{\Sigma_1}) \uplus image(g_2, \mathcal{LV}_{\Sigma_2})\right\rangle$.

7 BOUNDING RECURSIVE DEPTH WITH POTENTIALS

Our summarization procedure works by computing an abstraction (VASR or LVASR) of the semantics of a program and computing the reachability of this abstraction along the context-free language of trajectories through the program. This section improves the precision of this summary by refining the language of trajectories that are considered. It does so by taking advantage of our characterization of CFL-reachability using abstract trajectories, which are essentially a *counting abstraction* of context-free languages. Our insight is that we can compute other counting abstractions of the language of executions of a program to further constrain the abstract trajectories and thereby refine the considered language. In particular, we show how to synthesize *potential functions* to bound the number of invocations of each procedure.

Given a function $f : \Sigma \to \mathbb{Q}$, let $\hat{f} : \mathcal{N}(\Sigma, P) \to \mathbb{Q}$ be the function computing the f-*count* of a nested trajectory: $\hat{f}(\tau) = \sum_{s\in\Sigma} f(s) \cdot \pi(flat(\tau))(s)$. In Figure 3, since every call to `save_tree` is preceded by a c edge, letting $\#(s) = (\text{ if } s = c \text{ then } 1 \text{ else } 0)$, the #-count of a nested trajectory is the number of times `save_tree` is invoked within it.

Our aim is to synthesize potential functions [Tarjan 1985] $\nu : (P \times \mathbb{Q}^X) \to \mathbb{Q}$ for # such that if $\tau \in \mathcal{T}_M(p)$ and $\rho \xrightarrow{\tau}_{\mathit{tf}} \rho'$, then $\hat{\#}(\tau) \leq \nu(p, \rho)$. Put into words, if τ is a nested trajectory through procedure p that can be executed from some input state ρ, then $\nu(p, \rho)$ is an upper bound on the number of times `save_tree` is invoked in τ. The key insight is that $\hat{\#}(\tau)$ can be computed from an abstract trajectory of $flat(\tau)$ and that $\nu(p, \rho)$ can be computed from ρ; we can symbolically bound the variables of our summary representing the abstract trajectory to obey synthesized bounds and thereby refine the summarized language.

This section synthesizes potential functions and uses them to refine our summary as follows. First, we define *inductive potentials*, which give a sufficient "local" condition for a function to be a potential. Second, we show how to transform a program by adding variables which capture the necessary information to symbolically check this local condition, and use intra-procedural summarization of the transformed program to synthesize a convex polyhedron such that each point in the polyhedron corresponds to an inductive potential function. Third, we show how to encode all bounds that arise from a convex polyhedron of potential functions into a LIRA formula.

7.1 Defining Inductive Upper Potentials

Our method is described for a generic counting scheme $f : \Sigma \to \mathbb{Q}$. We constrain our attention to potential functions for f which satisfy an *inductiveness* condition that can be checked with an intra-procedural analysis. At a high level, a potential function ν is inductive if $\nu(p, \rho)$ is greater than or equal to the f-count of any nested trajectory through procedure p with input ρ where ν is used as an approximation of the f-count of any sub-nested trajectory.

Formally, consider a program graph M, a transition assignment $\mathit{tf} : \Sigma \to TF(X)$, and a partition of X into local variables X_L and global variables X_G. Suppose that for each procedure p, $S(p)$ is an over-approximate procedure summary for p (perhaps trivial – e.g., $S(p) = \bigwedge_{x \in X_L} x' = x$) such that if $\rho \xrightarrow{\langle p,\tau\rangle}_{\mathit{tf}} \rho'$ for any $\tau \in \mathcal{T}_M(p)$, then $[\rho, \rho'] \models S(p)$. Then $\mathit{tf} \uplus S$ defines a labeled transition system over the alphabet $\Sigma \cup P$. For any trace $e = \rho_0 \ldots \rho_n$ in this transition system of a trajectory $w = w_1 \ldots w_n \in (\Sigma \cup P)^n$, define:

$$\nu^*(f, e, w) = \left(\sum_{i \in [n], w_i \in \Sigma} f(w_i) \right) + \left(\sum_{i \in [n], w_i \in P} \nu(w_i, \rho_{i-1}) \right)$$

Let $skim : \mathcal{N}(\Sigma, P) \to (\Sigma \cup P)^*$ be the homomorphism sending $s \in \Sigma$ to s and $\langle p, \tau\rangle \in P \times \mathcal{N}(\Sigma, P)$ to p. If e is a trace of nested trajectory τ in tf, then since S is over-approximate e is also a trace of trajectory $skim(\tau)$ in $\mathit{tf} \uplus S$; then $\nu^*(f, e, skim(\tau))$ can be understood as an approximation of $\hat{f}(\tau)$ where ν is used in place of $\hat{f}$ for all recursive calls $\langle p, \tau'\rangle$ within τ.

We say that ν is an **inductive upper potential** for f if for any valuation ρ, procedure p, and nested trajectory $\tau \in \mathcal{T}_M(p)$:

$$\begin{pmatrix} \rho_0 \ldots \rho_{|\tau|} \text{ is a trace of} \\ skim(\tau) \text{ in } \mathit{tf} \uplus S \end{pmatrix} \implies \nu(p, \rho_0) \geq \nu^*(f, \rho_0 \ldots \rho_{|\tau|}, skim(\tau))$$

If ν meets this condition, then by induction over the structure of τ:

$$\begin{pmatrix} \rho_0 \ldots \rho_{|\tau|} \text{ is a trace of} \\ \tau \text{ in } \mathit{tf} \end{pmatrix} \implies \begin{pmatrix} \rho_0 \ldots \rho_{|\tau|} \text{ is a trace of} \\ \tau \text{ in } \mathit{tf} \uplus S \end{pmatrix} \implies \hat{f}(\tau) \leq \nu^*(f, \rho_0 \ldots \rho_{|\tau|}, skim(\tau)) \leq \nu(p, \rho_0)$$

We further constrain our attention to potentials of the form $\nu_\theta(p, \rho) = \max(0, \rho(\theta(p))$ where $\theta : P \to Lin(X)$. In the remaining subsections, we will:

(1) Compute a convex polyhedron $UB(M, \mathit{tf}, S, f) \subseteq (P \to Lin(X))$ such that for all $\theta \in UB(M, \mathit{tf}, S, f)$, we have ν_θ is an inductive upper potential on f.
(2) Define, given a polyhedron $UB \in P \to Lin(X)$, a formula $B_\uparrow(X, UB, p)$ with free variables $X \cup \{\xi\}$ such that $\rho \models B_\uparrow(X, UB, p)$ if and only if $\rho(\xi) \leq \nu_\theta(p, \rho)$ for all $\theta \in UB$.

7.2 Computing a Polyhedron of Inductive Upper Potentials

This section shows how to use an intraprocedural analysis to compute a convex polyhedron $UB(M, tf, S, f)$ of functions $\theta : P \to Lin(X)$ such that v_θ is an inductive upper potential for f. That is, we are interested in finding θ such that for all procedures p, valuations ρ, and nested trajectories $\tau \in \mathcal{T}_M(p)$, if $e = \rho, \ldots, \rho'$ is a trace of $skim(\tau)$ in $tf \uplus S$ then $v_\theta(p, \rho) \geq v^*_\theta(f, e, skim(\tau))$. The following example motivates our approach.

Example 7.1. Consider the program graph M and transition assignment tf displayed in Figure 3. We write `save_tree` as $\underline{\texttt{st}}$. Let $S : P \to TF(X)$ be the trivial procedure summary assignment sending $\underline{\texttt{st}}$ to $same(X_L)$. This example computes potential functions of $\#(s) = (\text{ if } s = c \text{ then } 1 \text{ else } 0)$; in other words, bounds on the number of calls to $\underline{\texttt{st}}$ within executions of $\underline{\texttt{st}}$.

For any trajectory τ through $\underline{\texttt{st}}$, observe that $skim(\tau)$ is equal to either b or $ac\underline{\texttt{st}}c\underline{\texttt{st}}$. As a stepping stone, consider the problem of synthesizing potentials of the form $v_\theta(p, \rho) = \rho(\theta(p))$ where $\theta : P \to Lin(X)$. Our approach is to compute an intra-procedural summary F of $tf \uplus S$ over all possible $skim(\tau)$ through $\underline{\texttt{st}}$ instrumented with extra variables ctr and $\{d_x : x \in X\}$ where ctr tracks the sum of $f(s)$ for every $s \in \Sigma$ in $skim(\tau)$ and d_x tracks the sum of the values of x at each invocation of $\underline{\texttt{st}}$. We then compute the set of θ such that

$$F \models \theta(\underline{\texttt{st}}) \geq ctr' + \theta(\underline{\texttt{st}})[x \mapsto d'_x]$$

Then, for every trace $\rho_0\rho_1$ of b in $tf \uplus S$, we have $[\bar{\rho}_0, \bar{\rho}_1] \models F$ where $\bar{\rho}$ is the extension of ρ to the variables ctr and d_x. Thus,

$$v_\theta(p, \rho_0) = \bar{\rho}_0(\theta(\underline{\texttt{st}})) \geq \bar{\rho}_1(ctr) + \bar{\rho}_1(\theta(\underline{\texttt{st}})[x \mapsto d_x]) = 0 = v^*_\theta(\#, \rho_0\rho_1, b)$$

For every trace $\rho_0 \ldots \rho_5$ of $ac\underline{\texttt{st}}c\underline{\texttt{st}}$ in $tf \uplus S$, we have $[\bar{\rho}_0, \bar{\rho}_5] \models F$, and thus

$$v_\theta(p, \rho_0) = \bar{\rho}_0(\theta(\underline{\texttt{st}})) \geq \bar{\rho}_5(ctr) + \bar{\rho}_5(\theta(\underline{\texttt{st}})[x \mapsto d_x]) = 2 + (\rho_2 + \rho_4)(\theta(\underline{\texttt{st}})) = v^*_\theta(\#, \rho_0 \ldots \rho_5, ac\underline{\texttt{st}}c\underline{\texttt{st}})$$

The key trick to this approach is that we exploit the linearity of θ to compute $\rho_2(\theta(\underline{\texttt{st}}))+\rho_4(\theta(\underline{\texttt{st}}))$ from $\rho_2 + \rho_4$, which is symbolically captured by $\{d_x : x \in X\}$. We now adapt this approach to the full template $v_\theta(p, \rho) = \max(0, \rho(\theta(p)))$ where $\theta : P \to Lin(X)$.

For any trace $\rho_0\rho_1$ of b in $tf \uplus S$, we have $v^*_\theta(\#, \rho_0\rho_1, b) = 0$ so by the definition of our template we have $v_\theta(p, \rho_0) \geq v^*_\theta(\#, \rho_0\rho_1, b)$ for any θ. We additionally instrument $tf \uplus S$ with an additional variable rec tracking whether a recursive call occurs in $skim(\tau)$ and refine our summary F by conjoining $(rec' = 1 \vee ctr' > 0)$ to ignore such cases where v^*_θ must be zero.

For any trace $\rho_0 \ldots \rho_5$ of $ac\underline{\texttt{st}}c\underline{\texttt{st}}$ in $tf \uplus S$, we can use the following equivalences to translate the condition $v_\theta(p, \rho_0) \geq v^*_\theta(\#, \rho_0 \ldots \rho_5, ac\underline{\texttt{st}}c\underline{\texttt{st}})$ into a conjunction of inequalities with the same form as the linear template.

$$\begin{array}{rrl} & \rho_0(\max(0, \theta(\underline{\texttt{st}})) & \geq 2 + \rho_2(\max(0, \theta(\underline{\texttt{st}}))) + \rho_4(\max(0, \theta(\underline{\texttt{st}}))) \\ \Longleftrightarrow & \rho_0(\theta(\underline{\texttt{st}})) & \geq 2 + \max_{\sigma \in \{\rho_2, 0\}, \sigma' \in \{\rho_4, 0\}} (\sigma + \sigma')(\theta(\underline{\texttt{st}})) \\ \Longleftrightarrow & \bigwedge_{\sigma \in \{\rho_2, 0\}, \sigma' \in \{\rho_4, 0\}} \rho_0(\theta(\underline{\texttt{st}})) & \geq 2 + (\sigma + \sigma')(\theta(\underline{\texttt{st}})) \end{array}$$

We modify our extended program by non-deterministically incrementing d_x by x or by 0 at each invocation - in this way, the possible valuations of d_x capture all options for $\sigma + \sigma'$. ⌟

Our high level approach is to construct an augmented transition assignment $tf_\uparrow$ over the alphabet $\Sigma \cup P$ such that traces of $skim(\tau)$ in $tf \uplus S$ correspond to traces in $tf_\uparrow$ in which the extra variables of $tf_\uparrow$ capture enough information to compute $v^*_\theta(f, e, skim(\tau))$; we then use consequence-finding on an intra-procedural summary of $tf_\uparrow$ to compute a polyhedron of all inductive upper potentials.

Define $tf_\uparrow : (\Sigma \cup P) \to TF(X \cup D \cup \{ctr, rec\})$ as follows, where $D \triangleq \bigcup_{p \in P} D_p$ and $D_p \triangleq \{d_{x,p} : x \in X\}$. The mapping $tf_\uparrow$ will meet the conditions for all $\tau \in \mathcal{T}_M(p)$ that if $e = \rho_0 \ldots \rho_{|\tau|}$ is a trace of $skim(\tau)$ in $tf \uplus S$ such that $v^*_\theta(f, e, skim(\tau)) > 0$, then there is a trace $\bar{e} = \bar{\rho}_0 \dots \bar{\rho}_{|\tau|}$ of $skim(\tau)$ in $tf_\uparrow$ such that:

(1) $\bar{\rho}_i|_X = \rho_i$ and $\bar{\rho}_0(z) = 0$ for all $z \in D \cup \{ctr, rec\}$
(2) $v^*_\theta(f, e, skim(\tau)) = \bar{\rho}_{|\tau|}\left(\sum_{p \in P} \theta(p)[X \mapsto D_p] + ctr\right)$
(3) if $skim(\tau) \in \Sigma^*$ then $\bar{\rho}_{|\tau|}(rec) = 0$ else $\bar{\rho}_{|\tau|}(rec) = 1$

The first condition ensures that $\bar{e}$ represents the same computation as e, and the second condition allows us to compute $v^*_\theta(f, e, skim(\tau))$ from the post valuation $\bar{\rho}_{|\tau|}$. The third condition describes the behavior of rec. The definition of $tf_\uparrow$ is as follows, where $same(X) \triangleq \bigwedge_{x \in X} x' = x$:

- For all $s \in \Sigma$, $tf_\uparrow(s) \triangleq tf(s) \wedge same(D \cup \{rec\}) \wedge ctr' = ctr + f(s)$
- For all $p \in P$, $tf_\uparrow(p) \triangleq S(p) \wedge same((D \backslash D_p) \cup \{ctr\}) \wedge rec' = 1 \wedge (same(D_p) \vee \vec{D}'_p = \vec{D}_p + \vec{X})$

Let $Intra$ refer to an intra-procedural analysis function, the inputs to which are a graph $\langle V, E\rangle$ where $E \subseteq V \times \Sigma \times V$, a transition assignment $tf \colon \Sigma \to TF(X)$, a source $src \in V$ and a target vertex $tgt \in V$. Its output is a transition formula F such that $[\rho, \rho'] \models F$ if and only if $\rho \xrightarrow{w}_{tf} \rho'$ where w is a path through the graph between src and tgt. We assume $Intra(\langle V, E\rangle, tf, src, tgt)$ is monotone: if $tf(s) \models tf'(s)$ for all $s \in E$, then $Intra(\langle V, E\rangle, tf, src, tgt) \models Intra(\langle V, E\rangle, tf', src, tgt)$.

If our program graph is $M = \langle V, \Sigma, P, E, in, out\rangle$, then $Intra(\langle V, E\rangle, tf_\uparrow, in(p), out(p))$ is a transition formula over-approximating $\bar{\rho} \xrightarrow{skim(\tau)}_{tf_\uparrow} \bar{\rho}'$ for all $\tau \in \mathcal{T}_M(p)$. We conjoin additional formulae to initialize our variables and ignore cases in which we know the f-count to be non-positive:

$$F \triangleq Intra(\langle V, E\rangle, tf_\uparrow, in(p), out(p)) \wedge \left(\bigwedge_{v \in \{ctr, rec\} \cup D} v = 0\right) \wedge (rec' = 1 \vee ctr' > 0)$$

Finally, we define $UB(M, tf, S, f)$ to be $\bigcap_{p \in P} UB_p$ where:

$$UB_p \triangleq \left\{\theta : P \to Lin(X) : F \models \theta(p) \geq ctr' + \sum_{p \in P} \theta(p)[X \mapsto D'_p]\right\}$$

All $\theta \in UB(M, tf, S, f)$ represent inductive upper potentials. The argument is as follows: consider any trace $\rho \dots \rho'$ of nested trajectory τ. If $v^*_\theta(f, \rho \dots \rho', skim(\tau)) \leq 0$ then then $v_\theta(p, \rho) \geq v^*_\theta(f, \rho \dots \rho', skim(\tau))$ by the template. Otherwise, by the definition of $tf_\uparrow$, there exist valuations $\bar{\rho}, \bar{\rho}'$ such that $[\bar{\rho}, \bar{\rho}'] \models F$, $\bar{\rho}|_X = \rho$, $\bar{\rho}'|_X = \rho'$, and $v^*_\theta(f, e, skim(\tau)) = \bar{\rho}'(counter + \sum_{p \in P} \theta(p)[X \mapsto D_p])$; then, $v_\theta(p, \rho) \geq \bar{\rho}'(\theta(p)) \geq v^*_\theta(f, e, skim(\tau))$.

Theorem 7.2. *Consider a program graph M, a transition assignment $tf \colon \Sigma \to TF(X)$, a procedure summary map $S : P \to TF(X)$, and a function $f : \Sigma \to \mathbb{Q}$. Let $\theta \in UB(M, tf, S, f)$. For any valuations ρ, ρ', procedure p, and nested trajectory $\tau \in \mathcal{T}_M(p)$ such that $\rho \xrightarrow{\tau}_{tf} \rho'$, we have $\hat{f}(\tau) \leq v_\theta(p, \rho)$.*

Lemma 7.3 (Anti-monotonicity). *For any two transition assignments $tf, tf' : \Sigma \to TF(X)$ and two summary assignments $S, S' : P \to TF(X)$, if $tf(s) \models tf'(s)$ for all $s \in \Sigma$ and $S(p) \models S'(p)$ for all $p \in P$, then $UB(M, tf', S', f) \subseteq UB(M, tf, S, f)$.*

7.3 Applying All Constraints of a Polyhedron of Upper Inductive Potentials

Suppose that $UB \subseteq P \to Lin(X)$ is a convex polyhedron and $p \in P$ is a procedure. In this section, we should how to construct a formula $B_\uparrow(X, X, p)$ with free variables in X plus a designated variable ξ such that $\rho \models B_\uparrow(X, UB, p)$ if and only if $\rho(\xi) \leq v_\theta(p, \rho)$ for all $\theta \in UB$. This formula will allow us to refine $AT(\mathcal{V}, G)$ from Section 4 by replacing ξ with relevant terms.

Let $\{v_1, \ldots, v_n\}$ and $\{r_1, \ldots, r_m\}$ be the vertices and rays of the generator representation of UB, respectively. Then, the following formula encodes our bound into LIRA:

$$B_{\uparrow}(X, UB, p) \triangleq \left(\left(\bigvee_{i \in [m]} r_i(p) < 0\right) \land \xi \le 0\right) \lor \left(\left(\bigwedge_{i \in [m]} r_i(p) \ge 0\right) \land \bigwedge_{i \in [n]} \xi \le \max(0, v_i(p))\right)$$

Lemma 7.4. *Let P be a set of procedure identifiers and let $tf : \Sigma \to TF(X)$ be a transition assignment. Consider any convex polyhedron $UB \subseteq P \to Lin(X)$. For any valuation ρ, we have $\rho \models B_{\uparrow}(X, UB, p)$ if and only if $\rho(\xi) \le \nu_\theta(p, \rho)$ for all $\theta \in UB$.*

A sketch of the proof of this lemma is that if there is a ray r_i such that $\rho(r_i(p)) < 0$, for any inductive potential θ we have that $\theta + \alpha r_i$ is an inductive potential for all $\alpha > 0$, and by linearity there must some value of α such that $\nu_{\theta+\alpha r_i}(p, \rho) = 0$. Otherwise, the least inductive upper bound in UB must be one of the vertices by convexity.

The work of this section can be straightforwardly extended to the template $\nu(p, \rho) = \max(0, \rho(\theta(p)))$ where $\theta \in P \to Aff(X)$ where $Aff(X)$ denotes the set of affine terms over X. Additionally, we can modify the procedure to generate and apply lower bounds; this is discussed in the Appendix.

8 AN END-TO-END SUMMARIZATION PROTOCOL

Here, we specify our end-to-end summarization procedure and formalize its monotonicity.

(1) The input is a program graph $M = \langle V, \Sigma, P, E, in, out \rangle$ representing the structure of the input procedure, a transition assignment $tf : \Sigma \to TF(X)$ over a set of local variables X_L and a set of global variables X_G, and a procedure to summarize p. We will assume that all calls to $p' \in P$ are preceded by a designated edge $begin(p') \in \Sigma$.
(2) Using the divide-and-conquer approach of Section 5 with the theory of Section 6, compute the best LVASR abstraction $\langle f, \mathcal{LV} \rangle$ of tf.
(3) Recalling that $\mathcal{G}(M, p)$ is the grammar generating $\mathcal{L}_M(p)$, use Section 4 to compute the CFL-reachability formula $\exists C.Reach(\mathcal{LV}, \mathcal{G}(M, p^*))$ where $C = \{c_{s,i} : s \in \Sigma, i \in [2d+1]\}$.
(4) Let $\#_{p'}(s) = (\text{if } s = begin(p') \text{ then } 1 \text{ else } 0)$ and $S(p') = \bigwedge_{x \in X_L} x' = x$ for all $p' \in P$. Using Section 7, compute the upper-bounding formula $B_{\uparrow}(X, UB(M, tf, S, \#_{p'}), p)$ for all $p' \in P$.
(5) Our final summary for p is the following transition formula:

$$Summary(M, tf, p) \exists C. \begin{pmatrix} Reach(\mathcal{LV}, \mathcal{G}(M, p))[Y \mapsto \text{SUB}_f(Y), Y' \mapsto \text{SUB}_f(Y)'] \\ \land \bigwedge_{p' \in P} B_{\uparrow}(X, UB(M, tf, S, \#_{p'}), p)[\xi \mapsto \sum_{i \in [2d+1]} c_{begin(p'),i} \end{pmatrix}$$

By Theorems 5.8, 4.6, and 7.2, we know for all valuations ρ, ρ', and trajectories $\tau \in \mathcal{T}_M(p^*)$ if $\rho \xrightarrow{\tau}_{tf} \rho'$, then $\rho, \rho' \models Summary(M, tf, p^*)$. Furthermore, *Summary* is *monotone*:

Theorem 8.1 (Monotonicity). *For any transition assignments tf and tf' such that $tf(s) \models tf'(s)$ for all symbols s, program graph M, and procedure p,*

$$Summary(M, tf, p) \models Summary(M, tf', p)$$

9 EVALUATION

This evaluation seeks to answer: (1) How does the precision of our summarization technique compare to state-of-the-art abstract interpreters? (2) How do the verification capabilities of a automated verifier based on our technique compare to state-of-the-art verifiers? (3) How do the refinements in Sections 6 and 7 affect the precision of the summaries?

	Rec-Supreme (60)			Recursive (17)			Recursive-simple (35)		
	✓	?	TO	✓	?	TO	✓	?	TO
LiP (*LVASR + Poten.*)	28	32	0	3	13	1	20	15	0
- *LVASR, No Poten.*	26	34	0	3	13	1	20	15	0
- *VASR + Poten.*	20	40	0	2	15	0	14	21	0
- *VASR, No Poten.*	14	46	0	2	15	0	13	22	0
CRA	16	44	0	4	13	0	13	22	0
LiP ∧ CRA	29	31	0	5	11	1	20	15	0
Korn	19	1	40	13	1	3	35	0	0
UAutomizer	23	0	37	11	0	6	22	0	13
Goblint	6	54	0	3	14	0	20	15	0

Fig. 6. Columns headers denote benchmark sets and # tasks per task. Row headers denote program verification tools; tools above the midrule are monotone. Table entries denote # of solves, unknowns, and timeouts.

Implementation. Our summarization methods are implemented in an algebraic program analyzer called LiP (LVASR summarization with Inductive Potentials). It uses a monotone variant of Compositional Recurrence Analysis (CRA) [Farzan and Kincaid 2015] as a backend intraprocedural analyzer to verify safety properties using procedure summaries computed using our technique. We evaluate three baseline methods representing our summarization routine without refinements. We also compare LiP with an instantiation of CRA with a different monotone procedure summarization technique, which is a classical abstract-interpretation-based analysis using the reduced product of a signed domain and the domain of affine relations.[4] Finally, we compare with the combination (LiP ∧ CRA) of both summarization techniques (which simply conjoins the summaries produced by each); since both techniques are monotone, their conjunction is also monotone.

Other tools. To answer research question 1, we compare our tool against monotone CRA and Goblint [Vojdani et al. 2016], a (non-monotone) abstract interpreter that performed well in SV-COMP 2023 [Beyer 2023]. To answer research question 2, we compare with the first and second place finishers in the ReachSafety Recursive category of SV-COMP 2023, Korn [Ernst 2020], a portfolio solver based on Spacer [Komuravelli et al. 2014] and Eldarica [Hojjat and Rümmer 2018], and UAutomizer [Heizmann et al. 2013], an a model checker based on trace abstraction.

Our experiments should be viewed with the qualitative differences between model checkers (Korn, UAutomizer) and abstract interpreters (LiP, CRA, Goblint) in mind. Abstract interpreters are terminating invariant generation algorithms, whereas software model checkers are semi-algorithms that verify or refute a given property. The two tool categories have different strengths and capabilities—software model checkers can refute safety properties, and abstract interpreters have clients beyond verification (e.g., compilers, resource bound analyzers, as a pre-processing step for software model checkers). Our experiments compare these tools on (safe) verification tasks, which lies at the intersection of their capabilities.

Evaluation Tasks. Since procedure summarization is only relevant in the presence of recursive procedures, we restrict our attention to verification tasks on recursive programs; for non-recursive procedures our tool is identical to CRA. We compiled a diverse set of recursive procedures in a suite called Rec-Supreme (available in supplement). We include the safe tasks from the Recursive and Recursive-Simple sets from SV-Comp in our evaluation, but remark that these sets have weaknesses as evaluation metrics. Firstly, these sets have low diversity—the sets include 18 variants of the

[4]This domain satisfies the ascending chain condition, which avoids the need for (non-monotone) widening operators.

Fibonacci function, 12 variants of the identity function, and 8 variants of a recursive implementation of addition. Secondly, the suite contains a large number of tasks that can be verified by unrolling (e.g., proving that `Fibonacci(9) == 34`), for which abstract interpreters are ill-suited.

Timings were gathered on a virtual machine running Ubuntu 20.04 with 8 GB of RAM with access to a 2.3 GHz Intel i7 CPU. The time limit was 10 minutes per task. The source code of *LiP* can be accessed online [anonymized]. Data can be found in Figure 6.

9.1 RQ1: Precision Comparison with Abstract Interpreters

Our results show that our summarization technique is a step forward for abstract interpretations of recursive programs. LiP significantly outperforms CRA and Goblint on Rec-Supreme; the three techniques have similar performance on the SV-Comp benchmarks. Since both LiP and CRA are monotone, their summaries can be conjoined while preserving monotonicity. These summaries are complementary, so this conjunction succeeds on some tasks that neither LiP nor CRA succeed on.

9.2 RQ2: Verification Comparison

Our evaluation shows that our method is comparable with state-of-the-art verifiers. LiP outperforms Korn and UAutomizer on Recursive-Supreme but is outperformed on SV-Comp benchmarks. The relative performance of these tools on these sets indicates that these software model checkers would benefit from using LiP as a pre-pass to generate initial summaries for each recursive procedure.

Figure 7 shows timing performance on tasks which were successfully solved. The solve time of LiP is comparable with CRA, Korn, and Goblint and is better than that of UAutomizer. An important practical consideration is that LiP, CRA, and Goblint return unknown within seconds on unsuccessful tasks, but UAutomizer and Korn consume the full time whenever they fail.

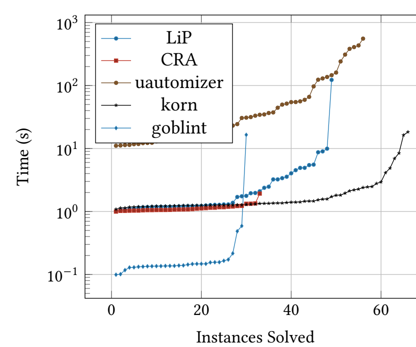


Fig. 7. Cactus Plot over all sets

9.3 RQ3: Component Analysis

By comparing the performance of LiP and the baseline methods, we can conclude that both Lossy VASR and inductive potentials are refinements that meaningfully increase the precision of our procedure summaries. The extension to the domain of Lossy VASRs appears to be a more powerful refinement than the inclusion of inductive potentials. In programs in which the base case is a conditional in terms of variables of the LVASR reflection, the LVASR summary sometimes effectively bounds the number of procedure calls without requiring potential functions.

10 RELATED WORK

Interprocedural analysis. Computing summaries that approximate the dynamics of recursive procedures is a classical problem in program analysis [Cousot and Cousot 1977; Sharir and Pnueli 1978]. The dominant approach is based on *iterative approximation*, which uses the limit of a Kleene iteration sequence as a summary. For abstract domains that fail the ascending chain condition, the limit can be over-approximated using widening; however, this results in a non-monotone analysis.

Newtonian program analysis [Esparza et al. 2010] is a method for solving inter-procedural dataflow equations based on Newton iteration rather than Kleene iteration. Newtonian program analysis relies on commutativity of the sequencing operation rather than the ascending chain condition. Newtonian program analysis cannot compute VASR reachability because resets do not commute.

SLAM [Ball and Rajamani 2001] is a suite of tools which abstract programs as *boolean programs* that have recursive procedures. SLAM iteratively refines its boolean program abstraction and uses pushdown model checking to answer questions about program behavior. SLAM's abstract model is restricted to boolean variables, while the work of this paper operates over rational variables.

Vector addition systems. Vector addition systems are a class of transition systems originally motivated as a model of parallel systems. Classically, vector addition systems operate over vectors of *natural numbers*. The complexity of the reachability problem is non-elementary [Lazic 2013a].

The *integer* VAS model was introduced by Haase and Halfon [2014]. Haase and Halfon showed that the reachability relation $\xrightarrow{L}_{\mathcal{V}}$ can be encoded as a Presburger formula in polynomial time in the case that $L = \Sigma^*$ and the case that L is regular (i.e. integer VASR with states). Chistikov et al. [2015] extend this construction to the case where L is the language of a communication-free Petri net. The approach is based representing the generalized Parikh image for L using a Presburger formula, by decomposing an accepting run through an automaton into phases, and computing a Parikh image for each phase. This approach does not readily extend to the case where L is context-free (i.e. pushdown integer VASR) since it is not clear how to represent such a decomposed accepting run of a pushdown automaton in Presburger arithmetic.

Pushdown vector addition systems over the naturals were investigated in [Ganardi et al. 2022; Lazic 2013b; Leroux et al. 2015]. Whether reachability is decidable for this model is an open problem.

Hague and Lin [2011] show how how to obtain a reachability formula for reversal bounded counter transition systems, which are equivalent to a Rational Vector Addition System (without resets) subject to a context free language via the reduction in Baumann et al. [2023].

Silverman and Kincaid [2019] give a loop summarization algorithm that is based on computing the reachability relation of a VASR that simulates the body of the loop. We generalize this work by computing the CFL-reachability of a VASR that simulates the body of a procedure. We show how to compute best VASR abstractions of transition formulas in LIRA, not just LRA, and we present a coordinate-free theory of VASR abstractions that is conducive to computing best abstraction in extensions of the VASR model, in particular Lossy VASR.

Monotone invariant generation. The classical iterative method for program analysis is monotone for abstract domains satisfying the ascending chain condition, such as affine relation analysis [Karr 1976; Müller-Olm and Seidl 2004] and the Houdini algorithm [Flanagan and Leino 2001]. A recent line of work has designed monotone program analyses [Kincaid et al. 2023; Silverman and Kincaid 2019; Zhu and Kincaid 2021a,b] using *algebraic program analysis* [Kincaid et al. 2021]. This work is intra-procedural, and falls back on an iterative strategy to summarize recursive procedures.

Resource bound analysis. The call-count bounding procedure of Section 7 is based on the potential function method for amortized resource analysis [Tarjan 1985], which also serves as the foundation of several resource bound analyses [Carbonneaux et al. 2015; Hoffmann et al. 2012; Hoffmann and Hofmann 2010]. Carbonneaux et al. [2015] encodes the space of linear potential functions as a linear program and using an LP solver to find an optimum. In contrast, our technique directly manipulates the entire space of potential functions, which is the key to making the analysis monotone.

11 CONCLUSION

This paper presented a compositional and monotone summarization technique for recursive integer programs. It computed the *best VASR abstraction* of an input program and computed the reachability relation of that VASR over the context-free language of paths through the program. Its key technical contributions were (1) a new theory of VASR abstractions and (2) the development of a counting abstraction of context-free languages, abstract trajectories. It leveraged the new theory of VASR abstractions to extend the summarization technique to the domain of Lossy VASRs. It took advantage of the counting abstraction to refine the summary with potential functions constraining the language of syntactic paths considered by the summary. These pieces combined to form an end-to-end summarization procedure which is monotone, giving end-users a theoretical guarantee of predictability of the summarization on related programs. The evaluation of this technique showed that it represents a step forward in monotone program analysis and in abstract-interpretation of procedures, and that its verification capabilities can compete with the state-of-the-art in verification.

REFERENCES

Thomas Ball and Sriram K. Rajamani. 2001. The SLAM Toolkit. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV '01)*. Springer-Verlag, Berlin, Heidelberg, 260–264.

Pascal Baumann, Flavio D'Alessandro, Moses Ganardi, Oscar Ibarra, Ian McQuillan, Lia Schütze, and Georg Zetzsche. 2023. Unboundedness problems for machines with reversal-bounded counters. arXiv:2301.10198 [cs.FL]

Dirk Beyer. 2023. Competition on Software Verification and Witness Validation: SV-COMP 2023. In *Tools and Algorithms for the Construction and Analysis of Systems*, Sriram Sankaranarayanan and Natasha Sharygina (Eds.). Springer Nature Switzerland, Cham, 495–522.

Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. 2015. Compositional Certified Resource Bounds. *SIGPLAN Not.* 50, 6 (jun 2015), 467–478. https://doi.org/10.1145/2813885.2737955

Dmitry Chistikov, Christoph Haase, and Simon Halfon. 2015. Context-Free Commutative Grammars with Integer Counters and Resets. *CoRR* abs/1511.04893 (2015). arXiv:1511.04893 http://arxiv.org/abs/1511.04893

Noam Chomsky. 1959. On certain formal properties of grammars. *Information and Control* 2, 2 (1959), 137–167. https://doi.org/10.1016/S0019-9958(59)90362-6

P. Cousot and R. Cousot. 1977. Static determination of dynamic properties of recursive procedures. In *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*, E.J. Neuhold (Ed.). North-Holland, 237–277.

Gidon Ernst. 2020. A Complete Approach to Loop Verification with Invariants and Summaries. *CoRR* abs/2010.05812 (2020). arXiv:2010.05812 https://arxiv.org/abs/2010.05812

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2010. Newtonian Program Analysis. *J. ACM* 57, 6, Article 33 (nov 2010), 47 pages. https://doi.org/10.1145/1857914.1857917

Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design* (Austin, Texas) *(FMCAD '15)*. FMCAD Inc, Austin, Texas, 57–64.

Cormac Flanagan and K. Rustan M. Leino. 2001. Houdini, an Annotation Assistant for ESC/Java. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity (FME '01)*. Springer-Verlag, Berlin, Heidelberg, 500–517.

Moses Ganardi, Rupak Majumdar, and Georg Zetzsche. 2022. The Complexity of Bidirected Reachability in Valence Systems. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science* (Haifa, Israel) *(LICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 26, 15 pages. https://doi.org/10.1145/3531130.3533345

Christoph Haase and Simon Halfon. 2014. Integer Vector Addition Systems with States. In *Lecture Notes in Computer Science*. Springer International Publishing, 112–124. https://doi.org/10.1007/978-3-319-11439-2_9

Matthew Hague and Anthony Widjaja Lin. 2011. Model Checking Recursive Programs with Numeric Data Types. In *Computer Aided Verification*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 743–759.

Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2013. Software Model Checking for People Who Love Automata. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 36–52.

Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2012. Multivariate Amortized Resource Analysis. *ACM Trans. Program. Lang. Syst.* 34, 3, Article 14 (nov 2012), 62 pages. https://doi.org/10.1145/2362389.2362393

Jan Hoffmann and Martin Hofmann. 2010. Amortized Resource Analysis with Polynomial Potential. In *Programming Languages and Systems*, Andrew D. Gordon (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 287–306.

Hossein Hojjat and Philipp Rümmer. 2018. The ELDARICA Horn Solver. In *2018 Formal Methods in Computer Aided Design (FMCAD)*. 1–7. https://doi.org/10.23919/FMCAD.2018.8603013

Michael Karr. 1976. Affine Relationships among Variables of a Program. *Acta Inf.* 6, 2 (jun 1976), 133–151. https://doi.org/10.1007/BF00268497

Zachary Kincaid. 2018. Numerical Invariants via Abstract Machines. In *Static Analysis*, Andreas Podelski (Ed.). Springer International Publishing, Cham, 24–42.

Zachary Kincaid, Nicolas Koh, and Shaowei Zhu. 2023. When Less Is More: Consequence-Finding in a Weak Theory of Arithmetic. *Proc. ACM Program. Lang.* 7, POPL, Article 44 (jan 2023), 33 pages. https://doi.org/10.1145/3571237

Zachary Kincaid, Thomas Reps, and John Cyphert. 2021. Algebraic Program Analysis. In *Computer Aided Verification*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer International Publishing, Cham, 46–83.

Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. 2014. SMT-based Model Checking for Recursive Programs. arXiv:1405.4028 [cs.LO]

Ranko Lazic. 2013a. The reachability problem for vector addition systems with a stack is not elementary. *CoRR* abs/1310.1767 (2013). arXiv:1310.1767 http://arxiv.org/abs/1310.1767

Ranko Lazic. 2013b. The reachability problem for vector addition systems with a stack is not elementary. *CoRR* abs/1310.1767 (2013). arXiv:1310.1767 http://arxiv.org/abs/1310.1767

Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. 2015. On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension. In *Automata, Languages, and Programming*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 324–336.

Markus Müller-Olm and Helmut Seidl. 2004. Precise Interprocedural Analysis through Linear Algebra. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Venice, Italy) *(POPL '04)*. Association for Computing Machinery, New York, NY, USA, 330–341. https://doi.org/10.1145/964001.964029

Rohit J. Parikh. 1966. On Context-Free Languages. *J. ACM* 13, 4 (oct 1966), 570–581. https://doi.org/10.1145/321356.321364

Thomas Reps, Mooly Sagiv, and Greta Yorsh. 2004. Symbolic Implementation of the Best Transformer. In *Verification, Model Checking, and Abstract Interpretation*, Bernhard Steffen and Giorgio Levi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 252–266.

A. Schrijver. 1986. *Theory of Linear and Integer programming*. Wiley-Interscience.

M Sharir and A Pnueli. 1978. *Two approaches to interprocedural data flow analysis*. New York Univ. Comput. Sci. Dept., New York, NY. https://cds.cern.ch/record/120118

Jake Silverman and Zachary Kincaid. 2019. Loop Summarization with Rational Vector Addition Systems (extended version). arXiv:1905.06495 [cs.PL]

Robert Endre Tarjan. 1985. Amortized Computational Complexity. *SIAM Journal on Algebraic Discrete Methods* 6, 2 (1985), 306–318. https://doi.org/10.1137/0606031 arXiv:https://doi.org/10.1137/0606031

Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. 2005. On the Complexity of Equational Horn Clauses. In *Automated Deduction – CADE-20*, Robert Nieuwenhuis (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–352.

Vesal Vojdani, Kalmer Apinis, Vootele Rõtov, Helmut Seidl, Varmo Vene, and Ralf Vogler. 2016. Static race detection for device drivers: the Goblint approach. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE '16)*. Association for Computing Machinery, New York, NY, USA, 391–402. https://doi.org/10.1145/2970276.2970337

Shaowei Zhu and Zachary Kincaid. 2021a. Reflections on Termination of Linear Loops. In *Computer Aided Verification*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer International Publishing, Cham, 51–74.

Shaowei Zhu and Zachary Kincaid. 2021b. Termination Analysis without the Tears. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 1296–1311. https://doi.org/10.1145/3453483.3454110

A APPENDIX

A.1 Proofs

Lemma 5.1. *For $\mathit{tf} : \Sigma \to TF(X)$ and any $s \in \Sigma$, $\langle f_s, \mathcal{V}_s \rangle$ is a VASR reflection of $\mathit{tf}|_{\{s\}}$.*

Proof. Let $\langle f_s', \mathcal{V}_s' \rangle$ be a different VASR abstraction over variables Y' of the transition system defined by $\mathit{tf}|_{\{s\}}$. Consider variable $y' \in Y$.

If $s \in Reset(\mathcal{V}', y')$, then $\langle \text{SUB}_{f_s'}(y'), Offset(\mathcal{V}', s, y' \rangle \in Res(\mathit{tf}(s))$ since for all $[\rho, \rho'] \models \mathit{tf}(s)$, we have $f_s'(\rho) \xrightarrow{s}_{\mathcal{V}'} f_s'(\rho')$ and so $\rho(\text{SUB}_{f_s'}(y')) = Offset(\mathcal{V}', s, y')$. Otherwise, it must be in $Add(\mathit{tf}(s))$ by similar reason. In either case, since $\{\langle t_1, a_1 \rangle \ldots\}$ and $\{\langle \hat{t}_1, b_1 \rangle \ldots\}$ are bases, there must exist a unique linear function $f^* : Y \to Y'$ such that $\text{SUB}_f(\text{SUB}_{f^*}(y')) = \text{SUB}_{f'}(y')$. Then, f^* is a simulation from $\mathcal{V}_s to \mathcal{V}_s'$ because $\mathcal{V}_s' = image(\mathcal{V}_s, f^*)$. Thus, $\langle f_s, \mathcal{V}_s \rangle$ is a VASR reflection of $\mathit{tf}|_{\{s\}}$.

unique vector $\vec{g}_{y'}$ such that:

$$\begin{bmatrix} t_1 & \ldots & t_n & \hat{t}_1 & \ldots & \hat{t}_n \end{bmatrix} \vec{g}_{y'} = \text{SUB}(y')$$

Then, $g = [\vec{g}_1 \ldots \vec{g}_{|Y^*|}]^T$ is the unique matrix such that $gf_s = f_s'$, and g must be a simulation from $\mathcal{V}_s$ to $\mathcal{V}_s'$ because $\mathcal{V}_s' = image(\mathcal{V}_s, g)$. Thus, $\langle f_s, \mathcal{V}_s \rangle$ is a best VASR abstraction of $\mathit{tf}|_{\{s\}}$. □

Theorem 5.4. *Let $\mathcal{V}$ and $\mathcal{V}^*$ be VASRs. If f is a linear simulation from $\mathcal{V}$ to $\mathcal{V}^*$, then there exists a function $w_f : D_{S(\mathcal{V}^*)} \to D_{S(\mathcal{V})}$ such that $\langle f, w_f \rangle$ is a coherent linear map from $S(\mathcal{V})$ to $S(\mathcal{V}^*)$.*

Proof. Let Y be the variables of $\mathcal{V}$ and Y^* be the variables of $\mathcal{V}^*$. Note that we can write $f(\rho)(y^*) = \rho(a_{y^*}^T \vec{Y})$ for some $a_{y^*}^T$ for all $y^* \in Y^*$.

First, observe that $Offset(\mathcal{V}^*, s, y^*) = [Offset(\mathcal{V}, s, y_1), Offset(\mathcal{V}, s, y_2), \ldots]a_{y^*}$.

$$\begin{aligned}
& [0, \rho'] \models \mathcal{V}(s) && \text{when } \rho'(y) = Offset(\mathcal{V}, s, y)\\
\Longrightarrow\ & [0, f(\rho')] \models \mathcal{V}^*(s) &&\\
\Longrightarrow\ & Offset(\mathcal{V}^*, s, y^*) = f(\rho')(y^*) = \rho'(a_{y^*}^T \vec{Y}) && = [Offset(\mathcal{V}, s, y_1), Offset(\mathcal{V}, s, y_2), \ldots]a_{y^*}
\end{aligned}$$

Next, observe that if $\mathcal{V}(s)$ increments ρ, we know $\mathcal{V}^*(s)$ increments $f(\rho)$.

$$\begin{aligned}
& [\rho, \rho'] \models \mathcal{V}(s) && \text{where } \rho'(y) = \rho(y) + Offset(\mathcal{V}, s, y)\\
& [f(\rho), f(\rho')] \models \mathcal{V}^*(s) && \text{where } f(\rho')(y^*) = \rho'(a_{y^*}^T \vec{Y}) = [\rho(y_1) + Offset(\mathcal{V}, s, y_1) \ldots]a_{y^*}\\
& [f(\rho), f(\rho')] \models \mathcal{V}^*(s) && \text{where } f(\rho')(y^*) = f(\rho)(y^*) + Offset(\mathcal{V}^*, s, y^*)
\end{aligned}$$

By similar reasoning, if $\mathcal{V}(s)$ resets ρ, then $\mathcal{V}^*(s)$ resets $f(\rho)$.

We show that for every coherence class C^* of $\mathcal{V}^*$, there exists at most one coherence class C of $\mathcal{V}$ such that $proj_{*2} \circ f \circ proj_C$ is non-zero. Assuming this result, we may let w_f be the function mapping C^* to the unique C meeting the above condition if one exists and an arbitrary coherence class otherwise, and observe that $\langle f, w_f \rangle$ is a coherent linear map.

Let C^* be a coherence class of $\mathcal{V}_2$. For a contradiction, suppose that there are distinct coherence classes C_1 and C_2 of $\mathcal{V}$ such that $proj_{C^*} \circ f \circ proj_{C_1}$ and $proj_{C^*} \circ f \circ proj_{C_2}$ are non-zero. Then there is some $\rho_1 \in C_1$ and $\rho_2 \in C_2$ such that $proj_{C^*}(f(\rho_1))$ and $proj_{C^*}(f(\rho_2))$ are non-zero. Without loss of generality, there is some $s \in \Sigma$ such that $\mathcal{V}$ increments ρ_1 and resets ρ_2. It follows that $\mathcal{V}^*(s)$ increments $f(\rho_1)$ and resets $f(\rho_2)$, and thus increments $proj_{C^*}(f(\rho_1))$ and resets $proj_{C^*}(f(\rho_2))$. Since $proj_{C^*}(f(\rho_1))$ and $proj_{C^*}(f(\rho_2))$ both belong to the same coherence class C^*, $\mathcal{V}^*(s)$ must either increment both or reset both. It follows that one of $proj_{C^*}(f(\rho_1))$ and $proj_{C^*}(f(\rho_2))$ must be both reset and incremented, and is therefore zero, contradicting our assumption that $proj_{C^*}(f(\rho_1))$ and $proj_{C^*}(f(\rho_2))$ are non-zero. □

Lemma 5.6. *The category* **Sep** *has pushouts.*

Proof. Let $\langle f, w_f \rangle$ be a linear coherent map from S to S' and $\langle g, w_g \rangle$ be a linear coherent map from S to $S^\dagger$. Define $E \triangleq \left\{ \langle C', C^\dagger \rangle \in D_{S'} \times D_{S^\dagger} : w_f(C') = w_g(C^\dagger) \right\}$. For all $\langle C', C^\dagger \rangle$ in E, let $\left\langle D_{\langle C',C^\dagger \rangle}, a_{\langle C',C^\dagger \rangle}, b_{\langle C',C^\dagger \rangle} \right\rangle$ denote the pushout of $proj_{C'} \circ f$ and $proj_{C^\dagger} \circ g$ in the category of rational vector spaces. Then, define V^* to be the direct product $\prod_{p \in E} D_p$. Let $\chi_p : V^* \to D_p$ be the function projecting elements of V^* onto D_p, and let $\chi_p^* : D_p \to V^*$ be the function sending vectors $d \in D_p$ to the unique vector in V^* such that $\chi_p(\chi_p^*(d)) = d$ and $\chi_{p'}(\chi_p^*(d)) = 0$ for all $p' \neq p \in E$. Define D^* be the set $\left\{ \chi_p^*(D_p) : p \in E \right\}$ and let w_a and w_b be the functions mapping $\chi^*_{\langle C',C^\dagger \rangle}(D_{\langle C',C^\dagger \rangle})$ to C' and $C^\dagger$ respectively. Finally, define:

$$a \triangleq \sum_{\langle C',C^\dagger \rangle \in E} \chi^*_{\langle C',C^\dagger \rangle} \circ a_{\langle C',C^\dagger \rangle} \circ proj_{C'} \qquad b \triangleq \sum_{\langle C',C^\dagger \rangle \in E} \chi^*_{\langle C',C^\dagger \rangle} \circ b_{\langle C',C^\dagger \rangle} \circ proj_{C^\dagger}$$

$$pushout_{\mathbf{Sep}}(\langle f, w_f \rangle, \langle g, w_g \rangle) \triangleq \langle \langle V^*, D^* \rangle, \langle a, w_a \rangle, \langle b, w_b \rangle \rangle$$

By the commutativity of the pushout of the category of rational vector spaces, for all pairs $\langle C', C^\dagger \rangle$ in E, we have that

$$a_{\langle C',C^\dagger \rangle} \circ proj_{C'} \circ f = b_{\langle C',C^\dagger \rangle} \circ proj_{C^\dagger} \circ g$$

It follows that $a \circ f = b \circ g$. Additionally, $w_f(C') = w_g(C^\dagger)$ for all $\langle C', C^\dagger \rangle \in E$ by the definition of E. Thus, $\langle a, w_a \rangle \circ \langle f, w_f \rangle = \langle b, w_b \rangle \circ \langle g, w_g \rangle$.

We now show universality. Suppose $\bar{S}$ is a separated space, $\langle c, w_c \rangle$ is a coherent linear map from S' to $\bar{S}$ and $\langle d, w_d \rangle$ is a coherent linear map from $S^\dagger$ to $\bar{S}$ such that $\langle c, w_c \rangle \circ \langle f, w_f \rangle = \langle d, w_d \rangle \circ \langle g, w_g \rangle$. Then, for all $\bar{C} \in D_{\bar{S}}$, we have that $w_f(w_c(\bar{C})) = w_g(w_d(\bar{C}))$, so $\langle w_c(\bar{C}), w_d(\bar{C}) \rangle \in E$. Additionally:

$$\begin{aligned} c \circ f &= d \circ g && \text{assumption} \\ proj_{\bar{C}} \circ c \circ proj_{w_c(\bar{C})} \circ f &= proj_{\bar{C}} \circ d \circ proj_{w_d(\bar{C})} \circ g && \langle c, w_c \rangle \text{ and } \langle d, w_d \rangle \text{ coherent} \end{aligned}$$

Then, by the universality of the pushout of the category of rational vector spaces on $(proj_{w_c(\bar{C})} \circ f$ and $proj_{w_d(\bar{C})} \circ g$, there must exist a unique $u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle}$ such that $u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \circ a_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} = proj_{\bar{C}} \circ c$ and $u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \circ b_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} = proj_{\bar{C}} \circ d$. Define:

$$u \triangleq \sum_{\bar{C} \in \bar{H}} u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \circ \chi_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle}$$

u is unique by the uniqueness of each $u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle}$ and the definition of χ_p. Let w_u be the function mapping $\bar{C}$ to $\chi^*_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle}(D_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle})$. We must show that $\langle u, w_u \rangle \circ \langle a, w_a \rangle = \langle c, w_c \rangle$ and $\langle u, w_u \rangle \circ \langle b, w_b \rangle = \langle d, w_d \rangle$. We will only show $u \circ a = c$ as $u \circ b = d$ is similar and equivalence of the witnesses is straightforward. For all $p, p' \in E$ such that $p \neq p'$, we have $\chi_p \circ \chi_p^* = 0$, and so:

$$
\begin{aligned}
u \circ a &= \left(\sum_{\bar{C} \in \bar{H}} u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \circ \chi_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \right) \circ \left(\sum_{\langle C', C^\dagger \rangle \in E} \chi^*_{\langle C', C^\dagger \rangle} \circ a_{\langle C', C^\dagger \rangle} \circ proj_{C'} \right) \\
&= \sum_{\bar{C} \in \bar{H}} u_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \circ a_{\langle w_c(\bar{C}), w_d(\bar{C}) \rangle} \circ proj_{w_c(\bar{C})} \\
&= \sum_{\bar{C} \in \bar{H}} proj_{\bar{C}} \circ c \circ proj_{w_c(\bar{C})} = c
\end{aligned}
$$

□

Theorem 5.7. *Consider a transition assignment $\mathit{tf} : \Sigma \to TF(X)$ and a partition Σ_1, Σ_2 of Σ. Let $\langle f_{\Sigma_1}, \mathcal{V}_{\Sigma_1} \rangle$ and $\langle f_{\Sigma_2}, \mathcal{V}_{\Sigma_2} \rangle$ be VASR reflections of $\mathit{tf}|_{\Sigma_1}$ and $\mathit{tf}|_{\Sigma_2}$ respectively, and let*

$$\langle S(\mathcal{V}), \langle a, w_a \rangle, \langle b, w_b \rangle \rangle = pushout_{\mathbf{Sep}}(\langle f_{\Sigma_1}, \tilde{w} \rangle, \langle f_{\Sigma_2}, \tilde{w} \rangle)$$

Then, $\langle a \circ f_{\Sigma_1}, \mathcal{V} \rangle$ is a VASR reflection of tf, where $\mathcal{V}|_{\Sigma_1} = image(\mathcal{V}_{\Sigma_1}, a)$ and $\mathcal{V}|_{\Sigma_2} = image(\mathcal{V}_{\Sigma_2}, b)$.

Proof. Let $\langle f', \mathcal{V}' \rangle$ be a VASR abstraction of tf. Since f' is a simulation from the system of $\mathit{tf}|_{\Sigma_1}$ to $\mathcal{V}|_{\Sigma_1}$, there exists a unique linear simulation $f^*_{\Sigma_1}$ from $\mathcal{V}_{\Sigma_1}$ to $\mathcal{V}'|_{\Sigma_1}$ such that $f' = f^*_{\Sigma_1} \circ f_{\Sigma_1}$ and similarly a unique linear simulation $f^*_{\Sigma_2}$ from $\mathcal{V}_{\Sigma_2}$ to $\mathcal{V}'|_{\Sigma_2}$ such that $f' = f^*_{\Sigma_2} \circ f_{\Sigma_2}$. Then, we have that $\mathcal{V}'|_{\Sigma_1} = image(\mathcal{V}_{\Sigma_1}, f^*_{\Sigma_1})$ and $\mathcal{V}'|_{\Sigma_2} = image(\mathcal{V}_{\Sigma_2}, f^*_{\Sigma_2})$.

Note that if $\langle V, D \rangle$ and $\langle V, D' \rangle$ are separated spaces where D' is a refinement of D, there is a linear coherent map $\langle id, w^\diamond \rangle$ between the spaces where id is the identity function and $w^\diamond$ maps each $C' \in D'$ to the unique $C \in D$ such that $C' \subseteq C$.

By Theorem 5.4, $f^*_{\Sigma_1}$ can be extended to a coherent linear map from $S(\mathcal{V}_{\Sigma_1})$ to $S(\mathcal{V}'|_{\Sigma_1})$. Since $D_{S(\mathcal{V}')}$ is a refinement of $D_{S(\mathcal{V}'|_{\Sigma_1})}$, we can compose this map with some $\langle id, w_1^\diamond \rangle$ to form a linear coherent map $\langle f^*_{\Sigma_1}, w^*_{\Sigma_1} \rangle$ from $S(\mathcal{V}_{\Sigma_1})$ to $S(\mathcal{V}')$. By repeating this reasoning with $f^*_{\Sigma_2}$, we can produce a linear coherent map $\langle f^*_{\Sigma_2}, w^*_{\Sigma_2} \rangle$ from $S(\mathcal{V}_{\Sigma_2})$ to $S(\mathcal{V}')$.

Then, by the universality of $pushout_{\mathbf{Sep}}(\langle f_{\Sigma_1}, w_{f_{\Sigma_1}} \rangle, \langle f_{\Sigma_2}, w_{f_{\Sigma_1}} \rangle)$ there exists a unique $\langle u, w_u \rangle$ such that $\langle f^*_{\Sigma_1}, w^*_{\Sigma_1} \rangle = \langle u, w_u \rangle \circ \langle a, w_a \rangle$ and $\langle f^*_{\Sigma_2}, w^*_{\Sigma_2} \rangle = \langle u, w_u \rangle \circ \langle b, w_b \rangle$.

Finally, the following reasoning shows that $\mathcal{V}' = image(\mathcal{V}, u)$, so u is a simulation from $\mathcal{V}$ to $\mathcal{V}'$ and $\langle a \circ f_{\Sigma_1}, \mathcal{V} \rangle$ is a VASR-reflection:

$$
\begin{aligned}
\mathcal{V}'|_{\Sigma_1} &= image(\mathcal{V}_{\Sigma_1}, f^*_{\Sigma_1}) = image(\mathcal{V}_{\Sigma_1}, u \circ a) = image(image(\mathcal{V}_{\Sigma_1}, a), u) = image(\mathcal{V}|_{\Sigma_1}, u) \\
\mathcal{V}'|_{\Sigma_2} &= image(\mathcal{V}_{\Sigma_2}, f^*_{\Sigma_2}) = image(\mathcal{V}_{\Sigma_2}, u \circ b) = image(image(\mathcal{V}_{\Sigma_2}, b), u) = image(\mathcal{V}|_{\Sigma_1}, u)
\end{aligned}
$$

□

Theorem 4.3. *Let $\mathcal{V}$ be a VASR over variables Y, w be a trajectory over Σ, and n be a $|Y|$-marked abstract trajectory well-formed according to $\mathcal{V}$ such that $w \Vdash n$. For all states ρ, ρ':*

$$[\rho, \rho'] \models Transition(\mathcal{V})[c_{s,i} \mapsto n(s,i)] \iff \rho \xrightarrow{w}_{\mathcal{V}} \rho'$$

Proof. This result is an equivalent formulation of Lemma 6 in [Haase and Halfon 2014]. □

Theorem 7.2. *Consider a program graph M, a transition assignment $\mathit{tf} : \Sigma \to TF(X)$, a procedure summary map $S : P \to TF(X)$, and a function $f : \Sigma \to \mathbb{Q}$. Let $\theta \in UB(M, \mathit{tf}, S, f)$. For any valuations ρ, ρ', procedure p, and nested trajectory $\tau \in \mathcal{T}_M(p)$ such that $\rho \xrightarrow{\tau}_{\mathit{tf}} \rho'$, we have $\hat{f}(\tau) \leq v_\theta(p, \rho)$.*

Proof. Consider any $\theta \in UB(M, tf, S, f)$. We will show that for all valuations ρ, procedures p, and nested trajectories $\tau \in \mathcal{T}_M(p)$, if $e = \rho \dots$ is a trace of $skim(\tau)$ in the transition system defined by tf and S then $v_\theta(p, \rho) \geq v^*_\theta(f, e, skim(\tau))$. With this in hand, we can prove by induction over the structure of τ that for all $\rho, \rho', \tau \in \mathcal{T}_M(p)$ if $\rho \xrightarrow{\tau}_{tf} \rho'$ then there is some execution e of $skim(\tau)$ in the transition system defined by tf and S beginning with ρ such that $v^*_\theta(f, e) \geq \hat{f}(\tau)$. We can finally conclude that $v_a(p, \rho) \geq \hat{f}(\tau)$.

Suppose $e = \rho_1 \dots \rho_{|\tau|+1}$ is a trace of $skim(\tau)$ in the transition system defined by tf and S. Let NV denote the set of new variables $D \cup \{counter, flag\}$. Let $\bar{\rho}_1$ be the extension of ρ_1 to include variables NV initialized to 0 and for all $i \in [|\tau|]$ let $\bar{\rho}_{i+1}$ be defined by $\bar{\rho}_{i+1}|_X = \rho_{i+1}$ and:

- if $skim(\tau)_i \in \Sigma$, $\bar{\rho}_{i+1}|_{NV} = \bar{\rho}_i|_{NV}[counter \mapsto \bar{\rho}_i(counter) + f(skim(\tau)_i)]$
- if $skim(\tau)_i \in P$ and $0 < \rho_i(\theta(p))$, $\bar{\rho}_{i+1}|_{NV} = \bar{\rho}_i|_{NV}[d_p \mapsto \bar{\rho}_i(d_p) + \bar{\rho}_i(\vec{X}), flag \mapsto 1]$
- if $skim(\tau)_i \in P$ and $0 \geq \rho_i(\theta(p))$, $\bar{\rho}_{i+1}|_{NV} = \bar{\rho}_i|_{NV}[d_p \mapsto \bar{\rho}_i(d_p), flag \mapsto 1]$

Let $\bar{\rho} = \bar{\rho}_1$ and $\bar{\rho}' = \bar{\rho}_{|\tau|+1}$. It is clear from the casework above $\sum_{p \in P} \vec{a}^T_p \bar{\rho}'(\delta_p) + \rho'(counter) = v^*_a(f, e, skim(\tau))$.

If $(\bar{\rho}'(counter) \leq 0 \wedge \bar{\rho}'(flag) = 0)$, then $v^*_\theta(f, e, skim(\tau)) \leq 0$ as $flag$ being 0 implies $v^*_\theta(f, e, skim(\tau)) = \rho'(counter)$, and so $v^*_\theta(f, e, skim(\tau)) \leq 0 \leq v_\theta(p, \rho)$ by the definition of our template. Otherwise by the definition of $Intra$, $[\bar{\rho}, \bar{\rho}'] \models Intra(M_p, tf_\uparrow) \wedge Ctxt_\uparrow$. Then, by the definition of $UB(M, tf, S, f)$, $v^*_\theta(f, e) \leq \rho(\theta(x)) \leq v_\theta(p, \rho)$. Thus, we have shown that v_θ is an inductive upper potential and thus the theorem statement is true. □

Lemma 7.4. *Let P be a set of procedure identifiers and let $tf : \Sigma \to TF(X)$ be a transition assignment. Consider any convex polyhedron $UB \subseteq P \to Lin(X)$. For any valuation ρ, we have $\rho \models B_\uparrow(X, UB, p)$ if and only if $\rho(\xi) \leq v_\theta(p, \rho)$ for all $\theta \in UB$.*

Proof. We will prove both directions of the proof simultaneously via a series of equivalences. Showing that $\rho(\xi) \leq v_\theta(p, \rho)$ for all θ is equivalent to showing that $\rho(\xi) \leq \min_{\theta \in UB} v_\theta(p, \rho)$. Since $\{v_1, \dots, v_n\}$ and $\{r_1, \dots, r_m\}$ is a generator representation of UB:

$$\rho(\xi) \leq \rho\left(\min_{\lambda,\alpha} \max(0, (\sum_{i \in [n]} \lambda_i v_i + \sum_{i \in [m]} \alpha_i r_i)(p))\right) \qquad s.t.\ \alpha \geq 0, \lambda \geq 0, \sum \lambda_i = 1$$

Proceed by cases. If there are any r_i such that $\rho(r_i(p)) < 0$, the term $\rho((\sum_{i \in [n]} \lambda_i v_i + \sum_{i \in [m]} \alpha_i r_i)(p)$ can be negative by making α_i large. Then, the above is equivalent to $\rho(\xi) \leq 0$.

If $\rho(r_i(p)) \geq 0$ for all r_i, a minimum must exist where all $\alpha_i = 0$, so the above is equivalent to $\xi \leq \min_\lambda \max(0, \sum_{i \in [n]} (\lambda_i v_i)(p))$ subject to $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$. This objective is a convex function, so its minimum must occur at one of the v_i.

$B_\uparrow(tf, UB, p)$ is a straightforward translation of this casework to logic and is thus equivalent. □

Lemma 7.3 (Anti-monotonicity). *For any two transition assignments $tf, tf' : \Sigma \to TF(X)$ and two summary assignments $S, S' : P \to TF(X)$, if $tf(s) \models tf'(s)$ for all $s \in \Sigma$ and $S(p) \models S'(p)$ for all $p \in P$, then $UB(M, tf', S', f) \subseteq UB(M, tf, S, f)$.*

Proof. Under the assumptions of the lemma statement, $tf_\uparrow(s) \models tf'_\uparrow(s)$ for all s in $\Sigma \cup P$. Then, the claim follows from the assumed monotonicity of $Intra$ and the anti-monotonicity of the $Ineq$ procedure described in Section 3. □

Theorem 8.1 (Monotonicity). *For any transition assignments tf and tf' such that $tf(s) \models tf'(s)$ for all symbols s, program graph M, and procedure p,*

$$Summary(M, tf, p) \models Summary(M, tf', p)$$

Proof. Suppose that $\langle f, \mathcal{V} \rangle$ is a VASR reflection of tf over Y, and $\langle f', \mathcal{V}' \rangle$ is a VASR reflection of tf' over Y^*.

Since $tf(s) \models tf'(s)$ for all s, we have that f' is a linear simulation from tf to $\mathcal{V}'$. Since $\langle f, \mathcal{V} \rangle$ is a VASR reflection of tf, there is a unique linear simulation $f^\dagger$ from $\mathcal{V}$ to $\mathcal{V}'$ such that $f^\dagger \circ f = f'$. Since $f^\dagger$ is a simulation, we have that for for all ρ, ρ' such that $\rho \xrightarrow{\mathcal{L}(\mathcal{G}(M,p))}_{\mathcal{V}} \rho'$, we have $f^\dagger(\rho) \xrightarrow{\mathcal{L}(\mathcal{G}(M,p))}_{\mathcal{V}'} f^\dagger(\rho')$. Let $(f^\dagger)^*$ be the substition mapping $y^* \in Y^*$ to $t_{y^*} \in Lin(Y)$ where $f^\dagger(\rho)(y^*) = \rho(t_{y^*})$, and let $(f')^*$ be the substituion mapping $y^* \in Y^*$ to $t_{y^*} \in Lin(X)$ where $f'(\rho)(y^*) = \rho(t_{y^*})$. It follows that:

$$\begin{aligned} Reach(\mathcal{V}, \mathcal{G}(M,p)) &\models Reach(\mathcal{V}', \mathcal{G}(M))[Y^* \mapsto (f^\dagger)^*(Y)] \\ Reach(\mathcal{V}, \mathcal{G}(M))[Y \mapsto f^*(X)] &\models Reach(\mathcal{V}', \mathcal{G}(M))[Y^* \mapsto (f^\dagger)^* f^*(X)] \\ Reach(\mathcal{V}, \mathcal{G}(M))[Y \mapsto f^*(X)] &\models Reach(\mathcal{V}', \mathcal{G}(M))[Y^* \mapsto (f')^*(X)] \end{aligned}$$

So by the definition of *Summary* and Lemmas A.2 and 7.3, we have

$$Summary(M, tf) \models Summary(M, tf') . \quad \square$$

A.2 Inductive Lower Potentials

This procedure for generating potentials can be slightly modified to produce and apply lower bounds to $\hat{f}$. This section will present the modifications tersely. v is an **inductive lower potential** for f if for all valuations ρ and procedures p, if $e = \rho \ldots$ is a trace of $skim(\tau)$ in the transition system defined by tf and S with $\tau \in \mathcal{T}_M(p)$, then $v(p, \rho) \leq v^*(f, e, skim(\tau))$. Our definition of $tf_\downarrow$ is as follows:

$$\begin{aligned} tf_\downarrow(s) &\triangleq tf(s) \wedge same(D \cup \{flag\}) \wedge counter' = counter + f(s) \\ tf_\downarrow(p) &\triangleq same(L \cup D \backslash \{D_p\}) \wedge \vec{D_p}' = \vec{D_p} + \vec{X} \wedge counter' = counter + f(p) \end{aligned}$$

We define our formula to codify our search conditions:

$$F \triangleq Intra(\langle V, E \rangle, tf_\uparrow, in(p), out(p)) \wedge \left(\bigwedge_{v \in \{counter, flag\} \cup D} v = 0 \right)$$

Our set of inductive lower potentials $LB(M, tf, S, f)$ is $\bigcap_{p \in P} LB_p$ where:

$$UB_p \triangleq \left\{ \theta : P \to Lin(X) : F \models \theta(p) \leq counter + \sum_{p \in P} \theta(p)[X \mapsto D_p] \right\}$$

Theorem A.1. *Consider a program graph M, a transition formula mapping $tf : \Sigma \to TF(X)$, a procedure summary map $S : P \to TF(X)$, and a function $f : \Sigma \to \mathbb{Q}$. Let $\theta \in UB(M, tf, S, f)$. For any valuations ρ, ρ', procedure p, and nested trajectory $\tau \in \mathcal{T}_M(p)$ such that $\rho \xrightarrow{\tau}_{tf} \rho'$, we have*

$$\hat{f}(\tau) \geq v_\theta(p, \rho)$$

Lemma A.2. *For any two transition formula mappings $tf, tf' : \Sigma \to TF(X)$ and two summary assignments $S, S' : P \to TF(X)$, if $tf(s) \models tf'(s)$ for all $s \in \Sigma$ and $S(p) \models S'(p)$ for all $p \in P$, then $LB(M, tf', S', f) \subseteq LB(M, tf, S, f)$.*

Finally, the following is the encoding of a convex polyhedron LB into a formula $B_\downarrow(\mathit{tf}, LB, p)$ applying the bounds corresponding to p to a free variables ξ. Let $\{v_1 \dots v_n\}$ and $\{r_1 \dots r_m\}$ be the generator representation of LB projected onto the component a_p.

$$\mathcal{B}_\downarrow \triangleq \bigwedge_{i \in [m]} r_i^T x \leq 0 \wedge \bigwedge_{i \in [n]} \xi \geq \max(0, v_i^T x)$$

Lemma A.3. *Let P be a set of procedure identifiers and let $\mathit{tf} : \Sigma \to T(X_L \cup X_G)$ be a transition formula mapping. Consider any convex polyhedron $LB \subseteq P \to Lin(X)$. For any valuation ρ, we have $\rho \models B_\downarrow(\mathit{tf}, LB, p)$ if and only if $\rho(\xi) \geq v_\theta(p, \rho)$ for all $\theta \in LB$.*

A.3 Pushout of Ordered Rational Vector Spaces

Recall that **Sep** refers to the category in which the objects are separated spaces and the arrows are coherent linear maps, and that $\mathbf{Sep}^{\leq}$ refers to the category in which the objected are ordered separated spaces and the arrows are positive coherent linear maps. The pushout of $\mathbf{Sep}^{\leq}$ follows the same approach as the pushout of **Sep**, with the pushout of ordered rational vector spaces in place of the pushout of rational vector spaces. This section shows that the category of ordered rational vector spaces has pushouts.

Specifically, given two positive linear maps $f : A \to B$ and $g : A \to C$ between ordered vector spaces, the pushout in the category of ordered rational vector spaces is an ordered vector space D and positive linear maps $p : B \to D$ and $q : C \to D$ such that:

(1) $p \circ f = q \circ g$
(2) For any other ordered vector space D' and positive linear maps $p' : B \to D'$ and $q' : C \to D'$ such that $p' \circ f = q' \circ g$, there exists a positive linear map $u : D \to D'$ such that $u \circ p = p'$ and $u \circ q = q'$

The positive cone of an ordered vector space V is the set $\{v \in V : v \geq_V 0\}$. The pushout is constructed as follows. Suppose that $b_1, \dots, b_n$ generate the positive cone of B and $c_1, \dots, c_m$ generate the positive cone of C. Then, consider the following set:

$$\left\{ \langle p, q \rangle \in (B \to_{lin} \mathbb{Q}) \times (C \to_{lin} \mathbb{Q}) : p \circ f = q \circ g \wedge \bigwedge_{i=1}^{n} p(b_i) \geq 0 \wedge \bigwedge_{j=1}^{m} q(c_j) \geq 0 \right\}$$

This set is a convex cone. Let $\langle p_1, q_1 \rangle \dots p_k, q_k$ be the rays of this cone. Then the pushout is $\left\langle \mathbb{Q}^k, p, q \right\rangle$ where the order on $\mathbb{Q}$ is defined as

$$\vec{u} \leq_{\mathbb{Q}^r} \vec{v} \iff u_i \leq v_i \text{ for all } i$$

and where p and q are defined as:

$$p(v') \triangleq \langle p_1(v'), \dots, p_k(v') \rangle$$
$$q(v^*) \triangleq \langle q_1(v^*), \dots, q_k(v^*) \rangle$$