

**From the holdings of the Department of Special Collections , Princeton University Library,
Princeton, NJ**

These documents are either in the public domain, or can only be used for educational and research purposes (Fair use) as per U.S. Copyright law (text below). By accessing this file, all users agree that their use falls in the public domain or within fair use as defined by the copyright law. We do not charge any permission or usage fees for the publication of images of material in our collections, including those provided by Princeton University Library, via our website, catalog, or directly from staff. For more information regarding copyright, credit, and citation please review the information on the Copyright, Credit and Citation Guidelines page of our website.

<https://library.princeton.edu/special-collections/policies/copyright-credit-and-citation-guidelines>. If you have any questions, including how to order higher-quality images from the collection, contact Special Collections Public Services via our Ask Us! Form. <https://library.princeton.edu/special-collections/ask-us>

Master's Theses, 2022-2026

Computer Science, 2023

2023

Counter-Based Summarization of Recursive Procedures

Pimpalkhare, Nikhil

Princeton, NJ : Princeton University

<http://arks.princeton.edu/ark:/88435/dsp01cc08hj91v>

U.S. Copyright law: The copyright law of the United States (Title 17, §108, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of fair use that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve a violation of copyright law. <https://www.copyright.gov/title17/title17.pdf>

COUNTER-BASED SUMMARIZATION OF RECURSIVE PROCEDURES

NIKHIL PIMPALKHARE

ADVISOR: ZACHARY KINCAID

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE
PRINCETON UNIVERSITY

MAY 2023

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Nikhil Pimpalkhare

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Nikhil Pimpalkhare

Abstract

The work of this thesis lies within the field of program analysis, which seeks to provide formal reasoning about the behavior of programs without ever executing them. The approach that we use for program analysis involves computing a logical summary of an input program that narrowly over-approximates its behavior, followed by automated reasoning algorithms to prove statements about program behavior. This thesis presents a compositional and monotone approach to the summarization of procedures with potentially multiple recursive calls, a particularly challenging domain. We summarize the dynamics of programs using vector addition systems with resets (VASR), a model that is simultaneously strong enough to capture interesting numerical behavior and weak enough for its reachability relation to be computable. We show how to compute a VASR that is a best abstraction of an input program, how to compute the reachability of a VASR-weighted pushdown system, and how to combine these algorithms to produce summaries of programs. Together, these techniques combine for a summarization procedure that is performant, is capable of proving useful assertions in real-world programs, and makes theoretical guarantees about its relative performance on related programs. Our work offers a valuable contribution to the field of program analysis and paves the way for future advancements in verified software.

Acknowledgements

I would like to sincerely thank Professor Zachary Kincaid for his unwavering support throughout the past two years. Zak has been incredibly influential in shaping my research interests, formalizing my mathematics, and inspiring me that there's no such thing as being too abstract. This thesis would not have been possible without his mentorship, and I look forward to continuing to work with him during my PhD.

To my friends, thank you for always being there for me. Our jam sessions, group dinners, and last-minute coffee runs have filled every day with a little bit of joy. Perhaps we've been having too much fun...

Finally, I am eternally grateful for my family for so many years of love and support. Your belief in me means the world.

To a few more years in Princeton

Contents

Abstract	iii
Acknowledgements	iv
1 Introduction	1
1.0.1 Overview	3
2 Preliminaries	9
2.0.1 Notation/Mathematical Background	9
2.0.2 Transition Systems	11
2.0.3 Language Models	11
3 Best VASR Abstractions of Programs	14
3.0.1 Best Per-Edge Transformations	15
3.0.2 The Sep Category	17
3.0.3 Combining VASRs	22
4 Reachability of VASRS along Context-Free Program Paths	25
4.0.1 Interval Grammars	28
4.0.2 End-to-end Summarization Procedure	30
5 Evaluation	32
5.0.1 RQ1: Efficacy of Context-Free Reachability	32
5.0.2 RQ2: Comparison with State of the Art	35

6	Discussion	37
6.0.1	Related Work	37
6.0.2	Limitations and Future Work	39
6.0.3	Conclusion	40

Chapter 1

Introduction

Our society runs on software, but writing reliable and bug-free software is hard. In recent years, software developers have increasingly turned to static analysis and verification tools to make provable guarantees about the behavior of their software. However, the problems that these tools need to solve lie at the edge of uncomputability. Even trivial questions, such as "does this program terminate?" were proven to be unanswerable nearly 100 years ago [24].

Many state-of-the-art approaches utilize heuristic techniques to prove assertions about program behavior, but such tools remain highly unsatisfactory. Developers may receive dramatically different results with slight changes to their input program. In the worst case, developers may even resist making useful edits to their programs because it interferes with their program analyzer's ability to prove correctness. This discrepancy between developer expectations and analysis tools is an obstacle to full adoption of formally verified software. A good program analyzer should be powerful enough to prove assertions that developers deem useful, predictable in its analysis of related programs, and performant enough to ensure that its incorporation into a software engineering workflow is not burdensome.

In this thesis, we present a series of algorithms that combine to form an analysis

method that meets all three conditions above for a particularly challenging subclass of programs: those with nonlinear recursion. Recursion is an intuitive and widely used programming methodology, but program analyzers often turn to imperfect heuristics to reason about programs exhibiting complex recursive structure. While these heuristics often enable verification tools to prove assertions, it is difficult for end-users to predict how the code changes that they make will affect the performance of their analysis tool due to “butterfly effects” in which a slight change to the input programs leads to a significantly different analysis. Insignificant changes, such as adding a redundant condition to an if statement or eliminating an unused variable, can drastically increase the verification time of program analyzers and can even cause them to fail. If programmers cannot understand how changing their code affects the efficacy of the resulting analysis, how can we convince them to adopt program analyzers into their workflow?

One theoretical condition for programs summaries that guarantees a degree of predictability for program analyses is *monotonicity*: if a program A’s behavior is a subset of program B’s, program A’s summary should be at least as specific as program B. Many existing monotone analyses follow the methodologies of algebraic program analyses [14], which compute procedure summaries by interpreting a regular expression representing the set of paths through a program under some algebraic structure. However, this approach faces serious challenges with regards to recursive programs because it is typically impossible to compute a regular expression representing the paths of interest through recursive procedures. For example, even the simple $f(): \text{if}(\ast) \ A; f(); \ B$ has a set of paths $(\{A^i B^i \mid i \geq 0\})$ that cannot be represented by any regular expression.

In this paper, we produce procedure summaries by interpreting a **context-free language** representing the paths through a program using rational vector addition systems with resets (VASRs). By using a context free language, we can fully generate

the paths through recursive procedures, as the language of pushdown automata is context free. Classically used as a model for parallel systems [11], VASRs are capable of describing a wide variety of numerical program behaviors, while still being weak enough for us to reason about their reachability when constrained to represent paths through recursive procedures. The end-to-end summarization procedure of this paper is *monotone*, *compositional*, and performs comparably to state-of-the-art heuristic tools on academic and industrial benchmarks.

In summary, the contributions of this thesis are:

1. A mechanism for synthesizing the *best* Vector Addition System with Resets abstraction of a program
2. A novel encoding for the reachability relation of a Vector Addition System with Resets along paths through recursive procedures
3. An open-source and robust implementation, shown in our evaluation section to be comparable with the state of the art.

1.0.1 Overview

The techniques outlined in this thesis describe a method for procedure summarization. This section aims to give the reader intuition about the models that are used to represent programs and how the chapters fit together to produce summaries. The concrete details of each algorithmic step can be found in subsequent chapters.

The input to our summarization method is a *program graph* representing the structure of a program and *transition formula mapping* corresponding each edge to a *transition formula* representing a transformation on program variables. A transition formula is a logical formula over a set of program variables X and primed copies X' which holds if X can transition to X' .

```

1 char lexer(char* s, int slen, int c) {
2   if (c > slen) {return EOF;}
3   if (whitespace(s[c])) {
4     end += 1;
5     hd = end;
6     //save word s[hd:end]
7   } else {
8     end += 1;
9   }
10  lexer(s, slen, c + 1);
11 }
12
13 void main() {
14   lexer(NULL, nondet_int());
15   assert(hd <= end);
16 }

```

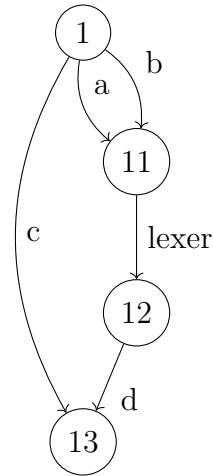


Figure 1.1: Lexer

For example, consider the code example describing the overall structure of a `lexer` function and the adjacent program graph representing its structure in Figure 1.0.1. A lexer aims to extract tokens from a buffer of characters, provided to it as a pointer s and length $slen$. It recursively iterates through the buffer and attempts to fit each token between the hd and end pointers; whenever it finds whitespace, it saves the current token (written abstractly as a comment) then moves hd to end to begin the next word. The vertices of the graph represent control locations within the program - it can be viewed as a simplified version of the control flow graph of the program. Vertex 1 corresponds to the beginning of line 2; likewise, vertex 2 corresponds with line 11, vertex 3 with line 12, and vertex 4 with line 13. The corresponding transition formula mapping is as follows:

$$\begin{aligned}
\mathcal{F}(a) &= \begin{pmatrix} slen > 0 \\ \wedge hd' = end' \\ \wedge end' = end + 1 \\ \wedge slen' = slen - 1 \end{pmatrix} & \mathcal{F}(b) &= \begin{pmatrix} slen > 0 \\ \wedge hd' = hd \\ \wedge end' = end + 1 \\ \wedge slen' = slen - 1 \end{pmatrix} \\
\mathcal{F}(c) &= \begin{pmatrix} slen \leq 0 \\ \wedge hd' = hd \\ \wedge end' = end \\ \wedge slen' = slen \end{pmatrix} & \mathcal{F}(d) &= \begin{pmatrix} hd' = hd \\ \wedge end' = end \\ \wedge slen' = slen + 1 \end{pmatrix}
\end{aligned}$$

The formula associated with each edge represents the program transformation associated with a program execution between the respective control locations. In the above example, a represents taking the else branch of the if statement on line 2 and taking the then branch of the if statement on line 3, b represents taking the else branch of the if statement on line 2 and taking to else case of the if statement on line 3, c represents taking the then branch of the if statement at line 2 and returning, and d represents line 12.

The edge from vertex 2 to vertex 3 is labeled by *lexer* - this represents the recursive call made at line 11. The source and end corresponding to this call are vertices 1 and 14 respectively. We define paths through our program inductively. Paths of depth 0 are paths from vertex 1 to 4 which use no recursive calls. Paths of depth $i + 1$ are paths from 1 through 4 in which every occurrence of a recursive call is replaced with a depth i path from the source of the call to its end. The union over all i is the set of all paths through our program.

At a high level, the logical summary that we produce must over-approximate the composition of transition formulas along any path through our program. However, as this model is Turing Complete, this problem is undecidable. Instead, we compute an over-approximate abstraction of our input program which we are able to approximate

the dynamics along paths through our program: *rational vector addition systems with resets*.

The transformations of a vector addition system with resets (VASR) operate on a set of counters, each containing a rational number value¹. Each transformation affects each counter independently: it either resets the counter to 0 or leaves the value in the counter unchanged, then adds some rational constant to the counter.

We ensure that there is a relationship between the transition formula mapping and our VASR mapping by mandating that there is a linear relationship between the global variables of the transition formula and the counters of the VASR. Specifically, we require that there is a linear function f mapping variable valuations to counter valuations such that if variable valuation x transitions to variable valuation x' along the transition formula associated with some edge $\#(e)$, then the counter valuation $f(x)$ transitions to counter valuation $f(x')$ along the VASR transformation associated with edge e . This allows us to treat the composition of VASR transformations along a program path as an overapproximation for the composition of the respective transition formulas.

Continuing with the above lexer example, the following VASR meets the conditions above and can be used to overapproximate the behavior of l . For notation, we will write the state of the counters as a d -dimensional vector u and describe the transformation corresponding to edge e as a reset vector $\phi_R(e) \in \{0, 1\}^d$ and an offset vector in $\phi_+(e) \in \mathbb{Q}^d$. The transformation associated with the transform can be then understood as $u' = \phi_R(e) * u + \phi_+(e)$, where $*$ is the elementwise product.

¹Vector Addition Systems with Resets are typically defined over vectors of natural numbers. Our choice of rational numbers allows us to better represent the dynamics of programs and reduces the complexity of the resulting reachability problem, as the complexity of computing VASR reachability using natural numbers is non-elementary [15].

Relationship between variables and counters: $u = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} hd \\ end \\ height \end{bmatrix}$

$$\langle \phi_R(a), \phi_+(a) \rangle = \left\langle \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \right\rangle \quad \langle \phi_R(b), \phi_+(b) \rangle = \left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \right\rangle$$

$$\langle \phi_R(c), \phi_+(c) \rangle = \left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\rangle \quad \langle \phi_R(d), \phi_+(d) \rangle = \left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \right\rangle$$

For example, the VASR transformation associated with edge a above can be understood as:

$$\begin{bmatrix} hd' - end' \\ end' \\ height' \end{bmatrix} = \phi_R(a) * \begin{bmatrix} hd - end \\ end \\ height \end{bmatrix} + \phi_+(a) = \begin{bmatrix} 0 \\ end + 1 \\ height + 1 \end{bmatrix}$$

It can easily be verified that for all edges $l(e)$ implies $u' = \phi_R(e) * u + \phi_+(e)$.

The remainder of this thesis can be understood as follows:

- Chapter 2: A formal description of the models presented in this overview and the mathematical background relevant to the other sections.
- Chapter 3: Given a transition formula mapping, how can we compute its *best* VASR abstraction?
- Chapter 4: Given a VASR mapping over the edges of a program graph with recursive calls, how can we encode all compositions of VASR transformations

along paths through the program into logic?

- Chapter 5: A description of the implementation of the methods described in this thesis and an evaluation against state of the art program analyzers on standardized benchmarks.
- Chapter 6: Concluding remarks and a discussion of future extensions.

Chapter 2

Preliminaries

2.0.1 Notation/Mathematical Background

Let $u * v$ denote the Hadamard (elementwise) product between u and v ; $(u * v)_i = u_i v_i$. Let e_i be the vector such that $(e_i)_i = 1$ and for all $j \neq i$ $(e_i)_j = 0$. Let $[n]$ denote the set $\{1, \dots, n\}$. Given a set Σ , a valuation function mapping elements of the set to rational values $\rho : \Sigma \rightarrow \mathbb{Q}$ and a finite vector $\mathbf{x} = [x_1, \dots, x_n]^T \in \Sigma^n$, we define $\rho(\mathbf{x})$ to be a vector of ρ 's valuations over $\mathbf{x}$: $\rho(\mathbf{x})_i = \rho(x_i)$. For any boolean b , we define $\mathbf{1}[b] = 1$ if $b = \text{true}$ and $\mathbf{1}[b] = 0$ if $b = \text{false}$.

A **category** is a mathematical structure composed of objects, morphisms between those objects, and a composition operator. A morphism is a binary relation $\rightarrow$ with a domain object on the left and a range object on the right; we write a morphism from object A to object B as $f : A \rightarrow B$. The composition operator $\circ$ combines two morphisms into a single one as long as the range of one is the domain of the other. For example, given two morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$, $f \circ g : A \rightarrow C$ is the resulting composed morphism. The composition operator is associative and has an identity. The set of all morphisms in a category is closed under composition.

The **pushout** of two morphisms with a common domain object $f : A \rightarrow B$ and $g : A \rightarrow C$ is a triple $\langle D, a, b \rangle$ consisting of an object Z and two morphisms $a : B \rightarrow D$

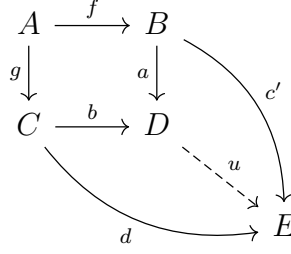


Figure 2.1: Pushout Diagram

and $b : C \rightarrow D$.

- Commutativity: $a \circ f = b \circ g$
- Universality: For all other objects E and morphisms $c : B \rightarrow E$ and $d : C \rightarrow E$ such that $c \circ f = d \circ g$, there exists a unique morphism $u : D \rightarrow E$ such that $u \circ a = c$ and $u \circ b = d$.

The pushout can be thought of as the most general way to find a common object between B and C which completes the commutative square.

An example category is $\mathbf{Vect}_{\mathbb{Q}}$, in which the objects are linear subspaces of $\mathbb{Q}^d$ for any $d \in \mathbb{N}$ and the morphisms are linear transformations. For any pair of matrices $A \in \mathbb{Q}^{n \times m}$ and $B \in \mathbb{Q}^{k \times m}$, let $pushout_{Vect}(A, B)$ denote the pushout of A and B —that is, a triple (d, C, D) with $C \in \mathbb{Q}^{d \times n}$ and $D \in \mathbb{Q}^{d \times k}$ satisfying the above conditions. We can compute this pushout efficiently by computing the basis of the following linear set via the algorithm in [21], as the set contains all possible rows of linear transformations from C and D to a common object and making the square commute.

$$\{(c, d) : cA = dB\}$$

The basis of this set satisfies commutativity by definition, and satisfies universality by the properties of a basis.

2.0.2 Transition Systems

A **labeled transition system** over a finite alphabet Σ is a pair $T = \langle S_T, \rightarrow_T \rangle$ where S_T is a (possibly infinite) set of states, and $\rightarrow_T \subseteq S_T \times \Sigma \times S_T$ is a labeled transition relation. We write $v \xrightarrow{s}_T v'$ to denote that $\langle v, s, v' \rangle$ belongs to the transition relation $\rightarrow_T$. For a word $w \in \Sigma^*$, we use $v \xrightarrow{w}_T v'$ to denote that there exists a finite sequence $v = v_0 \xrightarrow{s_1}_T v_1 \xrightarrow{s_2}_T \dots \xrightarrow{s_n}_T v_n = v'$ such that $s_1 s_2 \dots s_n = w$. For a language $L \subseteq \Sigma^*$, we use $v \xrightarrow{L}_T v'$ to denote that there is some word $w \in L$ such that $v \xrightarrow{w}_T v'$.

Let T, U be labeled transition systems over an alphabet Σ . A **linear simulation** is linear map $f : S_T \rightarrow S_U$ such that for all $s \in \Sigma$, for any $v \xrightarrow{s}_T v'$, we have $f(v) \xrightarrow{s}_U f(v')$. The existence of such a linear simulation implies that U is an overapproximate abstraction of T , as $\{u, v : u \rightarrow_T v\} \subseteq \{u, v : f(u) \rightarrow_U f(v)\}$; in other words, if we are able to compute the reachability of U , we have an overapproximation of the reachability of T .

A d -dimensional **labeled rational vector addition system with resets** (VASR) is a pair $\mathcal{V} = \langle \phi_R : \Sigma \rightarrow \{0, 1\}^d, \phi_+ : \Sigma \rightarrow \mathbb{Q}^d \rangle$ defining a labeled transition system $\langle \mathbb{Q}^d, \rightarrow_{\mathcal{V}} \rangle$ such that for each $s \in \Sigma$, we have $v \xrightarrow{s}_{\mathcal{V}} v'$ if and only if $v' = \phi_R(s) * v + \phi_+(s)$.

Given a set of variables X , a **valuation** is a map $\rho : X \rightarrow \mathbb{Q}$ between variables and their rational values. A **transition formula** in $TF(X)$ is a formula F in Linear Integer/Real Arithmetic whose free variables range over variables X and primed copies X' . Given two valuations over X , $\rho, \rho' \in X \rightarrow \mathbb{Q}$, we write $[\rho, \rho'] \models F$ if F holds when every occurrence of $x \in X$ and $x' \in X$ are replaced with $\rho(x)$ and $\rho'(x)$ respectively. A function $\# : \Sigma \rightarrow TF(X)$ defines a labeled transition system $\langle X \rightarrow \mathbb{Q}, \rightarrow_{\#} \rangle$ where $\rho \xrightarrow{s}_{\#} \rho'$ if and only if $[\rho, \rho'] \models \#(s)$.

2.0.3 Language Models

A **program graph** $M = \langle V, \Sigma, P, E, in, out, p_0 \rangle$ 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 $E \subseteq V \times (\Sigma \cup P) \times V$, two functions $in, out : P \mapsto V$ mapping each procedure to its input and output vertex respectively, and an initial procedure $p_0 \in P$.

For all $p \in P$, let the set of **executions** of p of depth 0, $\mathcal{E}_M(p, 0)$, be the set of all finite sequences $e \in \Sigma^*$ such that there exists a sequence of vertices $v_0 \dots v_n$ such that $v_0 = in(p)$, $v_n = out(p)$, and $(v_{i-1}, e_i, v_i) \in E$ for all $i \in [n]$. For all $j \in \mathbb{N}$, let the set of executions of p of depth $j+1$, $\mathcal{E}_M(p, j+1)$, be the set of all finite sequences $e \in (\Sigma \cup (P \times \bigcup_{q \in P} \mathcal{E}_M(q, j)))^*$ such that there exists a sequence of vertices $v_0 \dots v_n$ such that $v_0 = in(p)$, $v_n = out(p)$, and for all $i \in [n]$, either

- $e_i \in \Sigma$, $(v_{i-1}, e_i, v_i) \in E$, the *standard edges* of an execution
- $e_i = \langle p, e' \rangle \in (P \times \bigcup_{q \in P} \mathcal{E}_M(q, j))$, $(v_{i-1}, p, v_i) \in E$, and $e' \in \mathcal{E}_M(p, j)$, the *procedure calls* of an execution

Let $\chi : \bigcup_{q \in P} \bigcup_j \mathcal{E}_M(q, j) \rightarrow \Sigma^*$ be a flattening function which computes the string of edges corresponding to an execution. Formally, χ is the homomorphism that sends $e \in \Sigma$ to e , and $(p, e') \in P \times (\bigcup_j \mathcal{E}_M(p, j))$ to $\chi(e')$. Then, the language of paths through a procedure p is defined as

$$\mathcal{L}_M(p) = \{\chi(e) : \exists i \in \mathbb{N}, (e) \in \mathcal{E}_M(p, i)\}$$

. The language of a program $\mathcal{L}(M)$ is the language of its initial procedure $\mathcal{L}_M(p_0)$.

A **context-free grammar** $G = \langle N, \Sigma, R, s_0 \rangle$ 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 : $w_1 \alpha w_2 \rightarrow w_1 \beta w_2$. The language corresponding to a nonterminal $\mathcal{L}_G(n)$ is the set $\{w \in \Sigma^* : n \rightarrow^* w\}$. 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 $n \Rightarrow s$, $n \Rightarrow nm$, or $n_0 \Rightarrow \epsilon$ where $n, m \in N$ and $s \in \Sigma$. There is a polynomial time procedure to convert any context free grammar into a grammar in Chomsky Normal Form with an equal language [2].

Given a program M , we can generate the following grammar $C(M)$ such that $\mathcal{L}(M) = \mathcal{L}(C(M))$.

$$C(M = \langle V, \Sigma, P, E, in, out, p_o \rangle) \triangleq \langle V \times V, \Sigma, P_{C(M)}, (in(p_o), out(p_o)) \rangle$$

$$\begin{aligned} P_{C(M)} &\triangleq \cup \{ \langle v_i, v_j \rangle \Rightarrow s \langle v_k, v_j \rangle : \langle v_i, s, v_k \rangle \in E, s \in \Sigma, v_j \in V \} \\ &\cup \{ \langle v_i, v_i \rangle \Rightarrow \epsilon : v_i \in V \} \\ &\cup \{ \langle v_i, v_j \rangle \Rightarrow \langle in(p), out(p) \rangle \langle v_k, v_j \rangle : \langle v_i, p, v_k \rangle \in E, p \in P \} \end{aligned}$$

The **Parikh image** [19] of a word $w \in \Sigma^*$ is a function $\pi(w) : \Sigma \rightarrow \mathbb{N}$ 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 $\pi(L) \triangleq \{ \pi(w) : w \in L \}$.

Given a context-free grammar $G = \langle N, \Sigma, R, n_0 \rangle$, we can compute a formula, $\psi_{Parikh}(G)$, representing the Parikh image of $\mathcal{L}(G)$ in the following way [25]. Its free variables range over a designated set of symbols $\{c_s : s \in N \cup \Sigma\}$. For all valuations $m : \{c_s : s \in N \cup \Sigma\} \rightarrow \mathbb{Z}$, we have $m \models \psi_{Parikh}(G)$ if and only if m restricted to Σ is the Parikh image of some word $w \in \mathcal{L}(G)$ and for all nonterminals n , $m(n)$ counts the number of production rules with left-hand side n in the derivation of w . There is a polynomial time procedure to compute $\phi_{Parikh}(G)$ from any context free grammar.

Chapter 3

Best VASR Abstractions of Programs

The aim of this chapter is to compute a set of VASR transformations whose behavior overapproximates the behavior of the input program. Our input program is provided to us as a program graph $M = \langle V, \Sigma, P, E, in, out, p_0 \rangle$ and a transition formula mapping $tf : \Sigma \rightarrow TF(X)$, where $X = \{x_1, \dots, x_n\}$ is a set of global variables¹. We are interested in computing a transition formula that over-approximates the relation $\frac{\mathcal{L}_M(p_0)}{\rightarrow_{\#}}$ as this represents all possible executions of our program.

To do so, we derive a VASR $\mathcal{V} = \langle \phi_R, \phi_+ \rangle$ such that $\frac{\mathcal{L}_M(p_0)}{\rightarrow_{\mathcal{V}}}$ is an over-approximation of $\frac{\mathcal{L}_M(p_0)}{\rightarrow_{\#}}$. We achieve this by requiring that there is a linear simulation f from the transition system defined by $\#$ to the transition system defined by $\mathcal{V}$. It implies that for all characters $s \in \Sigma$, $\{u, v : u \xrightarrow{s}_{\#} v\} \subseteq \{u, v : f(u) \xrightarrow{s}_{\mathcal{V}} f(v)\}$; this fact allows us to use a VASR transformation as an overapproximation for a transition formula. We can extend this reasoning to use the reachability of the VASR as an overapproximation for the reachability of the transition formula system, as

¹Since our methods are targeting procedures with recursive calls, it is unproductive to include local variables in our VASR abstraction, as a memoryless system will not be able to capture the behavior of local variables during recursive calls.

$$\left\{ u, v : u \xrightarrow{\mathcal{L}_M(p_0)}_{\#} v \right\} \subseteq \left\{ u, v : f(u) \xrightarrow{\mathcal{L}_M(p_0)}_{\mathcal{V}} f(v) \right\}.$$

We additionally show that $\mathcal{V}$ is *best* in the following sense: for any other VASR $\mathcal{V}'$ with a linear simulation f' from the transition system of l , there is a unique linear simulation f^* from $\mathcal{V}$ to $\mathcal{V}'$ such that $f' = f^* \circ f$. This implies $\left\{ u, v : f(u) \xrightarrow{s}_{\mathcal{V}} f(v) \right\} \subseteq \left\{ u, v : f'(u) \xrightarrow{s}_{\mathcal{V}'} f'(v) \right\}$. Then, approximating the dynamics of $\mathcal{V}$ provides at least as much information as any other simulating VASR.

3.0.1 Best Per-Edge Transformations

Let $\mathbf{x}$ denote the vector $[x_1, \dots, x_n]^T$. Consider the space of resets implied by a transition formula and the space of additions, respectively denoted as follows:

$$Res_{\#(s)} \triangleq \{ \langle \mathbf{a}, o \rangle \in \mathbb{Q}^n \times \mathbb{Q} : \#(s) \models (\mathbf{a}^T \mathbf{x}') = o \}$$

$$Add_{\#(s)} \triangleq \{ \langle \mathbf{a}, o \rangle \in \mathbb{Q}^n \times \mathbb{Q} : \#(s) \models (\mathbf{a}^T \mathbf{x}') = (\mathbf{a}^T \mathbf{x}) + o \}$$

These sets are linear spaces; it can be easily checked that they are closed under addition and scalar multiplication. We can leverage the work in [21] to generate a basis for the above sets via Algorithm 1 in [23]. Let $\langle A_R, \mathbf{o}_R \rangle$ and $\langle A_+, \mathbf{o}_+ \rangle$ be matrix representations of the bases for $Res_{\#(s)}$ and $Add_{\#(s)}$ respectively. Then, the best VASR abstraction of $\#(s)$ and the corresponding linear simulation are:

$$\langle f_s, \phi_R(s), \phi_+(s) \rangle = \left\langle \begin{bmatrix} A_R \\ A_+ \end{bmatrix}, \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \begin{bmatrix} \mathbf{o}_R \\ \mathbf{o}_+ \end{bmatrix} \right\rangle$$

Let $abstract : TF(X) \rightarrow d \times (\mathbb{Q}^{d \times n} \times \{0, 1\}^d \times \mathbb{Q}^d)$ be a function computing the best VASR transformation abstraction of the input transition formula and respective linear simulation.

Theorem 3.0.1. $f_s, \langle \phi_R(s), \phi_+(s) \rangle$ is a best abstraction of $\#(s)$; for any linear sim-

ulation and VASR transformation $f'_s, \langle \phi_R(s)', \phi_+(s)' \rangle$ such that $\mathcal{H}(s) \models f'_s(\mathbf{x}') = \phi_R(s)' * f'_s(\mathbf{x}) + \phi_+(s)'$, there exists a unique linear simulation f^* from $\langle \phi_R(s), \phi_+(s) \rangle$ to $\langle \phi_R(s)', \phi_+(s)' \rangle$ such that $f'_s = f^* \circ f_s$.

Proof. Each row of the matrix representation of f'_s must be in either $Res_{\mathcal{H}(s)}$ or $Add_{\mathcal{H}(s)}$, as these sets describe the set of all possible VASR transformations entailed by $\mathcal{H}(s)$. Then, we can conclude that f^* exists by the definition of a basis. $\square$

We still must combine our various transforms into a single VASR $\langle \phi_R^*, \phi_+^* \rangle$ with a single linear simulation f^* . While we have computed VASR transformations for each transition formula, this is not yet a VASR because the linear simulations can be different. For example, consider the following example with two transformations extracted via the above procedure from some transition formulas:

$$\Sigma = \{a, b\}$$

$$f_a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \langle \phi_R(a), \phi_+(a) \rangle = \left\langle \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\rangle$$

$$f_b = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \langle \phi_R(b), \phi_+(b) \rangle = \left\langle \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\rangle$$

This is not yet a VASR - there is no way to compose the transform of character a with that of character b as their linear simulations are different. If n is the dimension of our transition formula transition system, and d_a and d_b are the dimensions that the VASR transformations we are trying to combine operate over, we are trying to find the most general linear functions g_a, g_b making the following square commute:

This is a category-theoretic pushout, as described in the preliminaries, but the category of vector spaces is not the correct one to use here, as the resulting transformations may combine rows of our VASR transformations which are reset and not

$$\begin{array}{ccc}
\mathbb{Q}^n & \xrightarrow{f_a} & \mathbb{Q}^{d_a} \\
f_b \downarrow & & \downarrow g_a \\
\mathbb{Q}^{d_b} & \xrightarrow{g_b} & \mathbb{Q}^m
\end{array}$$

Figure 3.1: Desired Commutative Square to Align Two VASR Transformations

reset, making the output an improperly formed transformation. To approach this task, we formalize the state spaces of VASRs as **separated spaces** and simulations between VASRs as **linear coherent maps**, show how to find the most general VASR simulating two input VASRs via a category-theoretic *pushout*, and use our formalisms to compute $\langle \phi_R^*, \phi_+^* \rangle$ and f^* .

3.0.2 The Sep Category

The goal of this section is to define a category representing the state space of a VASR. The key restriction on morphisms that is necessary to preserve the structure of VASR transformations is that dimensions which are reset cannot be combined with dimensions that are reset. We introduce the following constructs to formalize this notion, and concretely state how the following category is related to VASRs in Theorem 3.0.2

Two dimensions $i, j \in [d]$ of a VASR $\mathcal{V}$ are **coherent** if for all characters $s \in \Sigma$, $(\phi_R(s))_i = (\phi_R(s))_j$ (every transformation which resets i also resets j). The **coherence classes** of a VASR $\mathcal{V}$ are the equivalence classes of $[d]$ according to coherence.

Let a **separated space** be defined as a pair $\langle S, H \rangle$ where S is a finite dimensional linear space and $H = \{C_1, \dots, C_n\}$ is a finite family of disjoint subspaces of S such that each $v \in S$ can be uniquely written as a sum $v = \sum_{i=1}^n v_i$ where $v_i \in C_i$. Let each v_i be denoted the **orthogonal projection** of v onto C_i . For each $C \in H$, define $\zeta_C : S \rightarrow C$ to be the linear map that sends each vector to its orthogonal projection onto C .

Given a d -dimensional VASR $\mathcal{V}$ with coherence classes $C_1, \dots, C_n$, let the corresponding separated space be $\langle S_{\mathcal{V}}, H_{\mathcal{V}} \rangle$ with $S_{\mathcal{V}} = \mathbb{Q}^d$ and $H_{\mathcal{V}} = \{D_1 \dots D_n\}$ with $D_i = \text{span}(\{e_j : j \in i\})$. Intuitively, this representation is relevant to the behavior of the VASR because all of the orthogonal projections of an input vector differ in whether or not they are reset by at least one character $s \in \Sigma$. This information is important because when combining VASRs with differing linear simulations, we must use each coherence class separately to preserve the structure of the VASR.

Let $\langle S_1, H_1 \rangle$ and $\langle S_2, H_2 \rangle$ be separated spaces. A **coherent linear map** is a pair $\langle f, w_f \rangle$ in which $f : S_1 \rightarrow S_2$ is a linear map and $w_f : H_2 \rightarrow H_1$ is a *witness function* such that for all $C_1 \in H_1$ and $C_2 \in H_2$, if $C_1 \neq w_f(C_2)$ then $\zeta_{C_2}(f(\zeta_{C_1}(s))) = 0$ for all s . Let $f_{C_2} : w_f(C_2) \rightarrow C_2$ be $f_{C_2}(s) = \zeta_{C_2} \circ f$ restricted to the domain $w_f(C_2)$ for all $C_2 \in H_2$. Then, by linearity and the coherent condition, we have $f(s) = \sum_{C_2 \in H_2} f_{C_2}(\zeta_{w_f(C_2)}(s))$. Put simply, a linear map is coherent if each of the orthogonal projections of its output is only dependent on one of the orthogonal projections of this input. In the context of VASRs, this condition ensures that we do not combine counters in different coherence classes.

Theorem 3.0.2. *Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be vector addition systems with corresponding separated spaces $\langle S_{\mathcal{V}_1}, H_{\mathcal{V}_1} \rangle$ and $\langle S_{\mathcal{V}_2}, H_{\mathcal{V}_2} \rangle$. If $f : S_{\mathcal{V}_1} \rightarrow S_{\mathcal{V}_2}$ is a linear simulation from $\mathcal{V}_1$ to $\mathcal{V}_2$, then there must exist a w_f such that $\langle f, w_f \rangle$ is a coherent linear map from $\langle S_{\mathcal{V}_1}, H_{\mathcal{V}_1} \rangle$ to $\langle S_{\mathcal{V}_2}, H_{\mathcal{V}_2} \rangle$.*

Proof. Let the components of $\mathcal{V}_i$ be denoted $\langle \phi_R^{\mathcal{V}_i}, \phi_+^{\mathcal{V}_i} \rangle$. We will define $T_{\mathcal{V}_i}(s, u) = \phi_R^{\mathcal{V}_i}(s) * u + \phi_+^{\mathcal{V}_i}(s)$. For coherence class C_j of $\mathcal{V}_i$, we will use $\phi_R^{\mathcal{V}_i}(s)_{C_j}$ to refer to the value of $(\phi_R^{\mathcal{V}_i}(s))_k$ for any $k \in C_j$; in other words, this term tells us whether C_j was or was not reset by character s .

We want to show that for all coherence classes C_1, C_2 of $\mathcal{V}_1, \mathcal{V}_2$ respectively, if there is some u such that $(\zeta_{C_2} \circ f \circ \zeta_{C_1})(u) \neq 0$, then for all $s \in \Sigma$ we have $(\phi_R^{\mathcal{V}_1}(s))_{C_1} = (\phi_R^{\mathcal{V}_2}(s))_{C_2}$. This fact implies that for each C_2 there exists at most one C_1 such that

$(\zeta_{C_2} \circ f \circ \zeta_{C_1})(u) \neq 0$ for some u , as all for all C_1, C'_1 , $(\phi_R^{\mathcal{V}_1}(s))_{C_1} \neq (\phi_R^{\mathcal{V}_1}(s))_{C'_1}$ for some s by the definition of coherence. If there were C_1 and C'_1 such that $(\zeta_{C_2} \circ f \circ \zeta_{C_1})(u) \neq 0$ and $(\zeta_{C_2} \circ f \circ \zeta_{C'_1})(u) \neq 0$, then the statement we want to prove would contradict the definition of a coherence class as it would imply that for all $s \in \Sigma$ we have $(\phi_R^{\mathcal{V}_1}(s))_{C_1} = (\phi_R^{\mathcal{V}_1}(s))_{C'_1}$. Therefore, after proving this statement, we can construct a witness function by mapping each C_2 to the unique C_1 meeting the condition above if one exists, and an arbitrary C_1 otherwise.

Consider an arbitrary $s \in \Sigma$. The definition of linear simulation implies that for all u , $f(T_{\mathcal{V}_1}(s, u)) = T_{\mathcal{V}_2}(s, f(u))$. Letting $u = 0$, we have that $f(\phi_+^{\mathcal{V}_1}(s)) = \phi_+^{\mathcal{V}_2}(s)$. Now, consider any coherence classes C_1, C_2 of $\mathcal{V}_1, \mathcal{V}_2$ such that $(\zeta_{C_2} \circ f \circ \zeta_{C_1})(u) = v \neq 0$. We will show by cases that $(\phi_R^{\mathcal{V}_1}(s))_{C_1} = (\phi_R^{\mathcal{V}_2}(s))_{C_2}$.

- Case 1: $(\phi_R^{\mathcal{V}_1}(s))_{C_1} = 1$. Then,

$$\begin{aligned}
T_{\mathcal{V}_2}(s, f(\zeta_{C_1}(u))) &= f(T_{\mathcal{V}_1}(s, \zeta_{C_1}(u))) \\
&= f(\zeta_{C_1}(u) + \phi_+^{\mathcal{V}_1}(s)) \\
&= f(\zeta_{C_1}(u)) + f(\phi_+^{\mathcal{V}_1}(s)) \\
&= f(\zeta_{C_1}(u)) + \phi_+^{\mathcal{V}_2}(s)
\end{aligned}$$

Because VASR transformations apply to each coherence class independently and each coherence class is disjoint, the above implies that

$$T_{\mathcal{V}_2}(s, \zeta_{C_2}(f(\zeta_{C_1}(u)))) = \zeta_{C_2}(f(\zeta_{C_1}(u))) + \phi_+^{\mathcal{V}_2}(s)$$

By our assumption, $\zeta_{C_2}(f(\zeta_{C_1}(u))) \neq 0$, so $(\phi_R^{\mathcal{V}_2}(s))_{C_2} = 1$; otherwise, the above equation would not hold.

- Case 2: $(\phi_R^{\mathcal{V}_1}(s))_{C_1} = 0$. Then,

$$\begin{aligned}
T_{\mathcal{V}_2}(s, f(\zeta_{C_1}(u))) &= f(T_{\mathcal{V}_1}(s, \zeta_{C_1}(u))) \\
&= f(\phi_+^{\mathcal{V}_1}(s)) \\
&= \phi_+^{\mathcal{V}_2}(s)
\end{aligned}$$

Because VASR transformations apply to each coherence class independently and each coherence class is disjoint, the above implies that

$$T_{\mathcal{V}_2}(s, \zeta_{C_2}(f(\zeta_{C_1}(u)))) = \phi_+^{\mathcal{V}_2}(s)$$

By our assumption, $\zeta_{C_2}(f(\zeta_{C_1}(u))) \neq 0$, so $(\phi_R^{\mathcal{V}_2}(s))_{C_2} = 0$; otherwise, $T_{\mathcal{V}_2}$ would have left part of $f(\zeta_{C_1}(u))$ unreset.

□

Let **Sep** be the category where the objects are separated spaces and the arrows are coherent maps. The remaining goal of this section can be formalized as follows: Given $|\Sigma|$ individual VASR transformations operating in different separated spaces and coherent linear maps f_s from the separated space of our transition formulas to each space, compute the multi-dimensional pushout of all the separated spaces. We can compute the pushout for **Sep** via the following:

Consider two coherent linear maps, $\langle f, w_f \rangle$ from $\langle S_1, H_1 \rangle$ to $\langle S_2, H_1 \rangle$ and $\langle g, w_g \rangle$ from $\langle S_1, H_1 \rangle$ to $\langle S_3, H_3 \rangle$. Let $D_{\langle C_2, C_3 \rangle}$ be the object corresponding to the **Vect**_Q pushout of f_{C_2} and g_{C_3} and let $a_{\langle C_2, C_3 \rangle}, b_{\langle C_2, C_3 \rangle}$ be the respective morphisms. Then, let $S_4 = \Pi_{w_f(C_2)=w_g(C_3)} D_{\langle C_2, C_3 \rangle}$ and let $H_4 = \{D_{\langle C_2, C_3 \rangle} : w_f(C_2) = w_g(C_3)\}$ Finally, let $a = \sum_{w_f(C_2)=w_g(C_3)} a_{\langle C_2, C_3 \rangle} \circ \zeta_{C_2}$, $b = \sum_{w_f(C_2)=w_g(C_3)} b_{\langle C_2, C_3 \rangle} \circ \zeta_{C_3}$, $w_a(D_{\langle C_2, C_3 \rangle}) = C_2$, and $w_b(D_{\langle C_2, C_3 \rangle}) = C_3$. Then:

$$pushout_{\mathbf{Sep}} = \langle \langle S_4, H_4 \rangle, \langle a, w_a \rangle, \langle b, w_b \rangle \rangle$$

Theorem 3.0.3. *For any separated space, $pushout_{\mathbf{Sep}}$ is the pushout of the category $\mathbf{Sep}$.*

Proof. The fact that the diagram commutes follows directly from the commutativity of the $\mathbf{Vect}_{\mathbb{Q}}$ pushout : for all pairs $\langle C_2, C_3 \rangle$ such that $w_f(C_2) = w_g(C_3)$, we have that

$$\begin{aligned} a_{\langle C_2, C_3 \rangle} \circ \zeta_{C_2} \circ f &= a_{\langle C_2, C_3 \rangle} \circ f_{C_2} && \text{by coherent condition of } f \\ &= b_{\langle C_2, C_3 \rangle} \circ g_{C_3} && \text{by commutativity of pushout} \\ &= b_{\langle C_2, C_3 \rangle} \circ \zeta_{C_3} \circ g && \text{by coherent condition of } g \end{aligned}$$

It follows that:

$$\sum_{\substack{\langle C_2, C_3 \rangle \\ w_f(C_2) = w_g(C_3)}} a_{\langle C_2, C_3 \rangle} \circ \zeta_{C_2} \circ f = \sum_{\substack{\langle C_2, C_3 \rangle \\ w_f(C_2) = w_g(C_3)}} b_{\langle C_2, C_3 \rangle} \circ \zeta_{C_3} \circ g$$

And so:

$$a \circ f = b \circ g$$

To prove universality, consider another object $\langle S_5, H_5 \rangle$ and morphisms $\langle c, w_c \rangle, \langle d, w_d \rangle$ such $c \circ f = d \circ g$. For all $C_5 \in H_5$, we must have that $w_c(C_5) \in H_2$, $w_d(C_5) \in H_3$, and $w_f(w_c(C_5)) = w_g(w_d(C_5))$. Then, $\langle w_c(C_5), w_d(C_5) \rangle$ must have been considered by the pushout procedure, so by the universality of the $\mathbf{Vect}_{\mathbb{Q}}$ there must exist a unique $u_{\langle w_c(C_5), w_d(C_5) \rangle}$ such that

$$u_{\langle w_c(C_5), w_d(C_5) \rangle} \circ a_{\langle w_c(C_5), w_d(C_5) \rangle} = c_{\langle w_c(C_5), w_d(C_5) \rangle}$$

$$u_{\langle w_c(C_5), w_d(C_5) \rangle} \circ b_{\langle w_c(C_5), w_d(C_5) \rangle} = d_{\langle w_c(C_5), w_d(C_5) \rangle}$$

Compositionally, $u = \sum_{C_5 \in H_5} u_{\langle w_c(C_5), w_d(C_5) \rangle} \circ \zeta_{D_{\langle w_c(C_5), w_d(C_5) \rangle}}$ is the unique coherent linear maps such that $u \circ a = c$ and $u \circ b = d$. Thus, our pushout is universal. $\square$

Given a VASR transformation and a coherent linear map $\langle f, w_f \rangle$, Define the image of the VASR transformation $\langle \phi_R(s), \phi_+(s) \rangle$ under a coherent linear map $\langle f, w_f \rangle$ to be $image(\langle \phi_R(s), \phi_+(s) \rangle, \langle f, w_f \rangle) = \langle w_f \boxtimes \phi_R(s), f(\phi_+(s)) \rangle$, where $(w_f \boxtimes \phi_R(s))_i = \phi_R(s)_j$ for any $j \in w_f(C_i)$ where C_i is the coherence class containing i . This image is also a VASR transformation, which was exactly the property we were looking for to complete the commutative square in Figure 3.1. Now, we can use $pushout_{\mathbf{Sep}}$ to align all of our VASR transformations.

3.0.3 Combining VASRs

We can now compute the best VASR simulating the input transition formula system with only linear calls to our pushout procedure with Algorithm 10. For readability, explicit references to witness function w_f are omitted.

Input: Alphabet $\Sigma = \{s_1 \dots s_{|\Sigma|}\}$, global variables X , transition formula mapping $l : \Sigma \mapsto TF(X)$
Output: Linear simulation f , best d -dimensional VASR $\langle \phi_R, \phi_+ \rangle$

- 1 $f_{s_i}, \phi_R(s_i), \phi_+(s_i) \leftarrow abstract(l(s_i))$ for all $s_i \in \Sigma$
- 2 $curr \leftarrow f_{s_1}$
- 3 **for** $i \in [2, \dots, |\Sigma|]$ **do**
- 4 $\langle -, a_i, b_i \rangle \leftarrow pushout_{\mathbf{Sep}}(curr, f_{s_i})$
- 5 $curr \leftarrow a_i \circ curr$
- 6 $r \leftarrow \mathbb{I}$
- 7 **for** $i \in [|\Sigma| \dots 1]$ **do**
- 8 $\langle \phi_R(s_i), \phi_+(s_i) \rangle \leftarrow image(\langle \phi_R(s_i), \phi_+(s_i) \rangle, r \circ b_i)$
- 9 $r \leftarrow a_i \circ r$
- 10 **return** $b_{|\Sigma|} \circ f_{|\Sigma|}, \langle \phi_R, \phi_+ \rangle$

Algorithm 1: GenVASR

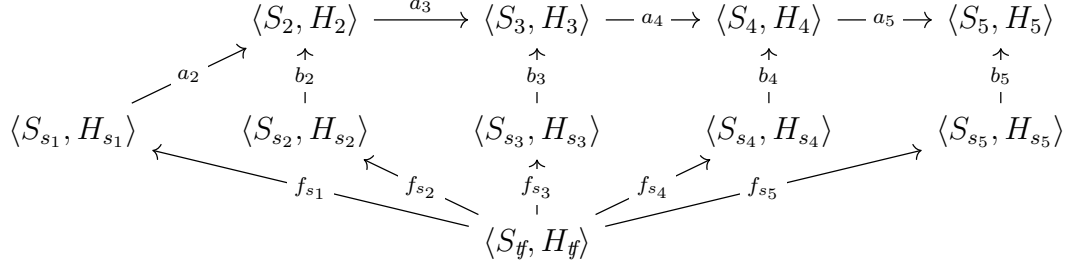


Figure 3.2: GenVASR Visualized

Theorem 3.0.4. *Let Σ be an alphabet, $X = \{x_1 \dots x_n\}$ be a set of global variables, and $\sharp: \Sigma \mapsto TF(X)$ be a transition formula mapping. Let $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$. Let $\langle f, \mathcal{V} \rangle = GenVASR(\Sigma, \sharp)$.*

- *Correctness: For any valuations $\rho, \rho' : X \mapsto \mathbb{Z}$ and character $s \in \Sigma$, if $\rho \xrightarrow{s}_{\sharp} \rho'$, then $f(\rho) \xrightarrow{s}_{\mathcal{V}} f(\rho')$.*
- *Universality: For all other VASRs $\tilde{\mathcal{V}}$ such that there exists a linear simulation $\tilde{f}$ from the transition system defined from $\sharp$ to the transition system defined by $\tilde{\mathcal{V}}$, there exists a linear simulation g from $\mathcal{V}$ to $\tilde{\mathcal{V}}$ such that $\tilde{f} = g \circ f$.*

Proof. Correctness follows directly from the correctness of the *abstract* procedure, which implies that the initial assignment to $\langle \phi_R(s_i), \phi_+(s_i) \rangle$ meets the correctness condition. The execution of *GenVASR* iteratively takes the image of this initial assignment under coherent linear maps, so since coherent linear maps are closed under composition we can conclude that overall effect of *GenVASR* is a coherent linear map. Then, by the definition of linear simulation, our correctness condition holds.

Likewise, universality follows from the fact that the *abstract* procedure is best and *pushout_{sep}* is universal. The backwards pass of the algorithm maintains the invariant that all transformations that have been processed thus far meet the universality constraint for the transition system restricted to the edges that have been processed thus far. $\square$

Summary This section described an algorithm to compute a decidable approximate model of an input program (VASR mapping) from an undecidable model (transition formula mapping). To do so, it computed the best possible VASR transformation simulating each transition formula independently via results from the literature, then used the *pushout* of the category of VASR state spaces to combine the transformations in the most general possible way. The output is a VASR mapping which captures as much information as any other simulating VASR mapping, and is therefore best.

Chapter 4

Reachability of VASRS along Context-Free Program Paths

In this section, we show how to compute in polynomial time a transition formula which encodes the transition relation of a labeled VASR over any context free language. This problem is relevant to our end-to-end procedure summarization because context free languages capture the behavior of recursive procedures.

Consider the single dimensional VASR described in Figure 4.1. To compute the transition relation associated with a particular word $w \in \Sigma^*$, it is sufficient to identify the characters corresponding to the final reset of each dimension from left to right

$$\begin{array}{lll}
 \Sigma = \{a, b, c\} & \phi_+(s) = \begin{cases} 1 & s = a \\ 2 & s = b \\ 3 & s = c \end{cases} & \phi_R(s) = \begin{cases} 0 & s = a \\ 1 & s = b \\ 1 & s = c \end{cases} \\
 w_1 = aba\bar{a} \underbrace{c} & \begin{array}{l} \text{c after final reset: 1} \\ \text{b after final reset: 0} \end{array} & u \xrightarrow{w_1}_{\mathcal{V}} v \leftrightarrow v = u + 0 \cdot 2 + 1 \cdot 3 \\
 w_2 = abc\bar{a} \underbrace{bcbb} & \begin{array}{l} \text{c after final reset: 1} \\ \text{b after final reset: 3} \end{array} & u \xrightarrow{w_2}_{\mathcal{V}} v \leftrightarrow v = u + 3 \cdot 2 + 1 \cdot 3 \\
 w_3 = \underbrace{bccbb} & \begin{array}{l} \text{c after final reset: 2} \\ \text{b after final reset: 4} \end{array} & u \xrightarrow{w_3}_{\mathcal{V}} v \leftrightarrow v = u + 2 \cdot 2 + 4 \cdot 3
 \end{array}$$

Figure 4.1: VASR example

along the word and to count the number of each character after the final reset. The final reset nullifies the effects of all previous characters, and all operations after the final reset commute because they are all adds so the count is all you need.

We formalize this reduced set of information about a word w as a *labeling*, and show how to compute the set of labelings of a context-free language $\mathcal{L}(G)$ from its grammar representation $G = \langle N, \Sigma, R, n_0 \rangle$. We then show how to use a labeling to compute the transition relation of the word it represents, and use this in conjunction with the set of all labelings to represent the reachability relation of the VASR.

Definition 4.0.1 (Labeling). *Let $\mathcal{V} = \langle \phi_R, \phi_+ \rangle$ be a d -dimensional VASR over alphabet Σ . A **labeling** is a function $n : (\Sigma \times [2d+1]) \mapsto \mathbb{N}$ which satisfies the condition that for all even i , $\sum_{s \in \Sigma} n(s, i) \leq 1$. A labeling identifies up to d symbols from a word using its even symbols, while the odd symbols represent the partial words in between.*

*A labeling is **well-formed** if for all odd $i \in [2d+1]$ and all $j \in [d]$, if there is some $s \in \Sigma$ such that $\phi_R(s)_j = 0$ and $n(s, i) > 0$, then there is some $a' \in \Sigma$ and some even $k > i$ such that $\phi_R(a')_j = 0$ and $n(a', k) > 0$. This condition ensures that the final reset of each class occurs at an even index.*

For a word $w \in \Sigma^*$ and a labeling n , write $w \Vdash n$ if and only if there exists a decomposition $w = w_1 w_2 \dots w_{2d+1}$ such that for all $i \in [2d+1]$ and $s \in \Sigma$, $n(s, i) = \pi(w_i)(s)$.

A well-formed labeling n such that $w \Vdash n$ identifies up to d symbols from w , corresponding to the nonzero values of $n(s, i)$ for all even i , such that the final reset from left to right of each dimension must be one of the symbols. We can encode the VASR operation corresponding to any word w from a well-formed labeling n such that $w \Vdash n$ via the following formula $\mathcal{Q}$. The free variables of this formula are count variables $c_{s,i}$ for all $s \in \Sigma$ and $i \in [2d+1]$ representing the value of the labeling and u_i, v_i for all $i \in d$ representing the components of the vectors that the transition is operating on. The transformation is computed by considering for each component of

the space that the VASR operates on what the last time the component is reset in the word that is labeled by the $c_{s,i}$; it accordingly sets the value of v_j .

$$\begin{aligned}
& \mathcal{Q} \bigwedge_{j \in [d]} \exists m_j \bigwedge_{k \in [d]} \left(\sum_{s \in \Sigma, \phi_R(s)_j=0} c_{s,2k} > 0 \wedge \sum_{s \in \Sigma, \phi_R(s)_j=0} \sum_{k < l \leq d} c_{s,2l} = 0 \right) \\
& \implies v_j = \left(\sum_{i=m_j}^{2d+1} \sum_{s \in \Sigma} \phi_+(s)_j \cdot c_{s,i} \right) \wedge \\
& \quad \left(\sum_{s \in \Sigma, \phi_R(s)_j=0} \sum_{l \in [d]} c_{s,2l} = 0 \right) \\
& \implies v_j = \left(\sum_{i=m_j}^{2d+1} \sum_{s \in \Sigma} \phi_+(s)_j \cdot c_{s,i} \right) + u_j
\end{aligned}$$

Theorem 4.0.2. *Consider a VASR $\mathcal{V}$. For any word $w \in \Sigma^*$ and any well-formed labelling n such that $w \Vdash n$, we have $u \xrightarrow{w}_{\mathcal{V}} v$ if and only if $\mathcal{Q}[c_{s,i} \mapsto n(s,i)]$ holds when every free variable u_j, v_j is replaced with the respective component of u and v .*

We now show an efficient method to compute a transition formula $\mathcal{F}(G, \mathcal{V})$ with free variables $c_{s,i}$ for all $s \in \Sigma, i \in [2d+1]$ such that n is a well-formed labeling and there exists some $w \in \mathcal{L}(G)$ such that $w \Vdash n$ if and only if $\mathcal{F}(G, \mathcal{V})[c_{s,i} \mapsto n(s,i)]$ holds. We dub this object the **label-ability formula** of G . Once we have the label-ability formula, we can conjoin it to $\mathcal{Q}$ as defined above to encode the reachability of $\mathcal{V}$ subject to the context free language $\mathcal{L}(G)$. This can be expressed formally via the following theorem.

Theorem 4.0.3. *There is a polynomial-time procedure which, given a VASR $\mathcal{V} = (\phi_R, \phi_+)$ over alphabet Σ and a grammar G over the same alphabet, computes a formula $R(u, v)$ such that $[u, v] \models R$ if and only if $u \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}} v$.*

4.0.1 Interval Grammars

In this section, we will show how to compute the label-ability formula $\mathcal{F}(G, \mathcal{V})$ of a grammar G . This section will assume that the input grammar $G = \langle N, \Sigma, P, s_0 \rangle$ is in Chomsky Normal Form.

Let $\Sigma_1 \dots \Sigma_{2d+1}$ be copies of Σ , $\Sigma_i \triangleq \{\langle s, i \rangle : s \in \Sigma\}$. Let $h : (\Sigma \times [2d+1])^* \rightarrow \Sigma^*$ be the homomorphism that maps $\langle a, i \rangle \mapsto a$ for all i .

Consider the regular language

$$\mathcal{O} \triangleq \Sigma_1^*(\Sigma_2 + \epsilon)\Sigma_3^* \dots \Sigma_{2d-1}^*(\Sigma_{2d} + \epsilon)\Sigma_{2d+1}^*$$

The Parikh image $\mathcal{O}$ is equal to the set of all labelings over Σ , since for any $w \in \mathcal{O}$, we have $\sum_{a \in \Sigma} \pi(w)(a, i) \leq 1$ for every even i . Then, if $w \in h^{-1}(\mathcal{L}(G)) \cap \mathcal{O}$, then $h(w) \in \mathcal{L}(G)$ and $h(w) \models \pi(w)$. Since context free languages are closed under inverse homomorphisms and intersections with regular languages, $h^{-1}(\mathcal{L}(G)) \cap \mathcal{O}$ is context-free. Then, it is possible to represent this language as a grammar and encode its Parikh image into a formula; this formula will be the label-ability formula of $\mathcal{L}(G)$.

Now, we provide an explicit construction of a grammar $\mathcal{I}(G)$ generating the language $h^{-1}(\mathcal{L}(G)) \cap \mathcal{O}$ and utilize it to construct the label-ability formula $\mathcal{F}(G, \mathcal{V})$ of $\mathcal{L}(G)$.

We can construct $\mathcal{I}(G)$ in the following way. We introduce new “interval symbols”

$$N_{\llbracket} \triangleq \{n_{[2i-1:2j-1]} : n \in N, i \in [d+1], j \in [d+1], i \leq j\}$$

The intention of the grammar design is that the set of words derivable from $n_{[i:j]}$ is $\mathcal{L}_{\mathcal{I}(G)}(n_{[i:j]}) = h^{-1}(\mathcal{L}_G(n)) \cap (\Sigma_i^*(\Sigma_{i+1} + \epsilon) \dots (\Sigma_{j-1} + \epsilon)\Sigma_j^*)$. To ensure all derived words are in $h^{-1}(\mathcal{L}_G(n))$, all rules in $\mathcal{I}(G)$ are copies of rules in G . To ensure all derived words are in $(\Sigma_i^*(\Sigma_{i+1} + \epsilon) \dots (\Sigma_{j-1} + \epsilon)\Sigma_j^*)$, we only allow productions from $n_{[i:j]}$ to produce

terminal symbols from $\Sigma_i \cup \dots \cup \Sigma_j$ and leverage the fact that each nonterminal can only create one symbol by ensuring that at every step of the derivation there is at most one $n_{[i:j]}$ such that $i \leq k \leq j$ for all even k .

$$\begin{aligned} & \{A_{[2i+1:2j+1]} \Rightarrow B_{[2i+1:2k+1]}C_{[2k+1:2j+1]} : A \Rightarrow BC \in P, 0 \leq i \leq k \leq j \leq d\} \\ P_{\square} \triangleq & \cup \{A_{[2i+1:2j+1]} \Rightarrow \langle a, k \rangle : A \Rightarrow a \in P, 2i+1 \leq k \leq 2j+1\} \\ & \cup \{s_{[1:2d+1]} \Rightarrow \epsilon : s \Rightarrow \epsilon \in P\} \end{aligned}$$

$$\mathcal{I}(G) = \langle N_{\square}, \Sigma \times [2d+1], P_{\square}, s_{[1,2d+1]} \rangle$$

Theorem 4.0.4. *For any grammar $G = (N, \Sigma, P, s_0)$, we have*

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

So, $\pi(\mathcal{L}(\mathcal{I}(G)))$ is the set of all labelings n such that $w \Vdash n$ for some $w \in \mathcal{L}(G)$.

Additionally, $\mathcal{I}(G)$ only blows up the size of the original grammar polynomially; specifically, $\mathcal{I}$ has $O(d|T|)$ terminals, $O(d^2|N|)$ nonterminals, and $O(d^3|P|)$ productions.

We can compute a formula $\psi_{parikh}(\mathcal{I}(G))$ representing the parikh image of $\mathcal{L}(\mathcal{I}(G))$, as mentioned in the preliminaries section [25]. We can place additional restrictions on the free variables of this formula to ensure that all models represent well-formed labelings subject to $\mathcal{V} = \langle \phi_R, \phi_+ \rangle$, and combine it with ψ_{parikh} to encode the set of well-formed labelings of words in the language of our grammar into LIRA.

$$\begin{aligned}
\psi_{wf}(\mathcal{I}(G), \langle \phi_R, \phi_+ \rangle) &\triangleq \bigwedge_{i \in [d+1]} \bigwedge_{j \in [d]} (\bigvee_{s \in \Sigma} \phi_R(s)_j = 0 \wedge c_{s, 2i-1} > 0) \\
&\implies (\bigvee_{k=i}^d \bigvee_{s \in \Sigma} \phi_R(s)_j = 0 \wedge c_{s, 2k} > 0)
\end{aligned}$$

$$\mathcal{F}(G, \mathcal{V}) \triangleq (\exists \{m_{s,i} : s \in \Sigma, i \in [2d+1]\}, \psi_{parikh}(\mathcal{I}(G))[c_{s,i} \mapsto m_{s,i}]) \wedge \psi_{wf}(\mathcal{I}(G), \mathcal{V})$$

Theorem 4.0.5. $\mathcal{F}(G, \mathcal{V})[c_{s,i} \mapsto n(s, i)]$ holds if and only if n is a well-formed labeling subject to $\mathcal{V}$ and there exists some $w \in \mathcal{L}(G)$ such that $w \Vdash n$.

Finally, we can define the following formula R which is an explicit construction of Theorem 4. This is a formula with d -dimensional free variables u, v such that $[u, v] \models R$ if and only if $u \xrightarrow{\mathcal{L}(G)}_{\mathcal{V}} v$.

$$R \triangleq \exists \{m_{s,i} : s \in \Sigma, i \in [2d+1]\}, (\mathcal{F}(G, \mathcal{V}) \wedge \mathcal{Q})[c_{s,i} \mapsto m_{s,i}]$$

4.0.2 End-to-end Summarization Procedure

Then, at a high level, our end-to-end summarization procedure is as follows. We receive as input a recursive graph M representing the structure of our input program and a transition formula mapping f over global variables X mapping the edges of our graph to transition formulas describing the program dynamics associated with computations between the endpoints of that edge. We apply the *GenVASR* algorithm from Chapter 3 to compute the best VASR abstraction of the input program $\mathcal{V}$ and the linear simulation f corresponding the variables of our original program with the counters of $\mathcal{V}$. We now compute a grammar G in Chomsky Normal Form generating the language of paths through our recursive graph M and apply the construction of

Theorem 4 to encode the relation $\xrightarrow{\mathcal{L}(G)}_{\mathcal{V}}$ into a formula R . Our final summary is the following transition formula over variables X and X' :

$$Summary(M, \mathfrak{f}) \triangleq R[u \rightarrow f(X), v \rightarrow f(X')]$$

By Theorems 3.0.4 and 4, we know that for all valuations ρ, ρ' , if $\rho \xrightarrow{\mathcal{L}_M(p_0)}_{\mathfrak{f}} \rho'$, then $\rho, \rho' \models Summary(M, \mathfrak{f})$. In addition to being correct in this sense, *Summary* is monotone in the sense specified by the following theorem, guaranteeing a type of predictability in the behavior of the summarization procedure on related programs.

Theorem 4.0.6 (Monotonicity). *Consider two transition formula mappings $\mathfrak{f}$ and $\mathfrak{f}'$ such that $\mathfrak{f}(s) \implies \mathfrak{f}'(s)$ for all symbols s . Then, for all program graphs,*

$$Summary(M, \mathfrak{f}) \implies Summary(M, \mathfrak{f}')$$

Summary This section describes *how* the reachability of a VASR mapping is computable over a context free language (representing a pushdown system). To do so, we formalized the necessary information to compute the transformation associated with a word and showed how to compute the set of all such informations for a context free language from its grammar representation. Conjoined with an additional formula encoding the resulting transformation on vectors, the output of this section is a LIRA formula representing the reachability of our VASR.

Chapter 5

Evaluation

We implemented the algorithms detailed in Chapters 3 and 4 as an extension to Compositional Recurrence Analysis (CRA), a program analysis framework which uses recurrence relations to summarize loops and Kleene iteration to summarize recursive procedures. The methods described in this paper are used to generate procedure summaries which we use as an alternative to the non-monotone iterative summaries. Concretely, our implementation was built as a part of Duet [6], a static analysis tool. The source code can be found at [20]. In this section, we aim to use this implementation to answer two research questions:

1. Does computing VASR reachability over context free languages generate effective invariants for programs with non-linear recursion?
2. How does our method compare to the state of the art on standard benchmarks?

5.0.1 RQ1: Efficacy of Context-Free Reachability

To highlight the strengths of the summarization methods described the previous sections, we introduce three new benchmarks, `aligned_access`, `tree_counter`, and `lexer`. The source code of each function can be seen in Figures 5.1, 5.2, and 1.0.1 respectively.

Figure 5.1: Aligned Memory Accesses

```
void random_sample(int x) {
    if (__VERIFIER_nondet_int()) {
        random_sample(x + 4);
    }
    else if (__VERIFIER_nondet_int()) {
        random_sample(x + 32);
    } else {
        last_access = x;
    }
}

void main() {
    random_sample(0);
    assert(last_access % 4 == 0);
}
```

Figure 5.2: Node Counting in Trees

```
void tree_count() {
    if (*) {
        leafs += 1;
        return;
    } else {
        internal_nodes += 1;
        tree_count();
        tree_count();
        return;
    }
}

void main() {
    tree_count();
    assert(internal_nodes + 1 == leafs);
}
```

In Figure 5.1, `aligned_access` recursively loops arbitrarily and increments x randomly by multiples of 4. We hypothesize that Kleene iteration-based analysis methods should be unable to find an accurate summary for this procedure because there is no linear relationship between successive values of x . However, generating per-edge VASR transformations allows us to fully capture the behavior of the program and prove the assertion. This toy example shows that our methods could be useful for proving that memory accesses are aligned in much bigger programs.

In Figure 5.2, `tree_counter` uses non-linear recursion and makes an assertion about the structure of its recursive call trace. Our methods are perfectly attuned to summarize this procedure, as we can summarize every operation as a VASR transformation and leverage our reachability over context-free languages to exactly identify the paths through the program. Then, we are able to effectively prove that in all binary trees the number of internal nodes is one less than the number of leaves.

In Figure 1.0.1, `lexer` highlights how including resets in our VASRs enables our summaries to be more expressive. By treating *start*–*end* as one of the VASR counters, we are able to accurately capture the behavior of the program and prove that the start pointer does not go past the end pointer. This example additionally shows an example of real-world code that would benefit from the content of this thesis.

5.0.2 RQ2: Comparison with State of the Art

	#tasks	GS (15s)		UAutomizer (15s)		UAutomizer (600s)	
		#correct	time	#correct	time	#correct	time
recursive	17	2	36.7	2	265.9	12	3440.2
recursive-simple	35	13	48.4	9	526.8	28	5834.6
cf-crafted	3	3	6.7	3	23.7	3	25.7
Total	55	18	91.8	14	816.4	33	9300.5
Timeouts		0		38		12	
Mean time		2.4		10.8		41.7	
Median time		1.2		10.1		16.5	

Finally, we compared our methods to the performance of state of the art assertion checkers. We chose to evaluate all tools on the `recursive` and `recursive-simple` directories within the recursive subcategory of the Software Verification Competition (SV-COMP). As our methods are only able to prove safety, we selected all benchmarks that contained true assertions within `recursive` and `recursive-simple`, yielding sets of size 17 and 35 respectively. We additionally test all checkers on a suite of the benchmarks we introduce in this paper, which we dub `cf-crafted`. Timings were gathered on a virtual machine running Ubuntu 20.04 with 8 GB of RAM on a machine with 16GB of RAM and a 2.3 GHz Intel i7 CPU.

In general, the `recursive` and `recursive-simple` suites are not ideal tests of our methods abilities, as many of their tasks include if statements for which our VASR abstractions are too approximate to prove the associated assertions. Then, in several cases, our inability to prove the assertions in some tasks is simply a shortcoming of the abstraction model we employ. In other cases, we are hampered by the fact that our VASR transformations can only be computed over global variables, as the behavior of local variables is unrepresentable. These limitations will be further explored in

the next chapter; despite them, we are able to solve a comparable number of benchmark tasks in a fraction of the time taken by a state of the art program analyzer, UAutomizer [10].

Chapter 6

Discussion

6.0.1 Related Work

Interprocedural analysis Computing summaries that approximate the dynamics of (possibly recursive) procedures is a classical problem in program analysis [4, 22]. The dominant approach to computing is based on *iterative approximation*, which computes a summary assignment as the limit of a Kleene iteration sequence. For abstract domains that fail the ascending chain condition, the limit can be overapproximated by using a widening operator [3]; however, this results in a non-monotone analysis.

Newtonian program analysis [5] is a generic method for solving interprocedural dataflow equations which is based on Newton iteration rather than Kleene iteration. Rather than requiring the ascending chain condition, Newtonian program analysis relies on commutativity of the sequencing operation. Newtonian program analysis can be used as an alternative to our method for computing the reachability relation of vector addition systems, but cannot accomodate resets since they do not commute.

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*. While reachability in the classical

model is decidable, the complexity of the problem is non-elementary [15].

The *integer* VAS model was introduced by Haase and Halfon [9]. In contrast to classical VAS, reachability for integer VAS is decidable in NP. Letting $S = \langle \Sigma, \phi \rangle$ denote a vector addition system, Haase and Halfon showed that the reachability relation $\xrightarrow{L}_S$ can be encoded in as a Presburger formula in polynomial time in the case that $L = \Sigma^*$ (i.e., the reachability relation of an integer VASR) and the case that L is regular (i.e., the reachability relation of an integer VASR with states). Chistikov et al. extends this construction to the case where L is the language of a communication-free Petri net [1]. Chapter 4 uses the same ideas as Haase and Halfon to show that $\xrightarrow{L}_S$ is Presburger-definable in the case that L is a context-free language. Our construction differs somewhat in that they show how the reachability relation can be computed using standard primitives in formal language theory. The essence of our result is that it is possible to compute the reachability relation for a VASR for any family of languages that is effectively closed under intersection with regular languages and inverse homomorphism, and for which Parikh images are Presburger-definable.

Pushdown vector addition systems (over the naturals) have been investigated in [16, 17, 8]. The question of whether reachability is decidable for this model is open.

Silverman and Kincaid give a loop summarization algorithm that is based on computing the reachability relation of a VASR that simulates the loop [23]. We extend these techniques beyond loops to programs with recursive procedures.

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 [12, 18] or the Houdini algorithm [7].

There has been a recent line of work on monotone program analysis [23, 27, 26, 13] using *algebraic program analysis* [14]. This work has concentrated on monotone analysis of loops, and falls back on an iterative strategy to summarize recursive procedures.

This paper provides an alternative.

6.0.2 Limitations and Future Work

A notable limitation of this analysis method that is evident in the evaluation is that its inability to consider local variables restricts its ability to capture the dynamics of conditional flows. The program paths that we consider are frequently not fully constrained with respect to the semantics of the conditional statements, so our method frequently produces a summary that is too over-approximate to prove the relevant assertion.

In cases in which the conditional is dependent on global variables, prior work ([9], [23]) has shown success in encoding conditional statements into the VASR model through the introduction of states. This points to a possible extension of the work in this paper - extending the VASR model with states and showing that the resulting reachability is still encodable in LIRA.

However, there are many cases in which conditionals are dependent on local variables. VASRs are unable to represent the behavior of local variables during recursive calls because doing so requires the state to store memory about the previous value of variable. This suggests that the analysis methods of this thesis may be complemented well by an independent analysis process aiming to incorporate local reasoning into the summary. One potential tie-in is the free variables in our summary representing the number of times each procedure is called in the grammar derivation of a program path. By bounding the number of recursive calls within any run of a procedure, we could effectively bound the depth of recursion that our system considers and increase our summaries' specificity enough to solve several more of the benchmarks in the evaluation section.

6.0.3 Conclusion

This thesis has presented an end-to-end procedure summarization algorithm that is capable of capturing complex numerical behavior in nonlinearly recursive programs. The key selling point of an analysis like this is that it is *monotonone* - if for each edge the associated transition formula of Program A implies the associated transition formula of Program B, we are guaranteed that the summary of Program A implies that of Program B. The practical implications for end users of such an analysis is that code changes which increase specificity are guaranteed to only improve the specificity of the analysis - many program analyzers fail to meet this expectation. Our evaluation empirically showed that the work of this thesis is able to solve a number of interesting benchmarks extremely quickly. Through its qualitative and quantitative contributions to the literature, this thesis lays the groundwork for a future where verified software is the norm, and program analysis tools play a central role in ensuring that the software we rely on is trustworthy and secure.

Bibliography

- [1] D. Chistikov, C. Haase, and S. Halfon. Context-free commutative grammars with integer counters and resets. *CoRR*, abs/1511.04893, 2015.
- [2] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.
- [3] A. Cortesi. Widening operators for abstract interpretation. In *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 31–40, 2008.
- [4] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*, pages 237–277. North-Holland, 1977.
- [5] J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *J. ACM*, 57(6), nov 2010.
- [6] A. Farzan and Z. Kincaid. Duet: Static analysis for unbounded parallelism. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, pages 191–196, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [7] C. Flanagan and K. R. M. Leino. 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, page 500–517, Berlin, Heidelberg, 2001. Springer-Verlag.
- [8] M. Ganardi, R. Majumdar, and G. Zetsche. The complexity of bidirected reachability in valence systems. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery.
 - [9] C. Haase and S. Halfon. Integer vector addition systems with states. In *Lecture Notes in Computer Science*, pages 112–124. Springer International Publishing, 2014.
 - [10] M. Heizmann, J. Hoenicke, and A. Podelski. Software model checking for people who love automata. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, pages 36–52, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
 - [11] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
 - [12] M. Karr. Affine relationships among variables of a program. *Acta Inf.*, 6(2):133–151, jun 1976.
 - [13] Z. Kincaid, N. Koh, and S. Zhu. When less is more: Consequence-finding in a weak theory of arithmetic. *Proc. ACM Program. Lang.*, 7(POPL), jan 2023.
 - [14] Z. Kincaid, T. Reps, and J. Cyphert. Algebraic program analysis. In A. Silva and K. R. M. Leino, editors, *Computer Aided Verification*, pages 46–83, Cham, 2021. Springer International Publishing.
 - [15] R. Lazic. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013.

- [16] R. Lazic. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013.
- [17] J. Leroux, G. Sutre, and P. Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Automata, Languages, and Programming*, pages 324–336, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [18] M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL ’04, page 330–341, New York, NY, USA, 2004. Association for Computing Machinery.
- [19] R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, oct 1966.
- [20] N. Pimpalkhare and Z. Kincaid. Implementation repository. <https://github.com/nikhilpim/duet>, 2023.
- [21] T. Reps, M. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 252–266, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [22] M. Sharir and A. Pnueli. *Two approaches to interprocedural data flow analysis*. New York Univ. Comput. Sci. Dept., New York, NY, 1978.
- [23] J. Silverman and Z. Kincaid. Loop summarization with rational vector addition systems. In I. Dillig and S. Tasiran, editors, *Computer Aided Verification*, pages 97–115, Cham, 2019. Springer International Publishing.

- [24] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.
- [25] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In R. Nieuwenhuis, editor, *Automated Deduction – CADE-20*, pages 337–352, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [26] S. Zhu and Z. Kincaid. Reflections on termination of linear loops. In A. Silva and K. R. M. Leino, editors, *Computer Aided Verification*, pages 51–74, Cham, 2021. Springer International Publishing.
- [27] S. Zhu and Z. Kincaid. Termination analysis without the tears. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 1296–1311, New York, NY, USA, 2021. Association for Computing Machinery.