Context-Free-Language Reachability for Almost-Commuting Transition Systems

NIKHIL PIMPALKHARE, Princeton University, USA
ZACHARY KINCAID, Princeton University, USA
THOMAS REPS, University of Wisconsin, USA

We extend the scope of context-free-language (CFL) reachability to a new class of infinite-state systems. Parikh's Theorem is a useful tool for solving CFL-reachability problems for transition systems that consist of *commuting* transition relations. It implies that the image of a context-free language under a homomorphism into a commutative monoid is semi-linear, and that there is a linear-time algorithm for constructing a Presburger arithmetic formula that represents it. However, for many transition systems of interest, transitions do not commute.

In this paper, we introduce *almost-commuting transition systems*, which pair finite-state control with commutative components, but which are in general not commutative. We extend Parikh's theorem to show that the image of a context-free language under a homomorphism into an *almost-commuting monoid* is semi-linear and that there is a polynomial-time algorithm for constructing a Presburger arithmetic formula that represents it. This result yields a general framework for solving CFL-reachability problems over *almost-commuting transition systems*. We describe several examples of systems within this class. Finally, we examine closure properties of almost-commuting monoids that can be used to modularly compose almost-commuting transition systems while remaining in the class.

CCS Concepts: • **Theory of computation** → **Grammars and context-free languages**; **Program analysis**.

Additional Key Words and Phrases: CFL-Reachability, Context-Free Language, Parikh Image, Semilinear sets, Program Analysis, Infinite-State Systems, Attribute Grammars

ACM Reference Format:
Nikhil Pimpalkhare, Zachary Kincaid, and Thomas Reps. 2026. Context-Free-Language Reachability for Almost-Commuting Transition Systems. *Proc. ACM Program. Lang.* 10, POPL, Article 44 (January 2026), 26 pages. https://doi.org/10.1145/3776686

1 Introduction

Context-free language reachability [Yannakakis 1990] is a fundamental yet challenging problem. It asks whether a pair of states in a transition system is connected by a path whose sequence of labels belongs to a context-free language. Many important program analyses can be formulated as CFL-reachability problems [Reps 1998], such as pointer analyses [Lu et al. 2013; Su et al. 2014] and flow analyses [Pratikakis et al. 2006; Rehof and Fähndrich 2001]. This paradigm provides a principled foundation for reasoning about recursion.

Traditionally, CFL-reachability has been studied over finite-state systems. However, it is important to be able to analyze models with an infinite state space. In practice, many systems are most effectively analyzed by abstracting their large, finite state spaces as infinite. For example, rather

Authors' Contact Information: Nikhil Pimpalkhare, Princeton University, Princeton, USA, nikhil.pimpalkhare@princeton.edu; Zachary Kincaid, Princeton University, Princeton, USA, zkincaid@cs.princeton.edu; Thomas Reps, University of Wisconsin, Madison, USA, reps@cs.wisc.edu.

ACM 2475-1421/2026/1-ART44
https://doi.org/10.1145/3776686

than dealing with all 2^{32} possible integer values in a 32-bit system, it is often more useful to model such variables as unbounded integers. This abstraction improves tractability and yields analyses that are robust to changes in system-specific limits (e.g., moving to a 64-bit system).

Past work can be put in three classes, as follows:

(1) CFL-reachability over a *finite*-state transition system [Reps et al. 1995; Reps 1998]
(2) *Ordinary* reachability over the (infinite-state) transition system of a pushdown system [Bouajjani et al. 1997; Esparza et al. 2000; Finkel et al. 1997]
(3) CFL-reachability over a class of *infinite*-state transition systems [Pimpalkhare and Kincaid 2024a,b]

Classes (1) and (2) address related problems. For example, when the finite-state transition system in (1) is the interprocedural control-flow graph G of a recursive program, the CFL-reachability problem can be turned into an ordinary reachability problem over the infinite-state transition system of a pushdown system (à la (2)). In essence, the infinite-state transition system would be the result obtained from an infinite process of exhaustively inlining procedures into the main procedure of G. A finite-state automaton is then constructed to represent symbolically the infinite set of states that solves the ordinary reachability problem in the infinite-state system.

In contrast, this paper is about class (3). In this class of problems, the transition system has infinitely many states, but there is still a CFL constraint on the word spelled out along a path for the path to be accepted. A recent line of work [Pimpalkhare and Kincaid 2024a,b] successfully solved problems of this class and leveraged these solutions to compute summaries for recursive procedures using a class of vector-addition-system abstractions. This paper is an investigation into the formal foundations making such work possible.

A central tool in this line of work is Parikh's Theorem [Parikh 1966], which gives us a powerful abstraction for reasoning about CFL-reachability in infinite-state systems. It states that the set of *character counts* of each word in a context-free language is a semi-linear set. Verma et al. [2005] showed that one can construct a logical formula defining that set in linear time. Thus, when transitions commute, CFL-reachability problems reduce to querying whether a character count exists in the language such that the corresponding transitions, taken in any order, connect the two states. Past work [Bouajjani et al. 2003; Esparza et al. 2010] has leveraged this approach for program analysis over infinite commutative domains.

However, the abstract domains of [Pimpalkhare and Kincaid 2024a] and [Pimpalkhare and Kincaid 2024b] are not commutative, making their successful use of Parikh's Theorem surprising. These works operate by computing the Parikh image of an expanded version of the context-free language, which decomposes each sequence in the language into a finite number of phases. Each phase corresponds to a subsequence in which the transitions do commute, enabling Parikh-based reasoning to proceed piecewise.

This discussion raises the following question:

Is there a generalization of these works that extends the power of Parikh's Theorem to reason about systems that are only partially commutative?

In this paper, we answer this question affirmatively by identifying a new algebraic structure that combines a finite-state control component with a commutative-state component. This class is neither finite nor commutative, but remains amenable to Parikh-based analysis. Our key technical result is that the image of a context-free language under a homomorphism into an *almost-commuting monoid* is a semi-linear set and definable in polynomial time. This result represents an expansion of the possible applicability of Parikh's Theorem, and may be of independent interest.

The motivating application for this result is in solving CFL-reachability problems for transition systems represented by almost-commuting monoids, which we call *almost-commuting transition systems*. We identify several concrete examples of transition systems that non-trivially inhabit this algebraic structure, including a generalization of the abstract domain used by Pimpalkhare and Kincaid [2024a]. Our technical result yields an automatic decision procedure for solving CFL-reachability queries concerning all of our examples, and in general all almost-commuting transition systems meeting a *queryability* condition.

Our work can be viewed as a potential "backend" for automated program analysis. A client would have two obligations: (i) identifying a suitable family of almost-commuting transition systems S in which to model programs, and (ii) developing an abstraction procedure to translate a given program into an S system that over-approximates the behavior of the program. Our work then allows the computation of a summary formula describing the possible executions of the abstraction over a context-free set of control paths.

Contributions. The work described in the paper makes the following contributions:

- We introduce the notion of *almost-commuting monoids*. We show that in this algebraic structure, the image of any context-free language is semi-linear and efficiently definable. This result constitutes an extension of the applicability of Parikh's Theorem.
- We present several concrete examples of *transition systems* that are represented by almost-commuting monoids. The previous contribution then automatically provides us with a decision procedure for answering CFL-reachability queries for these transition systems.
- We explore closure properties of almost-commuting monoids and the implications of these properties for transition systems.

Organization. §2 presents the high-level insights that underlie our work using two examples. §3 recalls relevant background information. §4 defines *almost-commuting monoids* and *almost-commuting transition systems*, and develops our core formalisms. §5 presents several concrete examples of almost-commuting transition systems. §6 shows our technical result that we can efficiently compute a representation of the image of a context-free language under a homomorphism into an almost-commuting monoid, immediately solving CFL-reachability problems for all of the aforementioned examples. §7 presents closure properties of almost-commuting transition systems. §8 discusses related work. §9 concludes.

2 Overview

In this section, we introduce the central technical ideas of the paper using two motivating examples: a finite-domain gen-kill analysis and a numeric analysis of mutually recursive procedures. The first example illustrates the technical details of our approach in a simple and familiar setting. The second example presents, at a higher level, an instance in which our technique simplifies a complex numeric program into one that can be analyzed via Parikh-image techniques.

2.1 Gen-Kill Analysis

Gen-kill problems are a classical and well-studied framework for dataflow analysis [Reps et al. 1995]. Each program operation $s \in \Sigma$ is associated with a pair of sets $\mathrm{GEN}_s, \mathrm{KILL}_s \subseteq D$ over a finite domain D of dataflow facts, which represent the dataflow transformer

$$\lambda\sigma.(\sigma \setminus \mathrm{KILL}_s) \cup \mathrm{GEN}_s,$$

where σ denotes an abstract state. The composition of two operations $s; s'$ yields:

$$(\lambda\sigma.(\sigma\setminus\mathrm{KILL}_{s'})\cup\mathrm{GEN}_{s'})\circ(\lambda\sigma.(\sigma\setminus\mathrm{KILL}_s)\cup\mathrm{GEN}_s) = \lambda\sigma.(\sigma\setminus(\mathrm{KILL}_s\cup\mathrm{KILL}_{s'}))\cup((\mathrm{GEN}_s\setminus\mathrm{KILL}_{s'})\cup\mathrm{GEN}_{s'})$$

or in terms of the two pairs of GEN and KILL sets $\langle \text{GEN}_s, \text{KILL}_s \rangle$ and $\langle \text{GEN}_{s'}, \text{KILL}_{s'} \rangle$,

$$\text{GEN}_{s;s'} = (\text{GEN}_s \setminus \text{KILL}_{s'}) \cup \text{GEN}_{s'} \qquad \text{KILL}_{s;s'} = \text{KILL}_s \cup \text{KILL}_{s'}$$

In the interprocedural setting, we aim to obtain the cumulative GEN-KILL pairs for the words of the context-free language of paths through a procedure. Gen-kill problems are well-explored and have existing solutions that are widely deployed (e.g., IFDS [Reps et al. 1995]). Our approach has the benefit of computing a more precise solution: in particular, our method computes a characterization of the *set* of all L-path values, whereas previous methods compute the *join*-over-all-L-paths value. The extra precision comes with the cost of an exponential-space complexity. However, our goal in this section is not to introduce a competitive alternative to existing solutions, but to use a familiar context with particularly simple dataflow transformers to illustrate the key ideas of how Parikh images can be employed for almost-commuting transition systems.

The Parikh image of a CFL is a concise representation of the set of *character counts* of the language's words [Parikh 1966]. Parikh images have been used in program analysis to provide an exact characterization of the transformations performed along a program's (context-free) paths when all dataflow transformers commute (e.g., [Bouajjani et al. 2003, §3.2.4] [Esparza et al. 2010, §2.3.3]). For performing gen-kill analysis, this direction does not initially appear promising because the gen-kill transformers do not commute: namely, the GEN composition rule is non-commutative.

The formalization of our technique, here and in later sections of the paper, involves transforming a context-free grammar $\mathcal{G}$, creating a new grammar $\mathcal{G}'$ that has an expanded set of terminals and nonterminals, with each such symbol in $\mathcal{G}'$ labeled with certain dataflow quantities. One way to explain what these labels mean—and how the labels on one symbol relate to labels on another symbol—is by means of an *attribute grammar* [Knuth 1968].

An attribute grammar extends a traditional context-free grammar by attaching a set of *attributes* to each terminal or nonterminal symbol of the grammar and by using a set of *attribute equations* to propagate attribute values through syntax trees. The attributes of a symbol S are divided into *inherited* attributes and *synthesized* attributes. Inherited attributes transfer information from parent to child, or from a node to itself. Synthesized attributes transfer information from child to parent, from a node to a sibling, or from a node to itself. We assume that the terminal symbols of the grammar have no synthesized attributes, and that the root symbol of the grammar can have a special set of inherited attributes, called the *initial attributes*, with prescribed values.

The *output* attributes of a production $S \to S_1, \ldots, S_k$ consist of the synthesized attributes of the nonterminal S, plus the inherited attributes of all of the S_i's. The *input* attributes are the inherited attributes of S, plus the synthesized attributes of the S_i's. The grammar's *attribute-definition functions* define the output attributes in terms of the input attributes. Thus, at each tree node n of a syntax tree generated by the underlying context-free grammar, n's attributes are defined in terms of (i) attributes of its parent, its siblings, and itself, or (ii) attributes of its children and itself. We assume that the attribute grammar is *noncircular*, i.e., in every syntax tree generated by the underlying context-free grammar, no attribute is defined transitively in terms of itself.

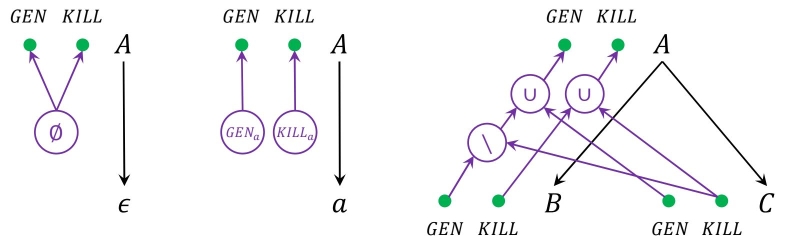
We are always interested in *consistently attributed* syntax trees. A tree T is consistently attributed if for every production instance p in T, each output attribute of p has a value equal to its attribute-definition function applied to the appropriate input attributes of p.[1] Henceforth, by "syntax tree" we mean a consistently attributed syntax tree.

It is well known how to give a context-free grammar that describes the set of paths through a program that respects the program's call-return structure [Reps 1998]. Below, we use capital

[1]Every syntax tree T generated by the underlying context-free grammar of a noncircular attribute grammar can be consistently attributed in time linear in $|T|$ by evaluating T's attribute instances in a topological order of the graph of the dependencies among the attribute instances.

letters for nonterminal symbols, and small letters for terminal symbols. The productions $A \longrightarrow B\ C$ and $A \longrightarrow a$ represent generic productions in such a grammar of paths in Chomsky normal form [Chomsky 1959].

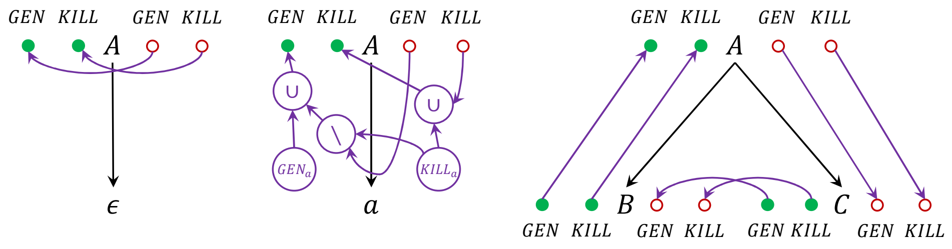
The flow of information among the attributes of a production's terminals and nonterminals can be depicted by means of *attribute-grammar diagrams*. The following diagrams show how the propagation of gen-kill information in a syntax tree that represents a program path can be captured by an attribute grammar that uses synthesized attributes to perform a bottom-up evaluation:



Productions are shown with the left-hand-side nonterminal at the top. Each synthesized attribute is shown as a solid green dot to the left of its associated nonterminal. (Each inherited attribute—used in subsequent diagrams—will be shown as a small red circle to the right of its nonterminal.) Attribute-definition functions are shown in purple, with operators shown inside circles. Constants are operators that have no incoming edges; an edge with no operator denotes the identity function.

With this attribute grammar, if we are given a syntax tree that represents, for example, an interprocedural path π from the entry of a procedure P to the exit of P, the values of the synthesized GEN and KILL attributes of the root node are the cumulative GEN and KILL sets for path π.

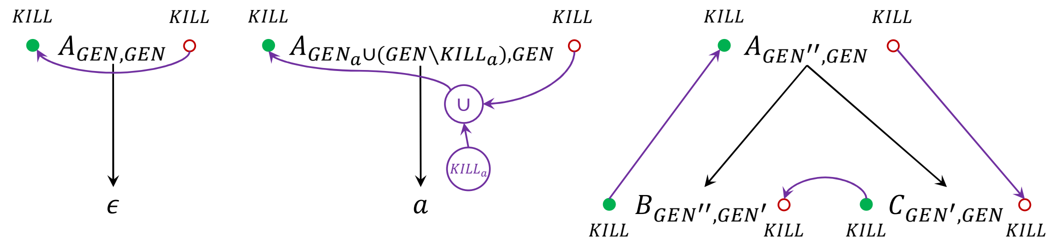
The pure bottom-up pattern of the attribute grammar shown above is not the only way to compute the desired quantities. An alternative is *right-to-left threading*, which uses both inherited and synthesized attributes, as depicted by the attribute grammar shown below:



(1)

(The ROOT nonterminal would have GEN and KILL initial attributes, which would be assigned ∅.)

We now come to the key transformation that allows Parikh images to be employed for almost-commuting transition systems. We create a new attribute grammar by labeling the nonterminals with *pairs* of GEN sets—essentially promoting before-and-after GEN values to the context-free-grammar portion of the attribute grammar (with the dependencies among their values in the old attribute grammar reflected in the GEN labels of the new attribute grammar). The result of the transformation creates productions of the form shown below:

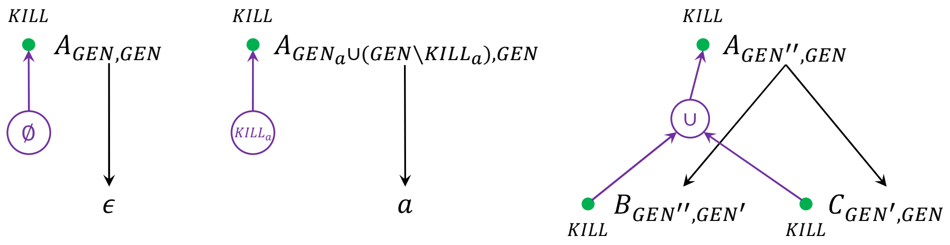


In the leftmost diagram, $A_{\text{GEN,GEN}}$ is a nonterminal labeled with a pair of sets that have the same value (indicated by the repeated label GEN) because the inherited GEN attribute is passed directly to the synthesized GEN attribute. There are $2^{|D|}$ possible GEN sets, and thus there would be $2^{|D|}$ productions of the form $A_{\text{GEN,GEN}} \to \epsilon$. In the middle diagram, GEN_a and KILL_a are known constants associated with terminal symbol a, and thus there would be $2^{|D|}$ productions of the form $A_{\text{GEN}_a \cup (\text{GEN} \setminus \text{KILL}_a),\text{GEN}} \to a$. In the rightmost diagram, each nonterminal symbol is labeled with a pair of possibly unequal GEN sets, but the repeated labels GEN, GEN′, and GEN″ impose equality constraints among the GEN-set labels of the different nonterminals consistently with how attributes are propagated in the set of diagrams labeled (1). Consequently, there would be $(2^{|D|})^3 = 2^{3|D|}$ productions of the form

$$A_{\text{GEN}'',\text{GEN}} \to B_{\text{GEN}'',\text{GEN}'}\ C_{\text{GEN}',\text{GEN}}.$$

At this point, the only operator in any attribute-definition function is $\cup$, which is commutative. Our transformations have, in essence, eliminated non-commutativity from all attribute-definition functions by blowing up the size of the underlying grammar by a factor that is exponential in $|D|$.

With one final transformation, we can convert back to an attribute grammar that (i) uses only commutative operators, and (ii) has only bottom-up dependencies among attributes.



Suppose that the cumulative GEN and KILL sets for a path π are G and K, respectively. With the transformed grammar, the syntax tree for π would have root nonterminal $\text{ROOT}_{\text{G},\emptyset}$, and the synthesized KILL attribute of the root node would be K.

By the sequence of transformations illustrated above, we obtain an attribute grammar that is amenable to analysis via Parikh images. More precisely, we are interested in the Parikh images of all languages of the form $\mathcal{L}(\text{ROOT}_{\text{G},\emptyset})$. The Parikh image of a specific language $\mathcal{L}(\text{ROOT}_{\text{G},\emptyset})$ precisely characterizes the set of cumulative gen-kill transformers with GEN set G. For instance, if there are three terminal symbols $\{a, b, c\}$, and we learn that the Parikh image of $\mathcal{L}(\text{ROOT}_{\text{G},\emptyset})$ is the set $\{\langle 2, 0, 0\rangle + \lambda_1\langle 5, 3, 0\rangle + \lambda_2\langle 1, 0, 5\rangle \mid 0 \le \lambda_1 \text{ and } 0 \le \lambda_2\}$, then the gen-kill transformers with GEN set G are exactly:[2]

$$\begin{array}{ll} \lambda\sigma.(\sigma \setminus \text{KILL}_a) \cup G & \lambda\sigma.(\sigma \setminus (\text{KILL}_a \cup \text{KILL}_b)) \cup G \\ \lambda\sigma.(\sigma \setminus (\text{KILL}_a \cup \text{KILL}_c)) \cup G & \lambda\sigma.(\sigma \setminus (\text{KILL}_a \cup \text{KILL}_b \cup \text{KILL}_c)) \cup G \end{array}$$

[2]For sets, because $\cup$ is idempotent, the count values 2, 3, and 5 do not generate additional transformers.

That is, the program contains at least one path for each of the transformers shown above, but has *no* paths for any of the following transformers:

$$\begin{array}{ll} \lambda\sigma.\sigma \cup G & \lambda\sigma.(\sigma \setminus \mathrm{KILL}_b) \cup G \\ \lambda\sigma.(\sigma \setminus \mathrm{KILL}_c) \cup G & \lambda\sigma.(\sigma \setminus (\mathrm{KILL}_b \cup \mathrm{KILL}_c)) \cup G \end{array}$$

In summary, this example illustrates how our framework allows Parikh's Theorem to be applied to problems not obviously amenable to such treatment, according to the following principle:

For CFL-reachability problems in which non-commutative transformers can be restructured so that all non-commutative operations are over a finite domain, one can track the effects of such operations in the nonterminals of an attribute grammar, thereby reducing the problem to a purely commutative one to which one can apply Parikh's Theorem.

2.2 A Numeric Program

Our grammar-transformation techniques can make challenging numerical analyses amenable to analysis via Parikh's Theorem by appealing to the underlying algebraic structure of the operations. For example, consider the following mutually recursive numerical programs:

```
def f(x):
    if (*):
        return 7
    else:
        return g(x + 1)
```

```
def g(x):
    if (*):
        return -x + 2
    else if (*):
        return f(-x)
    else:
        return f(x) + 1
```

These simple procedures could plausibly arise as numerical abstractions of more complex "real-world" programs. However, our goal is not to explain how such abstractions are derived, but rather how to compute summaries for them. We use this example to illustrate our method in a challenging setting, setting aside its possible concrete origins.

Suppose that we are interested in computing the input-output relation of the `f` procedure above. That is, we are interested in computing the set:

$$\{\langle \mathrm{in}, \mathrm{out}\rangle \mid \text{there exists a run such that } \mathrm{out} = f(\mathrm{in})\}$$

It is not clear that Parikh's Theorem can be applied here in any way, because the operations in our program do not commute. In particular, the operation of applying unary-minus used in the body of `g` (i.e., $\lambda z.\text{-}z$) does not commute with addition by a constant (e.g., $\lambda z.z + 2$). That is, $(\lambda z.\text{-}z) \circ (\lambda z.z + 2) = \lambda z.\text{-}(z + 2) \neq \lambda z.\text{-}z + 2 = (\lambda z.z + 2) \circ (\lambda z.\text{-}z)$.

For brevity, we will not spell out the steps of an explicit grammar transformation, relying on the reader to recognize the similarities between (i) the results of the grammar-transformation and nonterminal-renaming steps performed in §2.1, and (ii) the specialized procedures created below.

By unfolding the procedures in the example, we create an expanded cluster of mutually-recursive procedures with equivalent semantics to those above, but in which all operations commute. The unfolding is guided by the algebraic structure of the operations. In the original program, all operations are of the form $x' = r * x + o$ in which x and x' are the pre-state and post-state values respectively, $r \in \{-1, 0, 1\}$, and $o \in \mathbb{Z}$, and this form is preserved under composition. Analogously to the nonterminal-renaming step used in §2.1, we encode the finite multiplicative component r into procedure names, while modifying the additive component o based on the r context. This approach lets us reinterpret the program as a commutative system operating under a finite control skeleton.

```
def f_shell(x):
    if (*):
        return x + f_one()
    elif (*):
        return -x + f_neg()
    else:
        return f_zero()


def f_one():
    return 1 + g_one()
```

```
def f_neg():
    return -1 + g_neg()

def f_zero():
    if (*):
        return 7
    else:
        return g_zero()

def g_one():
    if (*):
        return f_neg()
    else:
        return f_one() + 1
```

```
def g_neg():
    if (*):
        return 2
    elif (*):
        return f_one()
    else:
        return f_neg() + 1


def g_zero():
    if (*):
        return f_zero()
    else:
        return f_zero() + 1
```

The f and g procedures have been unfolded into a shell procedure `f_shell`, which has the same input-output behavior as `f`, and six context-labeled procedures in which the contexts are `_one`, `_neg`, and `_zero`. The context tracks the effect of later operations on the current increment—i.e., how future manipulations of the x parameter (of the original program) affect the local increment. For example, because the body of `f_neg` presumes that calling `g_neg` will negate future increments, it negates the increment performed in procedure `f` (*decrementing* by 1 rather than *incrementing* by 1).

For example, consider the run of `f` with input $x = 3$ and returns -2 via the path:

`f(x)` ⟶ `g(x+1)` ⟶ `f(x+1)+1` ⟶ `g((x+1)+1)+1` ⟶ `(-((x+1)+1)+2)+1` ⟶ `-x+1` ⟶ `-2`

This run corresponds to the following run in `f_shell` with the same input-output behavior:

`f_shell(x)` ⟶ `-x+f_neg()` ⟶ `-x-1+g_neg()` ⟶ `-x-1+f_neg()+1`

⟶ `-x-1-1+g_neg()+1` ⟶ `-x-1-1+2+1` ⟶ `-x+1` ⟶ `-2`

Each run of `f_shell` contains (i) exactly one initial operation, which sets the context and adds in the value of x relevant to that context, (ii) a sequence of operations performed in `f_one`, `f_neg`, `f_zero`, `g_one`, `g_neg`, and `g_zero`, and (iii) exactly one constant return from `f_zero` or `g_neg`. All arithmetic operations in (ii) are increments to the top-level return variable, which commute with each other.

The occurrence of the "-x" term in the second branch of `f_shell` is not an instance of a non-commuting operation in the sequence of operations performed during the run. One should think of "-x" as preceding the sequence, and setting the context of what is being computed. Its role is similar to that of G in §2.1, where the Parikh image of a specific language $\mathcal{L}(\text{ROOT}_{G,\emptyset})$ precisely characterizes the set of cumulative gen-kill transformers with GEN set G. Thus, the input–output behavior of `f_shell` is fully determined by the Parikh set of operation counts in the context-free language of runs. Because `f_shell` is semantically equivalent to `f`, the transformation allows us to compute the input-output relation of `f`, as desired.

In contrast with the gen-kill analysis from §2.1, this example shows that the residual commutative component of an analysis problem can be an *infinite set*—in this case $\mathbb{Z}$. In gen-kill analysis, both the GEN and KILL components are subsets of a finite set D; the GEN sets are folded into the nonterminal names in the nonterminal-renaming steps, leaving only commutative operations on the finite domain of KILL sets. In the numeric-analysis example, the finite-valued r component is analogously folded into the procedure names, producing a program that has only commutative operations.

3 Background

A monoid $\langle M, \cdot, 1\rangle$ consists of a universe of elements M, a binary operator $(-)\cdot(-) : M \times M \to M$, and an identity element 1, where:

- (Associativity) For any $a, b, c \in M$ we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- (Identity) For any $a \in M$ we have $1 \cdot a = a \cdot 1 = a$

Presburger arithmetic [Presburger 1929] is a decidable first-order theory over the integers. A quantifier-free Presburger formula is a Boolean combination of linear inequalities of the form:

$$a_1x_1 + \cdots + a_nx_n \leq b,$$

where variables $x_1, \ldots, x_n$ range over the integers and a_i, b are integer constants. An existential Presburger formula has the form $\exists \mathbf{y}.\ \phi(\mathbf{x}, \mathbf{y})$, where ϕ is quantifier-free. A tuple $\mathbf{m} \in \mathbb{Z}^{|\mathbf{x}|}$ satisfies this formula if there exists some $\mathbf{y}$ making $\phi(\mathbf{m}, \mathbf{y})$ true, which we denote by $\mathbf{m} \models \exists \mathbf{y}.\ \phi(\mathbf{x}, \mathbf{y})$. Each formula defines the subset of $\mathbb{Z}^{|\mathbf{x}|}$ corresponding to its models $\{\mathbf{m} : \mathbf{m} \models \exists \mathbf{y}.\ \phi(\mathbf{x}, \mathbf{y})\}$.

A subset $S \subseteq \mathbb{Z}^n$ is called linear if it has the form:

$$\{\mathbf{b} + \lambda_1\mathbf{p}_1 + \cdots + \lambda_k\mathbf{p}_k \mid \lambda_1, \ldots, \lambda_k \in \mathbb{N}\}$$

where $\mathbf{b}, \mathbf{p}_1, \ldots, \mathbf{p}_k \in \mathbb{Z}^n$. A set is *semi-linear* if it is a finite union of linear sets. It is a classical result [Ginsburg and Spanier 1966, Theorem 1.3] that the semi-linear subsets of $\mathbb{Z}^n$ are exactly those definable by Presburger formulas.

A labeled transition system over a finite alphabet Σ is a pair $T = \langle S, \{\to_s \subseteq S \times S \mid s \in \Sigma\}\rangle$ in which S is the state space of T and each $\to_s \subseteq S \times S$ is a transition relation between states. We write $\rho \to_s \rho'$ if $\langle \rho, \rho'\rangle \in \to_s$. For a language $L \subseteq \Sigma^*$, the L-reachability relation of T is:

$$\left\{\langle \rho_0, \rho_n\rangle \mid \exists \rho_1, \ldots, \rho_{n-1} \in S. \exists s_1 \ldots s_n \in L. \rho_0 \to_{s_1} \rho_1 \to_{s_2} \cdots \to_{s_n} \rho_n\right\}$$

A context-free grammar $G = \langle N, \Sigma, R, S\rangle$ consists of a finite set of non-terminals N, a finite alphabet of terminals Σ, a finite set of production rules $R \subseteq N \times (N \cup \Sigma)^*$ and a start non-terminal $S \in N$. We write production rules as $A \to w$; the application of this rule translates words of the form $w_1Aw_2 \in (N \cup \Sigma)^*$ into w_1ww_2. The language of G, denoted by $\mathcal{L}(G)$, is the set of all terminal words $w \in \Sigma^*$ produced by repeated application of production rules in R to start symbol S.

A context-free grammar $G = \langle N, \Sigma, R, S\rangle$ is in Chomsky Normal Form if all production rules in R take one of the following forms: $A \to BC$ (for $A, B, C \in N$), $A \to a$ (for $A \in N$ and $a \in \Sigma$), or $A \to \epsilon$ (for $A \in N$). Any context-free grammar can be transformed into a context-free grammar in Chomsky Normal Form that recognizes the same language. The transformation can be performed in time linear in the size of the grammar [Chomsky 1959].

Let $\Sigma = \{s_1, \ldots, s_n\}$ be a finite alphabet. The Parikh image of a word $w \in \Sigma^*$ is a vector $\pi(w) \in \mathbb{Z}^n$ such that $\pi(w)_i$ is the number of occurrences of s_i in w. The Parikh image of a language is the set of all Parikh images $\pi(w)$ of words w in the language. Parikh's Theorem [Parikh 1966] states that the Parikh image of any context-free language is a semi-linear set. A representation of the Parikh image of a context-free language can be constructed in linear time with respect to the size of the grammar [Verma et al. 2005]. That is, for any context-free grammar G, we can efficiently create a Presburger formula ϕ_G such that:

$$\{\mathbf{m} \mid \mathbf{m} \models \phi_G\} = \{\pi(w) \mid w \in \mathcal{L}(G)\}$$

Because ϕ_G is a Presburger formula, we can use standard SMT-solving techniques to perform operations, such as (i) satisfiability checking, and (ii) intersection with other Presburger-definable sets.

4 Technical Definitions

This section defines *almost-commuting monoids* (ACMs). The following sections will provide examples of such monoids and examine their properties.

An ACM is a monoid $\langle M, \cdot, 1\rangle$ whose elements can be fully described as the image of a finite set $F \subseteq M$ under a right action $\triangleleft$ by a commutative monoid $\langle C \subseteq \mathbb{Z}^n, +, 0\rangle$. The action $\triangleleft$ respects both the monoid structure of M (via a kind of "reassociativity" rule for $\cdot$ and $\triangleleft$), and the additive structure of C (via a kind of "mixed" associativity rule for $\triangleleft$ and $+$).

Definition 1 (Almost-Commuting Monoid)**.** A monoid $\langle M, \cdot, 1\rangle$ is almost-commuting if there exists a finite subset F of M, a commutative integer monoid $\langle C \subseteq \mathbb{Z}^n, +, 0\rangle$, and an action $\triangleleft : M \times C \longrightarrow M$ such that:

(1) $M = \{f \triangleleft c : f \in F, c \in C\}$
(2) (Reassociativity) For all $m_1, m_2 \in M$ and all $c \in C$, we have $(m_1 \cdot m_2) \triangleleft c = m_1 \cdot (m_2 \triangleleft c)$
(3) (Action) For all $m \in M$ and all $c_1, c_2 \in C$, we have $(m \triangleleft c_1) \triangleleft c_2 = m \triangleleft (c_1 + c_2)$ and $m \triangleleft 0 = m$
(4) There is an algorithm that, given $m \in M$, computes $f \in F$ and $c \in C$ such that $m = f \triangleleft c$.

(Note the mnemonic: F and (possibly subscripted) f for "finite"; C and (possibly subscripted) c for "commutative.")

Example 4.1. This example is a simple instance of an almost-commuting monoid. Consider the monoid $\langle \{0, 1\} \times \mathbb{Z}, \cdot, \langle 1, 0\rangle\rangle$ in which $\cdot$ is defined as:

$$\langle a, b\rangle \cdot \langle a', b'\rangle = \langle aa', a'b + b'\rangle .$$

This monoid is an almost-commuting monoid, with finite subset $F = \{\langle 0, 0\rangle, \langle 1, 0\rangle\}$, commutative monoid $\langle \mathbb{Z}, +, 0\rangle$, and action $\triangleleft$ defined by $\langle a, b\rangle \triangleleft c = \langle a, b + c\rangle$. The first three conditions can be straightforwardly checked; the algorithm for the fourth condition is component-wise projection.

Our goal is to compute the image of context-free languages under homomorphisms into almost-commuting monoids. Effective representation of such images hinges on identifying a class of sets that is both expressive and tractable. semi-linear sets, central to Parikh's Theorem, fit this role, and we represent them using formulas in Presburger arithmetic.

Definition 2. Let M be an ACM parametrized by finite subset F, commutative monoid $\langle C \subseteq \mathbb{Z}^n, +, 0\rangle$, and action $\triangleleft : M \times C \longrightarrow M$. A *linear set* is a set of the form:

$$S = \{f \triangleleft (b + \lambda_1 p_1 + \cdots + \lambda_k p_k) \mid \lambda_i \in \mathbb{N}\}$$

for $f \in F$ and $b, p_1, \ldots, p_k \in C$. semi-linear sets are finite unions of linear sets.

Definition 3. Let M be an ACM parametrized by finite subset $F = \left\{f_1, \ldots, f_{|F|}\right\}$, commutative monoid $\langle C \subseteq \mathbb{Z}^n, +, 0\rangle$, and action $\triangleleft : M \times C \longrightarrow M$. We say that Presburger formula $\phi(x_0, \ldots, x_n)$ *weakly defines* a subset $S \subseteq M$ if

$$S = \{f_i \triangleleft (c_1, \ldots, c_n) \mid \phi(i, c_1, \ldots, c_n) \text{ holds}\} .$$

Presburger formula $\phi(x_0, \ldots, x_n)$ *strongly defines* subset $S \subseteq M$ if

$$\{\langle f_i, c_1, \ldots, c_n\rangle \mid \phi(i, c_1, \ldots, c_n) \text{ holds}\} = \{\langle f, c_1, \ldots, c_n\rangle \mid f \triangleleft (c_1, \ldots, c_n) \in S\} .$$

The distinction between a formula strongly defining a subset S versus weakly defining S is that the former means that the formula recognizes *all* $\langle f, c\rangle$ such that $f \triangleleft c$ lies within S, whereas the latter means that the formula recognizes *at least one* $\langle f, c\rangle$ such that $f \triangleleft c = m$ for each element m of S. As expected, a formula that strongly defines a subset also weakly defines it. Weakly definable sets coincide with semi-linear sets over almost-commuting monoids.

The above definitions allow us to use Presburger formulas to define semi-linear subsets of almost-commuting monoids. Note that what the formulas define is not quite the same as the notion of definability in the traditional sense: our formulas do not identify the elements of a subset of an ACM, they identify a set of *representations* of these elements. We provide several examples of ACMs that model transition systems in §5 , and in §6, we show that even though our formulas identify representations of sets of interest, they still allow CFL-reachability queries to be answered.

Our central technical result shows that we can compute a Presburger formula that weakly defines the image of a context-free language under a monoid homomorphism into an ACM. That is, let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language over a finite alphabet Σ, let $v : \Sigma \to M$ be a valuation of that alphabet, and let $v^* : \Sigma^* \to M$ be v extended to a monoid homomorphism over ACM $\langle M, \cdot, 1\rangle$. Then, the set:

$$\{v^*(w) \mid w \in \mathcal{L}(G)\}$$

is semi-linear, and we can compute a Presburger formula that weakly defines it in polynomial time.

This result generalizes Parikh's Theorem beyond commutative monoids, and may be of independent technical interest. Our motivation for this generalization is its application to answering context-free-language reachability queries involving transition systems that are represented by almost-commuting monoids.

Definition 4 (Almost-Commuting Transition System (ACTS))**.** A labeled transition system $\langle S, \{\to_s \subseteq S \times S \mid s \in \Sigma\}\rangle$ over finite alphabet Σ is *almost-commuting* if the monoid $\langle M, \cdot, \mathbb{I}\rangle$ is almost-commuting, in which:

- M is the closure of $\{\to_s \mid s \in \Sigma\}$ under relational composition,
- $\cdot$ denotes relational composition, and
- $\mathbb{I} \subseteq S \times S$ is the identity relation.

Our main technical result in the context of transition systems means that for any context-free language $\mathcal{L}(G)$ and any almost-commuting transition system T, we can compute a representation of all transition relations corresponding to $\mathcal{L}(G)$ paths through T. That is, we can weakly define the set:

$$\left\{\to_{s_1} \circ \cdots \circ \to_{s_n} \mid s_1 \ldots s_n \in \mathcal{L}(G)\right\}$$

However, this result is not a sufficient representation to decide $\mathcal{L}(G)$-reachability for T because we are unable to test for membership. That is, given two states of the transition system, we do not have a way to test if any of our transition relations contain the states. To bridge this gap, we introduce the following definition:

Definition 5. Let $T = \langle S, \{\to_s \mid s \in \Sigma\}\rangle$ be a labeled transition system over finite alphabet Σ and let M be the closure of $\{\to_s \mid s \in \Sigma\}$ under relational composition. T is *queryable* if there is an algorithm that computes, for any states $\rho, \rho' \in S$, a Presburger formula strongly defining the set

$$\{\to \in M \mid \rho \to \rho'\}$$

With this condition in hand, given two states ρ, ρ' of the transition system, we may conjoin the formula that strongly defines the set of transitions between the states with the formula that weakly defines the set of $\mathcal{L}(G)$ paths through the transition system to produce a formula that weakly defines the set:

$$\left\{\rightarrow_{s_1} \circ \cdots \circ \rightarrow_{s_n} \mid \rho \rightarrow_{s_1} \circ \cdots \circ \rightarrow_{s_n} \rho', s_1 \ldots s_n \in \mathcal{L}(G)\right\}$$

CFL-reachability between ρ and ρ' then reduces to checking satisfiability of this formula.

In the following section, we present several concrete examples of almost-commuting transition systems; we also show that these transition systems are queryable. In §6, we formally prove that we have a decision procedure for CFL-reachability problems over queryable ACTS.

5 Examples of ACTS

This section provides examples of almost-commuting transition systems. Theorem 4 in §6 implies that we immediately have a decision procedure for context-free reachability problems involving all of these systems. §7 shows that ACTS are closed under several operations, enabling us to generalize all of these examples to arbitrary dimension and allowing us to build complex ACTS out of simple components.

In practice, the systems described below can serve as abstract models for program analyses. To instantiate such an analysis, one specifies (i) a family of almost-commuting transition systems capturing the desired abstract semantics, and (ii) an abstraction procedure translating a program into an ACTS that over-approximates its behavior. Our results then automatically produce a symbolic summary formula that describes all executions of the abstraction along context-free control paths, yielding an over-approximate summary of the original program.

5.1 Finite Monoid Affine Vector Addition Systems

Vector Addition Systems are a broad class of transition systems that additively manipulate a vector of counters, and have historically been used to model concurrent systems [Karp and Miller 1969]. VAS are classically defined over counters in $\mathbb{N}$, but here we work with their integer-valued extension, where counters range over $\mathbb{Z}$. Finite Monoid Affine Vector Addition Systems (FMAVAS) are a generalization of integer VAS that were introduced in [Blondin et al. 2021], drawing on prior work on computing closures of affine relations [Boigelot 1998; Finkel and Leroux 2002]. Their work showed that the regular-language reachability relation of FMAVAS is computable via a reduction to the regular-language reachability relation of vector addition systems. We show here that FMAVAS are queryable almost-commuting transition systems, and thereby show that we can solve CFL-reachability problems over them.

Definition 6. A *finite monoid affine vector addition system* over a finite alphabet Σ is a labeled transition system $\langle \mathbb{Z}^n, \{\rightarrow_s \mid s \in \Sigma\}\rangle$ equipped with a finite monoid $\langle L, \circ, \mathbb{I}\rangle$ of linear functions $\mathbb{Z}^n \rightarrow \mathbb{Z}^n$, such that for each label $s \in \Sigma$, the transition relation $\rightarrow_s$ satisfies:

$$x \rightarrow_s x' \iff x' = f(x) + o \quad \text{for some } f \in L,\ o \in \mathbb{Z}^n$$

The program in §2.2 is an FMAVAS. The following is an example of a 2-dimensional FMAVAS.

Example 5.1. Let the alphabet Σ consist of a single character a . Let $\left\langle \mathbb{Z}^2, \{\rightarrow_a\}\right\rangle$ be a FMAVAS in which:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow_a \begin{pmatrix} x' \\ y' \end{pmatrix} \Leftrightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

This FMAVAS is equipped with finite monoid $\langle L, \circ, \mathrm{id}\rangle$ in which:

$$L = \left\{ \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathrm{id} \right\}$$

Finally, we observe that every FMAVAS is an almost-commuting transition system, and that they are queryable.

Lemma 1. Every finite-monoid affine vector addition system is an almost commuting transition system.

Proof. Let $\langle \mathbb{Z}^n, \{\to_s \mid s \in \Sigma\}\rangle$ be an FMAVAS over finite alphabet Σ, equipped with a finite monoid $\langle L, \circ, \mathbb{I}\rangle$ of linear functions. Let M denote the closure of $\{\to_s \mid s \in \Sigma\}$ under relational composition $\cdot$, and let $\mathbb{I}$ be the identity relation on $\mathbb{Z}^n$. We will show that $\langle M, \cdot, \mathbb{I}\rangle$ is an almost-commutative monoid.

For each $f \in L$, define $R_f = \{(x, x') \mid x' = f(x)\}$. Let $F = \{R_f \mid f \in L\}$ and let $C = \mathbb{Z}^n$. Define the action $\triangleleft : M \times C \to M$ by:

$$m \triangleleft c = \{(x, x' + c) \mid (x, x') \in m\}\,.$$

To verify Condition (1) of Definition 1, observe that every generator $\to_s$ of M is of the form:

$$\{(x, x') \mid x' = f(x) + o\} = R_f \triangleleft o$$

for some $f \in L$, $o \in C$. Additionally, we have the following closure property: for all $f, f' \in L$ and $o, o' \in C$:

$$(R_f \triangleleft o) \cdot (R_{f'} \triangleleft o') = R_{f' \circ f} \triangleleft (f'(o) + o'),$$

Because $\left\{R_f \triangleleft o : R_f \in F, o \in \mathbb{Z}\right\}$ contains the generators of M and is closed under composition, it is equal to M. Thus, every element of M is of the form $R_f \triangleleft o$ for some $R_f \in F$ and $o \in \mathbb{Z}^n$.

Conditions (2) and (3) follow directly from the definition of $\triangleleft$. Condition (4) is straightforward: the affine transformation $x' = f(x) + o$ decomposes into $R_f \in F$ and $o \in C$. □

Lemma 2. Finite-monoid affine vector addition systems are queryable.

Proof. Let $\langle \mathbb{Z}^n, \{\to_s \mid s \in \Sigma\}\rangle$ be an FMAVAS over finite alphabet Σ, equipped with a finite monoid $\left\langle L = \left\{f_1 \ldots f_{|L|}\right\}, \circ, \mathbb{I}\right\rangle$ of linear functions. Let M denote the closure of $\{\to_s \mid s \in \Sigma\}$ under relational composition.

Consider any states $\rho, \rho' \in \mathbb{Z}^n$. We can define the set $\{\to \in M \mid \rho \to \rho'\}$ with the following formula:

$$\bigvee_{i=1}^{|L|} x_0 = i \land \rho' = f_i(\rho) + (x_1, \ldots, x_n),$$

where x_0 encodes the choice of $f_i \in L$ and $(x_1, \ldots, x_n)$ encodes the offset vector in $\mathbb{Z}^n$. This formula thus strongly defines (Definition 3) the required subset of M. □

5.2 Natural Offset Max-Plus Linear Systems

Max-Plus Linear Systems arise from the use of the max-plus algebra, where “addition” is replaced by taking maximums and “multiplication” by addition. Certain discrete-event systems, including production lines, railway networks, and communication protocols, can be modeled within this algebra [De Schutter and van den Boom 2008]. Max-Plus Linear Systems are transition systems in which the transitions correspond to the analogue of affine updates in the max-plus algebra. We show here that a subclass of Max-Plus Linear Systems are queryable almost-commuting transition systems.

Definition 7. A *natural offset max-plus linear system* over a finite alphabet Σ is a labeled transition system $\langle \mathbb{Z}, \{\to_s \mid s \in \Sigma\}\rangle$ equipped with functions $a : \Sigma \to \mathbb{Z}$ and $b : \Sigma \to \mathbb{N}$ such that for each label $s \in \Sigma$, the transition relation $\to_s$ satisfies:

$$x \to_s x' \iff x' = \max(x, a(s)) + b(s)$$

We now observe that these transition systems are almost-commuting and queryable.

Theorem 1. Every Natural Offset Max-Plus Linear System is an almost commuting transition system.

Proof. Let $\langle \mathbb{Z}, \{\rightarrow_s \mid s \in \Sigma\}\rangle$ be a natural offset max-plus linear system equipped with functions $a : \Sigma \rightarrow \mathbb{Z}$ and $b : \Sigma \rightarrow \mathbb{N}$. Let M denote the closure of $\{\rightarrow_s \mid s \in \Sigma\}$ under relational composition $\cdot$ and let $\mathbb{I}$ be the identity relation on $\mathbb{Z}$. We will show that $\langle M, \cdot, \mathbb{I}\rangle$ is an almost-commuting monoid.

Let I be the closed interval $[\min_s a(s), \max_s a(s)]$ and define $R_v = \{(x, x') \mid x' = \max(x, v)\}$ for all $v \in I$.

We define:

$$F = \{R_v \mid v \in I\} \quad C = \mathbb{N} \quad m \triangleleft c = \{(x, x' + c) \mid (x, x') \in m\}$$

Observe that for every generator $\rightarrow_s$ of M, we have that:

$$\rightarrow_s = R_{a(s)} \triangleleft b(s)$$

We now show closure under composition. For all $R_v, R_{v'} \in F$ and $c, c' \in \mathbb{N}$:

$$\begin{aligned} R_v \triangleleft c \cdot R_{v'} \triangleleft c' &= \{\langle x, x''\rangle \mid x' = \max(x, v) + c, x'' = \max(x', v') + c'\} \\ &= \{\langle x, x''\rangle \mid x'' = \max(\max(x, v) + c, v') + c'\} \\ &= \{\langle x, x'\rangle \mid x' = \max(\max(x, v) + c, v') + c'\} \\ &= \{\langle x, x'\rangle \mid x' = \max(\max(x + c, v + c), v') + c'\} \\ &= \{\langle x, x'\rangle \mid x' = \max(x + c, \max(v + c, v')) + c'\} \\ &= \{\langle x, x'\rangle \mid x' = \max(x, \max(v, v' - c)) + c + c'\} \\ &= R_{\max(v, v' - c)} \triangleleft (c + c') \end{aligned}$$

Because $v, v' \in I$ and $c \in \mathbb{N}$, $\max(v, v' - c)$ also lies in I. Hence, we have that compositions remain in the form $R_v \triangleleft c$, and thus by structural induction over M we have that every element therein can be represented in that form (Condition (1) of Definition 1). Conditions (2) and (3) follow straightforwardly from the definition of $\triangleleft$. Condition (4) is straightforward: transformer $x' = max(x, a) + b$ is associated with $R_a \in F$ and $b \in \mathbb{N}$. ◻

Lemma 3. Natural Offset Max-Plus Linear Systems are queryable.

Proof. Let $\langle \mathbb{Z}, \{\rightarrow_s \mid s \in \Sigma\}\rangle$ be a natural offset max-plus linear system equipped with functions $a : \Sigma \rightarrow \mathbb{Z}$ and $b : \Sigma \rightarrow \mathbb{N}$. Let M denote the closure of $\{\rightarrow_s \mid s \in \Sigma\}$ under relational composition. Let $l = \min_s a(s)$ and $r = \max_s a(s)$. Let the finite subset of M be denoted $F = \{R_l, \ldots, R_r\}$, where $R_v = \{(x, x') \mid x' = \max(x, v)\}$ for all $v \in [l, r]$.

Consider any states $\rho, \rho' \in \mathbb{Z}$. We can define the set $\{\rightarrow \in M \mid \rho \rightarrow \rho'\}$ with the following formula:

$$\bigvee_{i=1}^{|F|} x_0 = i \wedge \rho' = \max(\rho, l + i - 1) + x_1$$

where x_0 encodes the choice of the threshold in the max, and x_1 encodes the offset. Note that max can be straightforwardly encoded into Presburger arithmetic. ◻

5.3 Parity-Guarded Systems

Parity-Guarded Systems are transition systems whose update rule depends on the parity of the current state. They are a simple example of how guarded updates can be accommodated within our framework. We show here that Parity-Guarded Systems are *dual* ACTS.

Definition 8. A *parity-guarded system* over a finite alphabet Σ is a labeled transition system $\langle \mathbb{Z}, \{\rightarrow_s \mid s \in \Sigma\}\rangle$ equipped with functions $a, b : \Sigma \rightarrow \mathbb{Z}$ such that for each label $s \in \Sigma$, the transition relation $\rightarrow_s$ satisfies:

$$x \rightarrow_s x' \Leftrightarrow x' = \text{if } x \text{ is even then } x + a(s) \text{ else } x + b(s)$$

Parity-Guarded Systems follow a related structure to ACTS.

Definition 9. A *dual* almost-commuting monoid is a monoid $\langle M, \cdot, 1\rangle$ such that there exists a finite subset F of M, a commutative monoid $\langle C, +, 0\rangle$, and a function $\triangleleft : M \times C \rightarrow M$ such that:

(1) Conditions (1), (3), and (4) of Definition 1 are met
(2) For all $m_1, m_2 \in M$ and all $c \in C$, we have $(m_1 \cdot m_2) \triangleleft c = (m_1 \triangleleft c) \cdot m_2$ and $m_1 \triangleleft 0 = m_1$

Definition 10. A *dual* almost-commuting transition system is defined analogously to an ACTS with "dual ACM" in place of "ACM".

Theorem 2. Every Parity-Guarded System is a dual almost-commuting transition system.

Proof. Let $\langle \mathbb{Z}, \{\rightarrow_s \mid s \in \Sigma\}\rangle$ be a parity-guarded system equipped with functions $a, b : \Sigma \rightarrow \mathbb{Z}$. Let M denote the closure of $\{\rightarrow_s \mid s \in \Sigma\}$ under relational composition $\cdot$ and let $\mathbb{I}$ be the identity relation on $\mathbb{Z}$. We will show that $\langle M, \cdot, \mathbb{I}\rangle$ is a dual ACM.

We define:

$$F = \left\{ \begin{array}{c} f_{00} = \{\langle x, x'\rangle \mid x' = \text{if } x \text{ is even then } x \text{ else } x\}, \\ f_{10} = \{\langle x, x'\rangle \mid x' = \text{if } x \text{ is even then } x + 1 \text{ else } x\}, \\ f_{01} = \{\langle x, x'\rangle \mid x' = \text{if } x \text{ is even then } x \text{ else } x + 1\}, \\ f_{11} = \{\langle x, x'\rangle \mid x' = \text{if } x \text{ is even then } x + 1 \text{ else } x + 1\} \end{array} \right\}$$

$$C = \langle \mathbb{Z} \times \mathbb{Z}, +, \langle 0, 0\rangle\rangle \text{ where } \langle a, b\rangle + \langle c, d\rangle = \langle a + c, b + d\rangle$$

$$m \triangleleft \langle c_e, c_o\rangle = \{\langle x, x''\rangle \mid \langle x, x'\rangle \in m, x'' = \text{if } x \text{ is even then } x' + 2c_e \text{ else } x' + 2c_o\}$$

Observe that for every generator $\rightarrow_s$ of M we have:

$$\rightarrow_s = \{\langle x, x'\rangle \mid x' = \text{if } x \text{ is even then } x + (a(s)\%2) \text{ else } x + (b(s)\%2)\} \triangleleft \left\langle \left\lfloor \frac{a(s)}{2} \right\rfloor, \left\lfloor \frac{b(s)}{2} \right\rfloor \right\rangle$$

We investigate composition $f \triangleleft \langle c_e, c_o\rangle \cdot f' \triangleleft \langle c'_e, c'_o\rangle$ by case analysis over f and f'. In the following, if x or y appears in the subscript of an f, the equation holds regardless of whether the x or y is replaced with a 1 or a 0.

$$\begin{aligned}
f_{00} \triangleleft \langle c_e, c_o\rangle \cdot f_{xy} \triangleleft \langle c'_e, c'_o\rangle &= f_{xy} \triangleleft \langle c_e + c'_e, c_o + c'_o\rangle \\
f_{01} \triangleleft \langle c_e, c_o\rangle \cdot f_{0x} \triangleleft \langle c'_e, c'_o\rangle &= f_{01} \triangleleft \langle c_e + c'_e, c_o + c'_e\rangle \\
f_{01} \triangleleft \langle c_e, c_o\rangle \cdot f_{1x} \triangleleft \langle c'_e, c'_o\rangle &= f_{10} \triangleleft \langle c_e + c'_e, c_o + c'_e + 1\rangle \\
f_{10} \triangleleft \langle c_e, c_o\rangle \cdot f_{x0} \triangleleft \langle c'_e, c'_o\rangle &= f_{10} \triangleleft \langle c_e + c'_o, c_o + c'_o\rangle \\
f_{10} \triangleleft \langle c_e, c_o\rangle \cdot f_{x1} \triangleleft \langle c'_e, c'_o\rangle &= f_{01} \triangleleft \langle c_e + c'_o + 2, c_o + c'_o\rangle \\
f_{11} \triangleleft \langle c_e, c_o\rangle \cdot f_{00} \triangleleft \langle c'_e, c'_o\rangle &= f_{11} \triangleleft \langle c_e + c'_o, c_o + c'_e\rangle \\
f_{11} \triangleleft \langle c_e, c_o\rangle \cdot f_{01} \triangleleft \langle c'_e, c'_o\rangle &= f_{01} \triangleleft \langle c_e + c'_o + 2, c_o + c'_e\rangle \\
f_{11} \triangleleft \langle c_e, c_o\rangle \cdot f_{10} \triangleleft \langle c'_e, c'_o\rangle &= f_{10} \triangleleft \langle c_e + c'_o, c_o + c'_e + 1\rangle \\
f_{11} \triangleleft \langle c_e, c_o\rangle \cdot f_{11} \triangleleft \langle c'_e, c'_o\rangle &= f_{00} \triangleleft \langle c_e + c'_o + 2, c_o + c'_e + 1\rangle
\end{aligned}$$

Therefore, we have that composition retains the structure $f \triangleleft c$ and that therefore by structural induction all elements within M are representable in that form (Condition (1) of Definition 1).

Condition (3) of Definition 1 follows from the definition of $\triangleleft$, and Condition (4) can be computed straightforwardly from the transformer. Finally, note from the case analysis above that $(m_1 \cdot m_2) \triangleleft c = (m_1 \triangleleft c) \cdot m_2$. Thus, we have that parity-guarded systems are dual ACTS. □

Lemma 4. Parity-Guarded Systems are queryable.

Proof. Let $\langle \mathbb{Z}, \{\rightarrow_s \mid s \in \Sigma\}\rangle$ be a parity-guarded system. Let M denote the closure of $\{\rightarrow_s \mid s \in \Sigma\}$ under relational composition $\cdot$ and let $\mathbb{I}$ be the identity relation on $\mathbb{Z}$. Denote the finite subset $F = \{f_0 = f_{00}, f_1 = f_{01}, f_2 = f_{10}, f_3 = f_{11}\}$. In the following formula, the x_0 variable is a selector variable over these elements: when $x_0 = i$, the formula is selecting f_i.

Consider any states $\rho, \rho' \in \mathbb{Z}$. We can define the set $\{\rightarrow \in M \mid \rho \rightarrow \rho'\}$ with the following formula, in which x_1 and x_2 refer to the even-case and odd-case increments of the commutative component respectively:

$$\exists y. \rho = 2y \land (((x_0 = 0 \lor x_0 = 1) \land \rho' = \rho + 2x_1) \lor ((x_0 = 2 \lor x_0 = 3) \land \rho' = \rho + 1 + 2x_1)) \lor$$
$$\exists y. \rho = 2y + 1 \land (((x_0 = 0 \lor x_0 = 2) \land \rho' = \rho + 2x_2) \lor ((x_0 = 1 \lor x_0 = 3) \land \rho' = \rho + 1 + 2x_2))$$

in which x_0 selects a base relation in F, and (x_1, x_2) encode the even- and odd-case offsets of the commutative component. □

6 Context-Free Language Reachability of ACTS

This section presents the main result of this paper: that we can efficiently define the image of context-free languages under homomorphisms into an almost-commuting monoid. In the context of transition systems, this result means that we can answer context-free-language reachabilty queries about queryable almost-commuting transition systems. Our results are stated in the following theorems:

Theorem 3. Let $\langle M, \cdot, 1\rangle$ be an almost-commuting monoid, let Σ be a finite alphabet, and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Let $v : \Sigma \rightarrow M$ be a valuation function and let $v^* : \Sigma^* \rightarrow M$ be that valuation function extended to a monoid homomorphism. Then, the set

$$\{v^*(w) \mid w \in \mathcal{L}(G)\}$$

is semi-linear and can be weakly defined in time $O(|G||F|^3)$ where $F \subseteq M$ is the finite subset of M.

Theorem 4. Let T be a queryable ACTS defined over a finite alphabet Σ, and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Then there is a decision procedure to query the $\mathcal{L}(G)$-reachability relation of T for membership.

We present our construction for Theorem 3 diagrammatically in §6.1, and rigorously in §6.2.

6.1 Theorem 3 in Attribute-Grammar Diagrams

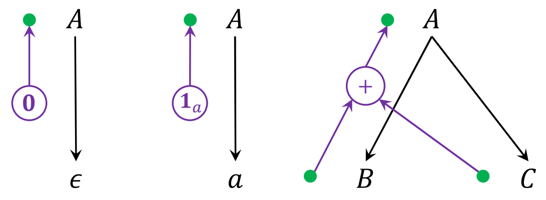
In this section, we give a proof of Theorem 3, using attribute-grammar diagrams in the style of §2.1 to portray graphically the problem-transformation steps that establish the theorem. Readers who prefer a proof without such visual aids should skip to §6.2.

We begin by formulating the computation of the Parikh image of a context-free language using attribute grammars. Parikh's Theorem establishes that images of context-free languages computed via such attribute grammars are semi-linear and efficiently definable. We then formulate the computation of the image of a context-free language in an almost-commuting monoid using attribute grammars. We apply a sequence of equivalence-preserving rewrites to these attribute grammars to produce one that mirrors that of the Parikh image. We thereby show that we can compute the image of a context-free language in an almost-commuting monoid via the Parikh image of an expanded version of the grammar.

Without loss of generality, we can assume that the context-free grammar G is in Chomsky Normal Form. For each $a \in \Sigma$, let $\mathbf{1}_a : \Sigma \longrightarrow \mathbb{N}$ be the function that maps a to 1 and all other characters $a' \neq a$ to 0. Let $\mathbf{0} : \Sigma \longrightarrow \mathbb{N}$ denote the function $\lambda\sigma.0$. We can define an addition operation for such functions as follows:

$$+ : (\Sigma \longrightarrow \mathbb{N}) \times (\Sigma \longrightarrow \mathbb{N}) \longrightarrow (\Sigma \longrightarrow \mathbb{N}) \qquad g_1 + g_2 = \lambda\sigma.g_1(\sigma) + g_2(\sigma).$$

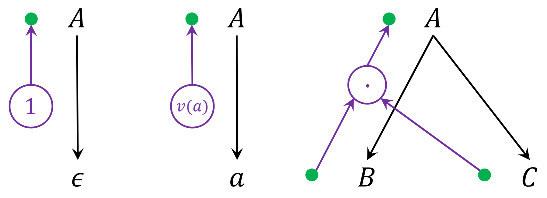
The following diagram depicts the form of an attribute grammar that computes the Parikh image of each word in $\mathcal{L}(G)$.



Each nonterminal propagates the Parikh image of the sub-word that it generates to its parent non-terminal. Nonterminals that derive the empty string ϵ pass up the $\mathbf{0}$ function; nonterminals that derive a single character a pass up $\mathbf{1}_a$; all other nonterminals pass up the sum of the attributes computed by their children. Parikh's Theorem implies that the set of attribute values that are computed via the above strategy is semi-linear and definable in polynomial time.

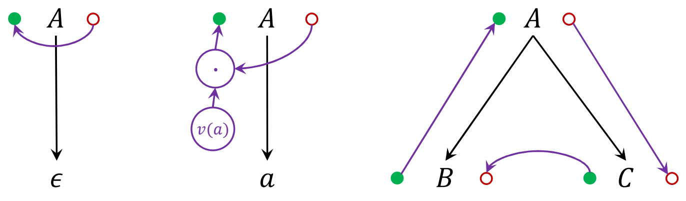
By Condition (4) of Definition 1, there exists an algorithm that, given any element $m \in M$, computes a decomposition $m = f \triangleleft c$. Let $\mathcal{A} : M \longrightarrow F \times C$ be the function computing that decomposition.

Consider the following bottom-up strategy for computing $v^*(w)$, for all $w \in \mathcal{L}(G)$:



To characterize the values computed for this language via Parikh's Theorem, we apply a series of equivalence-preserving transformations to the underlying grammar similar to the ones presented in §2.1.

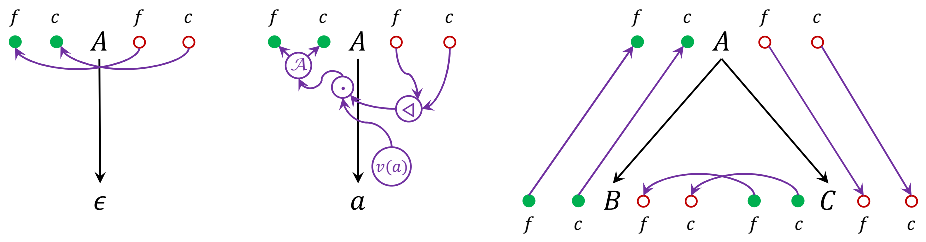
First, we change the order of evaluation to use to right-to-left threading:



(2)

(The initial attribute of the ROOT nonterminal would be assigned 1.) This transformation maintains equivalence by associativity of the monoid operation "·".

We now take advantage of the properties of the almost-commuting monoid M. First, because all inherited and synthesized attribute values are elements of M, by Condition (4) of Definition 1, we know that each attribute value $m \in M$ can be decomposed into $(f, c) = \mathcal{A}(m)$, where (i) $f \in F \subseteq M$, (ii) F is a finite subset of M, (iii) $c \in$ commutative monoid C, and (iv) $m = f \triangleleft c$.



(The f and c initial attributes of the ROOT nonterminal would be assigned 1 and 0, respectively.)

We now perform four rewrites, shown below, of the attribute-definition functions in the production $A \longrightarrow a$. (Rewritten elements are shown as dashed blue lines.)

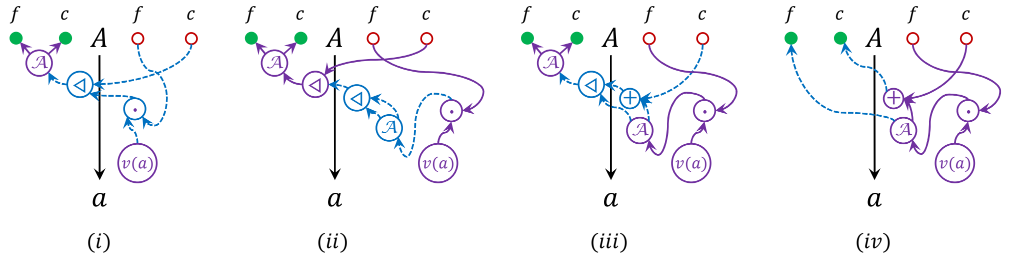
- To obtain (i), we use Condition (2) of Definition 1: $v(a) \cdot (f \triangleleft c) = (v(a) \cdot f) \triangleleft c$.
- To obtain (ii), we use the identity

$$\text{for all } m, m = \text{let } \langle f_1, c_1 \rangle = \mathcal{A}(m) \text{ in } f_1 \triangleleft c_1.$$

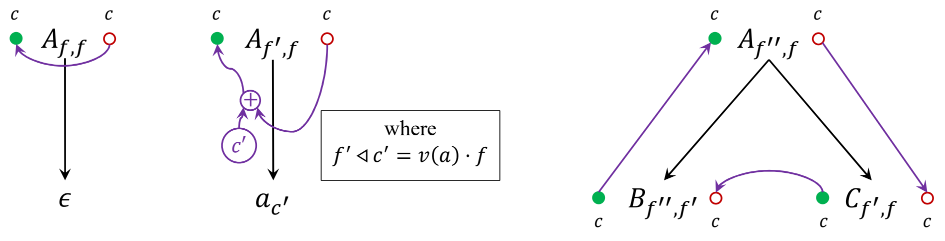
- To obtain (iii), we use Condition (3) of Definition 1 in the form $(f_1 \triangleleft c_1) \triangleleft c = f_1 \triangleleft (c_1 + c)$.
- To obtain (iv), we observe that because

$$\text{for all } f_1 \text{ and } c_1, (f_1 \triangleleft c_1) = \text{ let } \langle f, c \rangle = \mathcal{A}(f_1 \triangleleft c_1) \text{ in } (f \triangleleft c)$$

we can pass through $\langle f_1, c_1 \rangle$ in place of $\mathcal{A}(f_1 \triangleleft c_1)$.



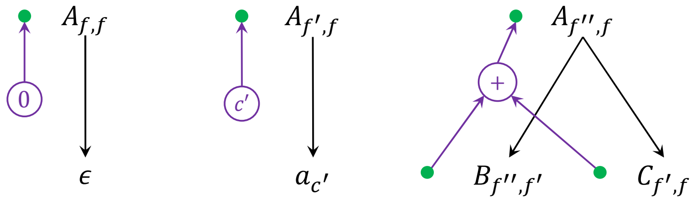
As in §2.1, the next grammar transformation involves labeling the nonterminals with values of inherited and synthesized f attributes. Because F is finite, this transformation incurs only a constant-factor blow-up in the number of productions.



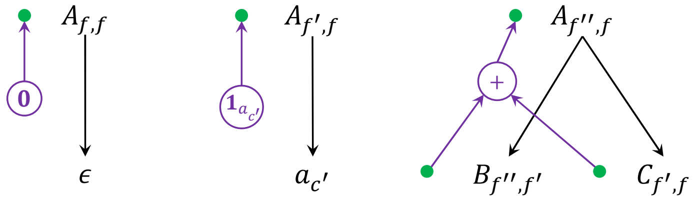
(3)

Unlike in §2.1, this step also expands the alphabet Σ, but again the expansion is only by at most a constant factor. In particular, every production of the form $A \longrightarrow a$ becomes $A_{f,f'} \longrightarrow a_{c'}$, where (i) $f \in F$ is a possible value of the inherited attribute f of the original left-hand-side nonterminal A, (ii) $f' \triangleleft c' = v(a) \cdot f$, and (iii) $a_{c'}$ is a new terminal symbol. Note that $v(a) \in M$ is a fixed known value associated with alphabet symbol a. Condition (4) of Definition 1 ensures that $v(a) \cdot f$ can be decomposed into $f' \triangleleft c' = v(a) \cdot f$, and thus, for a given terminal symbol a and value $f \in F$, $c' \in C$ is also a fixed known value.

The attribute grammar above accumulates a value in C via right-to-left threading. Because the operator $+$ of the commutative monoid C is associative, we can rewrite the attribute grammar to use a bottom-up accumulation pattern:



Finally, we transform the grammar to the form for computing the Parikh image of a word:



Because $+ : C \times C \longrightarrow C$ is commutative, we can obtain a value in C of the Parikh image of a given word w. For instance, if $a_{c'}$ has a count of 3 in the Parikh image of w, then the contribution to the value of w in C is $c' + c' + c'$. The value of w is the sum of the contributions of each terminal symbol in w.

We can relate the values of Parikh images obtained in this way to the values of the words in the original grammar as follows. Consider the Parikh images of languages of the form $\mathcal{L}(\mathrm{ROOT}_{f,1})$. For a given word $w \in \mathcal{L}(\mathrm{ROOT}_{f,1})$, let $c_w \in C$ be the value of w computed from the Parikh image of w as discussed above. Let $\widehat{w}$ be the word in the original alphabet Σ obtained by dropping all c labels from the terminal-symbols in w. Then the $v^*(\widehat{w}) = f \triangleleft c_w$.

6.2 Formal Proofs

This subsection presents formal proofs of Theorems 3 and 4.

Theorem 3. Let $\langle M, \cdot, 1\rangle$ be an almost-commuting monoid, let Σ be a finite alphabet, and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Let $v : \Sigma \longrightarrow M$ be a valuation function and let $v^* : \Sigma^* \longrightarrow M$ be that valuation function extended to a monoid homomorphism. Then, the set

$$\{v^*(w) \mid w \in \mathcal{L}(G)\}$$

is semi-linear and can be weakly defined in time $O(|G||F|^3)$ where $F \subseteq M$ is the finite subset of M.

Proof. Let finite subset $F \subseteq M$, commutative monoid $\langle C, +, 0\rangle$, and action $\triangleleft : M \times C \longrightarrow M$ be the objects witnessing that $\langle M, \cdot, 1\rangle$ is almost-commuting.

By Condition (4) of Definition 1, there exists an algorithm that, given any element $m \in M$, computes a decomposition $m = f \triangleleft c$. Let $\mathcal{A} : M \longrightarrow F \times C$ be the function computing that decomposition.

Our strategy is to construct a grammar G' such that $\{v^*(w) \mid w \in \mathcal{L}(G)\}$ can be calculated from the Parikh image of $\mathcal{L}(G')$. This strategy is enabled by the observation that the homomorphism $v^* : \Sigma^* \longrightarrow M$ factors as $v^*(w) = \sigma(\phi(w, 1))$, where $\sigma : F \times C^* \longrightarrow M$ is defined by

$$\sigma(f, c_1 \ldots c_n) \triangleq f \triangleleft (c_1 + \cdots + c_n),$$

and for $\phi : \Sigma^* \times F \to F \times C^*$, $\phi(w, f)$ calculates a representation of $v^*(w) \cdot f$ by a "right fold" (analogous to the "right-to-left" evaluation strategy depicted in the set of diagrams labeled (2) in §6.1). Formally, ϕ is defined by

$$\phi(\epsilon, f) \triangleq \langle f, \epsilon \rangle$$
$$\phi(wa, f) \triangleq \langle f'', tc \rangle \text{ where } \langle f'', t \rangle = \phi(w, f') \text{ and } \langle f', c \rangle = \mathcal{A}(v(a) \cdot f)$$

We show that for all $f \in F$, and all $w \in \Sigma^*$, we have $v^*(w) \cdot f = \sigma(\phi(w, f))$ by induction on w. For the base case we have

$$v^*(\epsilon) \cdot f = 1 \cdot f = f = \sigma(f, \epsilon) = \sigma(\phi(\epsilon, f)) .$$

For the induction step, we have

$$\begin{aligned}
v^*(wa) \cdot f &= v^*(w) \cdot v(a) \cdot f && \text{Homomorphism}\\
&= v^*(w) \cdot (f' \triangleleft c) && \text{where } \langle f', c \rangle = \mathcal{A}(v(a) \cdot f)\\
&= (v^*(w) \cdot f') \triangleleft c && \text{Definition 1(2) (Reassociativity)}\\
&= (\sigma(\phi(w, f'))) \triangleleft c && \text{Induction hypothesis}\\
&= (\sigma(f'', t_1 \dots t_n)) \triangleleft c && \text{where } \langle f'', t_1 \dots t_n \rangle = \phi(w, f')\\
&= (f'' \triangleleft (t_1 + \dots + t_n)) \triangleleft c && \text{Definition of } \sigma\\
&= f'' \triangleleft (t_1 + \dots + t_n + c) && \text{Definition 1(3) (Action)}\\
&= \sigma(f'', t_1 \dots t_n c) && \text{Definition of } \sigma\\
&= \sigma(\phi(wa, f)) && \text{Definition of } \phi
\end{aligned}$$

Our next task is to define a grammar G' that recognizes $\{\phi(w, 1) \mid w \in \mathcal{L}(G)\}$. Although formally $\phi(w, 1)$ belongs to $F \times C^*$, it may be thought of as a word over the alphabet $F \cup \hat{C}$, where

$$\hat{C} \triangleq \{c \in C \mid \exists s \in \Sigma, f \in F, \mathcal{A}(v(s) \cdot f) = \langle f', c \rangle\}$$

is a finite subset of C (the fact that $\sigma(w, f)$ belongs to $F \times \hat{C}^*$ can be seen by inspection of the definition of σ).

Given context-free grammar $G = \langle N, \Sigma, R, S \rangle$ in Chomsky Normal Form, we construct the grammar G' as follows:

$$\begin{aligned}
G' &= \left\langle N', F \cup \hat{C}, R', S^* \right\rangle\\
N' &= \left\{A_{f',f} : A \in N, f, f' \in F\right\} \cup \{S^*\}\\
R' &= \left\{A_{f'',f} \to B_{f'',f'}\ C_{f',f} : A \to B\ C \in R, f, f', f'' \in F\right\}\\
&\quad \cup \left\{A_{f',f} \to c : A \to a \in R, f, f' \in F, c \in \hat{C}, f' \triangleleft c = v(a) \cdot f\right\}\\
&\quad \cup \left\{A_{f,f} \to \epsilon : A \to \epsilon, f \in F\right\}\\
&\quad \cup \left\{S^* \to f\ S_{f,1} : f \in F\right\}
\end{aligned}$$

The definition of the new production rules R' mirrors the set of diagrams labeled (3) in §6.1. In the last line, the first f on the right-hand side of $S^* \to f\ S_{f,1}$ is a "terminal-symbol tag" indicating that the F value for each word derived from $S_{f,1}$ is f. (The diagrams in (3) do not depict the productions of the form $S^* \to f\ S_{f,1}$.)

Then we have the following property:

$$\{v^*(w) \mid w \in \mathcal{L}(G)\} = \left\{ f \triangleleft \left(\sum_{c \in \hat{C}} \phi(c) c \right) \middle| \phi \in Parikh(\mathcal{L}(G')), \phi(f) = 1 \right\} \tag{4}$$

Here, $\phi(f) = 1$ and $f \triangleleft (\ldots)$ serve to (i) "read" the terminal-symbol tag f, and (ii) incorporate f into the value computed for a word, respectively.

We may thereby use the formula $\psi_{G'}$ representing the Parikh image of G' [Verma et al. 2005] to define the following formula defining our subset of interest. The free variables of $\psi_{G'}$ correspond to the terminals of the grammar G', which we write as $Y = \{y_i \mid 1 \leq i \leq |F|\}$ and $Z = \left\{z_c \mid c \in \hat{C}\right\}$. Then, the formula that weakly defines (Definition 3) $\{v^*(w) \mid w \in \mathcal{L}(G)\}$ is:

$$\exists Y, Z. \left(\psi_{G'}(Y, Z) \land \left(\bigwedge_{i=1}^{|F|} y_i = 1 \Leftrightarrow x_0 = i \right) \land (x_1, \ldots, x_n) = \sum_{c \in \hat{C}} z_c \cdot c \right).$$

□

Theorem 4. Let T be a queryable ACTS defined over a finite alphabet Σ, and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Then there is a decision procedure to query the $\mathcal{L}(G)$-reachability relation of T for membership.

Proof. Given any states ρ, ρ', we can compute a formula that strongly defines the set $\{\rightarrow \mid \rho \rightarrow \rho'\}$ because T is queryable, conjoin this formula with that weakly defining $\left\{\rightarrow \mid \rightarrow = \rightarrow_{s_1} \cdot \ldots \cdot \rightarrow_{s_n}, s_1 \ldots s_n \in \mathcal{L}(G)\right\}$ via Theorem 3, and test for satisfiability. Because satisfiability of Presburger formulas is decidable, this procedure constitutes a decision procedure for checking membership of $\langle \rho, \rho' \rangle$ in the $\mathcal{L}(G)$-reachability relation of T. □

We have that similar theorems hold for dual ACM and dual ACTS.

Theorem 5. Let $\langle M, \cdot, 1 \rangle$ be a dual almost-commuting monoid, let Σ be a finite alphabet, and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Let $v : \Sigma \rightarrow M$ be a valuation function and let $v^* : \Sigma^* \rightarrow M$ be that valuation function extended to monoid homomorphism. Then, the set:

$$\{v^*(w) \mid w \in \mathcal{L}(G)\}$$

is semi-linear and definable in polynomial time.

Proof. This theorem is proven analogously to Theorem 3, except with the relevant subset of the commutative monoid $\hat{C}$ and the production rules R' of the blown up grammar G' defined as:

$$\begin{aligned}
\hat{C} &= \{c \in C \mid s \in \Sigma, f, f' \in F, f' \triangleleft c = f \cdot v(s)\} \\
R' &= \left\{A_{f,f''} \rightarrow B_{f,f'}\, C_{f',f''} \mid A \rightarrow B\, C \in R, f, f', f'' \in F\right\} \\
&\quad \cup \left\{A_{f,f'} \rightarrow c \mid A \rightarrow a \in R, f, f' \in F, c \in \hat{C}, f' \triangleleft c = f \cdot v(a)\right\} \\
&\quad \cup \left\{A_{f,f} \rightarrow \epsilon \mid A \rightarrow \epsilon, f \in F\right\} \\
&\quad \cup \left\{S^* \rightarrow f\, S_{1,f} \mid f \in F\right\}
\end{aligned}$$

Diagrammatically, this construction follows the same reasoning as that shown in §6.1 with "left-to-right" threading in place of "right-to-left" threading.

□

Theorem 6. Let T be an queryable dual ACTS defined over finite alphabet Σ and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Then, there is a decision procedure to query the $\mathcal{L}(G)$-reachability relation of T for membership.

Proof. Similar to the proof of Theorem 4. □

7 Closure Properties of ACTS

In this section, we demonstrate that almost-commuting transition systems (ACTS) enjoy robust closure properties. These properties ensure that complex systems constructed from smaller, analyzable components remain within the ACTS class, preserving tractability of CFL-reachability problems. Specifically, we (i) show that ACTS are closed under direct product and (ii) generalize the framework to encompass a recursive class of systems we call generalized ACTS. We also provide a concrete example of a transition system that lies in this broader class.

7.1 Direct Product

We begin with the result that the direct product of two almost-commuting monoids is itself an almost-commuting monoid.

Theorem 7. Let $\langle M, \cdot, 1\rangle$ and $\langle M', \cdot', 1'\rangle$ be almost-commuting monoids. Then, their direct product $\langle M \times M', \otimes, \langle 1, 1'\rangle\rangle$ is an almost-commuting monoid, in which:

$$\left\langle m_1, m_1'\right\rangle \otimes \left\langle m_2, m_2'\right\rangle = \left\langle m_1 \cdot m_2, m_1' \cdot' m_2'\right\rangle$$

Proof. By the definition of ACMs, there exist finite subsets $F \subseteq M$ and $F' \subseteq M'$, commutative monoids $\langle C, +, 0\rangle$ and $\langle C', +', 0'\rangle$, and actions $\triangleleft : M \times C \to M$ and $\triangleleft' : M' \times C' \to M'$ fulfilling the conditions of Definition 1 for $\langle M, \cdot, 1\rangle$ and $\langle M', \cdot', 1'\rangle$, respectively.

Then, finite subset $F \times F' \subseteq M \times M'$, commutative monoid $\langle C \times C', \oplus, \langle 0, 0'\rangle\rangle$ in which $\left\langle c_1, c_1'\right\rangle \oplus \left\langle c_2, c_2'\right\rangle = \left\langle c_1 + c_2, c_1' +' c_2'\right\rangle$, and action $\blacktriangleleft\colon (M \times M') \times (C \times C') \to (M \times M')$ defined as $\langle m, m'\rangle \blacktriangleleft \langle c, c'\rangle = \langle m \triangleleft c, m' \triangleleft' c'\rangle$ fulfill the conditions of Definition 1. Therefore, $\langle M \times M', \otimes, \langle 1, 1'\rangle\rangle$ is an almost-commuting monoid. ◻

This result implies that the ACTS framework is compositional: multiple ACTS components can be combined in parallel while remaining analyzable. The following example shows this property concretely.

Example 7.1. Let $T_1 = \left\langle \mathbb{Z}, \left\{\xrightarrow{1}_s : s \in \Sigma\right\}\right\rangle$ and $T_2 = \left\langle \mathbb{Z}, \left\{\xrightarrow{2}_s : s \in \Sigma\right\}\right\rangle$ be Natural Offset Max-Plus Linear Systems (from §5.2) over finite alphabet Σ. By Theorem 1, we have that both of these transition systems are ACTS.

Let $T_3 = \left\langle \mathbb{Z}^2, \left\{\xrightarrow{3}_s : s \in \Sigma\right\}\right\rangle$ be a transition system in which for all $\rho_1, \rho_2, \rho_1', \rho_2' \in \mathbb{Z}$ and all $s \in \Sigma$ we have:

$$\langle \rho_1, \rho_2\rangle \xrightarrow{3}_s \left\langle \rho_1', \rho_2'\right\rangle \Leftrightarrow \left(\rho_1 \xrightarrow{1}_s \rho_1' \text{ and } \rho_2 \xrightarrow{2}_s \rho_2'\right)$$

Theorem 7 implies that T_3 too is an ACTS, because its underlying monoid of transition relations is the direct product of that of T_1 and T_2. This theorem thus shows that any ACTS can be generalized to an arbitrary dimensional state and that multiple ACTS can be joined in parallel while retaining the same algebraic structure.

7.2 Generalized ACTS

Next, we show that our construction can be generalized to a hierarchy of algebraic structures beyond almost-commuting monoids. This generalization allows the "commutative component" of an ACM to itself be a (generalized) ACM.

Definition 11. A commutative integer monoid is a rank-0 generalized almost-commuting monoid. A monoid $\langle M, \cdot, 1\rangle$ is a rank-n generalized almost-commuting monoid if there exists a finite subset F of M, a rank-$(n-1)$ generalized almost commuting monoid $\langle C, +, 0\rangle$, and a function $\triangleleft : M \times C \to M$ that follow the conditions of Definition 1 or Definition 9.

Definition 12. A generalized almost-commuting transition system is defined analogously to an ACTS with "generalized ACM" in place of "ACM."

The context-free-language-reachability relation remains semi-linear and efficiently definable over generalized ACTS, just as it does with basic ACTS.

Theorem 8. Let $\langle M, \cdot, 1\rangle$ be a generalized almost-commuting monoid, let Σ be a finite alphabet, and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Let $v : \Sigma \to M$ be a valuation function and let $v^* : \Sigma^* \to M$ be that valuation function extended to a monoid homomorphism. Then, the set

$$\{v^*(w) \mid w \in \mathcal{L}(G)\}$$

is semi-linear and weakly definable in polynomial time.

Proof. We can define our construction recursively in terms of the construction in the proof of Theorem 3.

If the underlying monoid $\langle C, +, 0\rangle$ is commutative, then $\langle M, \cdot, 1\rangle$ is in fact an almost-commuting monoid and we can directly apply the construction of Theorem 3.

If the underlying monoid $\langle C, +, 0\rangle$ is a generalized almost-commuting monoid, then this theorem states that we can compute a formula Ψ that weakly defines the $\mathcal{L}(G)$ reachability of C. We then apply the construction of Theorem 3 replacing Equation (4) with:

$$\{v^*(w) \mid w \in \mathcal{L}(G)\} = \left\{ f \triangleleft \left(\sum_{c \in \hat{C}} \phi(c)c \right) \mid \phi \models \Psi, \phi(f) = 1 \right\}.$$

□

Theorem 9. Let T be a queryable generalized ACTS defined over finite alphabet Σ and let $\mathcal{L}(G) \subseteq \Sigma^*$ be a context-free language. Then, there is a decision procedure to query the $\mathcal{L}(G)$-reachability relation of T for membership.

Proof. Similar to Theorem 4 using Theorem 8. □

Example 7.2. We now present a concrete example of a system that is not a basic ACTS but is a generalized ACTS. This example combines features from both FMAVAS and parity-guarded systems.

Definition 13. A *parity-guarded rotating system* over a finite alphabet Σ is a labeled transition system $\langle \mathbb{Z}, \{\to_s \mid s \in \Sigma\}\rangle$ equipped with functions $a, c : \Sigma \to \{-1, 0, 1\}$ and $b, d : \Sigma \to \mathbb{Z}$ such that for each label $s \in \Sigma$, the transition relation $\to_s$ satisfies:

$$x \to_s x' \Leftrightarrow x' = \text{if } x \text{ is even then } a(s)x + b(s) \text{ else } c(s)x + d(s)$$

Theorem 10. A parity-guarded rotating system is a generalized ACTS.

Proof. Let $\langle \mathbb{Z}, \{\to_s \mid s \in \Sigma\}\rangle$ be a parity-guarded rotating system. Let M denote the closure of $\{\to_s \mid s \in \Sigma\}$ under relational composition $\cdot$, and let $\mathbb{I}$ be the identity relation on $\mathbb{Z}$. We will show that $\langle M, \cdot, \mathbb{I}\rangle$ is a generalized ACM.

Consider the monoid $\langle M' = \{-1, 0, 1\} \times 2\mathbb{Z}, \otimes, \langle 1, 0\rangle\rangle$ in which:

$$\langle a, b\rangle \otimes \langle a', b'\rangle = \langle a * a', a' * b + b'\rangle$$

Let $F' \subseteq M' = \{\langle -1, 0\rangle, \langle 0, 0\rangle, \langle 1, 0\rangle\}$, let $C = \langle 2\mathbb{Z}, +, 0\rangle$, and define action $\triangleleft' : M' \times 2\mathbb{Z} \to M'$ as $\langle a, b\rangle \triangleleft' c = \langle a, b + c\rangle$. These definitions fulfill the conditions of Definition 1, making $\langle M', \otimes, \langle 1, 0\rangle\rangle$ an almost-commuting monoid.

By Theorem 7, we have that $\langle M' \times M', \oplus, \langle\langle 1, 0\rangle, \langle 1, 0\rangle\rangle\rangle$ is a generalized almost-commuting monoid, where we define $\langle m_1, m_2\rangle \oplus \langle m_1', m_2'\rangle = \langle m_1 \otimes m_1', m_2 \otimes m_2'\rangle$.

Now, define finite subset $F \subseteq M$ and action $\triangleleft : M \times (M' \times M') \to M$ to be:

$$F = \left\{ \begin{array}{c} f_{00} = \{\langle x, x' \rangle \mid x' = \text{if } x \text{ is even then } x \text{ else } x\}, \\ f_{10} = \{\langle x, x' \rangle \mid x' = \text{if } x \text{ is even then } x+1 \text{ else } x\}, \\ f_{01} = \{\langle x, x' \rangle \mid x' = \text{if } x \text{ is even then } x \text{ else } x+1\}, \\ f_{11} = \{\langle x, x' \rangle \mid x' = \text{if } x \text{ is even then } x+1 \text{ else } x+1\} \end{array} \right\}$$

$$m \triangleleft \langle \langle a, b \rangle \, \langle c, d \rangle \rangle = \{x, x'' \mid \langle x, x' \rangle \in m, x'' = \text{if } x \text{ is even then } a * x' + b \text{ else } c * x' + d\}$$

Every generator $\to_s$ can be expressed as $f \triangleleft c$ for some $f \in F$ and $c \in M' \times M'$. Composition $f \triangleleft \langle \langle a, b \rangle, \langle c, d \rangle \rangle \cdot f' \triangleleft \langle \langle a', b' \rangle, \langle c', d' \rangle \rangle$ follows the same reasoning as that in the proof of Theorem 2. Therefore, $\langle M, \cdot, \mathbb{I} \rangle$ is a generalized almost-commuting monoid. □

8 Related Work

Reachability problems. Blondin et al. [2021] shows that the reachability problem for finite-monoid affine vector addition systems with states can be reduced to the reachability problem for integer vector addition systems with states using a similar method that we use to prove Theorem 3. Our paper extends this technique in two ways: (1) we "algebraize" the method, expressing the essence of the techniques as the laws of almost-commuting monoids, and (2) we extend the applicability from a model with states to one with a pushdown stack (i.e., *context-free* reachability rather than *regular* reachability).

Pimpalkhare and Kincaid [2024a] shows that the reachability relation of a labeled integer vector addition system with resets constrained to a context-free language can be encoded in Presburger arithmetic. This model is generalized by finite-monoid affine vector addition systems, and so Theorem 4 generalizes this result. While the technique presented in the present paper is more general, it is less efficient: the size of the formula produced by Theorem 4 is an exponential factor larger than that of Pimpalkhare and Kincaid [2024a]. Pimpalkhare and Kincaid [2024b] presents a similar result for a class of *semi-linear* integer vector addition systems with states, which is also an almost-commuting transition system. Again, Theorem 4 generalizes this result, at the cost of a double exponential factor.

Interprocedural program analysis. One approach to program analysis is to model the (abstract) meaning of a program as values drawn from a semiring, with the multiplication operation corresponding to sequential composition of actions along two program paths, and the addition operation to joining information across parallel program paths. In this setting, the problem of interest is to compute the sum of the weights of all interprocedurally-valid paths from one point to another—that is, to compute $\sum_{\pi \in \mathcal{L}(G)} w(\pi)$, where $\mathcal{L}(G)$ is a (context-free) language of paths of interest, and $w(\pi)$ is the weight of path π obtained by multiplying (in order) the semiring value on each edge of π. There is no general method for computing this value, but there are algorithms for some particular cases: (1) semirings in which there are no infinite ascending chains [Bouajjani et al. 2003; Reps et al. 2005], and (2) semirings in which the sequencing operation is commutative [Esparza et al. 2010]. This paper offers a new method for solving this problem for semirings of semi-linear sets (see [Bouajjani et al. 2003, §3.4.4]) over an almost-commuting monoid. Alternatively, one may view this paper as computing $\{w(\pi) : \pi \in \mathcal{L}(G)\}$—the *set* of all weights of all paths belonging to a context-free language $\mathcal{L}(G)$—rather than $\sum_{\pi \in \mathcal{L}(G)} w(\pi)$—the *sum* of such weights.

Extensions of Parikh's Theorem. Prior work has generalized Parikh's theorem to various weighted and algebraic settings, but always over commutative domains [Bhattiprolu et al. 2017; Ganty and Gutiérrez 2018]. Our work differs fundamentally by considering homomorphisms from context-free languages into *non-commutative* almost-commuting monoids.

9 Conclusion

This paper is motivated by solving context-free-language reachability problems for transition systems operating over infinite state spaces. Past work has successfully leveraged the celebrated Parikh's Theorem for this purpose, but under the restrictive assumption that *all* transitions must commute. We loosen this assumption, allowing transitions to be partially commutative and partially finite; that is, we show that Parikh's Theorem is applicable to *non-commutative* monoids that are almost-commuting.

This result opens a rich new frontier of *almost-commuting transition systems* for which we can now solve CFL-reachability problems. We identify several concrete examples of systems that lie on this frontier, and show that they can be combined compositionally while remaining within the class. We believe that our work holds promise as the backend of future program-analysis techniques that abstract programs as almost-commuting transition systems.

References

Vijay Bhattiprolu, Spencer Gordon, and Mahesh Viswanathan. 2017. Extending Parikh's Theorem to Weighted and Probabilistic Context-Free Grammars. In *Quantitative Evaluation of Systems*, Nathalie Bertrand and Luca Bortolussi (Eds.). Springer International Publishing, Cham, 3–19.

Michael Blondin, Christoph Haase, Filip Mazowiecki, and Mikhail Raskin. 2021. Affine Extensions of Integer Vector Addition Systems with States. *Logical Methods in Computer Science (LMCS)* 17, 3 (2021). doi:10.46298/lmcs-17(3:1)2021

Bernard Boigelot. 1998. *Symbolic Methods for Exploring Infinite State Spaces.* Ph. D. Dissertation. University of Liège, Belgium. https://orbi.uliege.be/handle/2268/74874

Ahmed Bouajjani, Javier Esparza, and Oded Maler. 1997. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1243)*, Antoni W. Mazurkiewicz and Józef Winkowski (Eds.). Springer, 135–150. doi:10.1007/3-540-63141-0_10

Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. 2003. A generic approach to the static analysis of concurrent programs with procedures. In *Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003*, Alex Aiken and Greg Morrisett (Eds.). ACM, 62–73. doi:10.1145/604131.604137

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

Bart De Schutter and Ton van den Boom. 2008. Max-plus algebra and max-plus linear discrete event systems: An introduction. In *2008 9th International Workshop on Discrete Event Systems.* 36–42. doi:10.1109/WODES.2008.4605919

Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. 2000. Efficient Algorithms for Model Checking Pushdown Systems. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1855)*, E. Allen Emerson and A. Prasad Sistla (Eds.). Springer, 232–247. doi:10.1007/10722167_20

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2010. Newtonian program analysis. *J. ACM* 57, 6 (2010), 33:1–33:47. doi:10.1145/1857914.1857917

Alain Finkel and Jérôme Leroux. 2002. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India, December 12-14, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2556)*, Manindra Agrawal and Anil Seth (Eds.). Springer, 145–156. doi:10.1007/3-540-36206-1_14

Alain Finkel, Bernard Willems, and Pierre Wolper. 1997. A direct symbolic approach to model checking pushdown systems. In *Second International Workshop on Verification of Infinite State Systems, Infinity 1997, Bologna, Italy, July 11-12, 1997 (Electronic Notes in Theoretical Computer Science, Vol. 9)*, Faron Moller (Ed.). Elsevier, 27–37. doi:10.1016/S1571-0661(05)80426-8

Pierre Ganty and Elena Gutiérrez. 2018. The Parikh Property for Weighted Context-Free Grammars. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 122)*, Sumit Ganguly and Paritosh Pandya (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 32:1–32:20. doi:10.4230/LIPIcs.FSTTCS.2018.32

Seymour Ginsburg and Edwin H. Spanier. 1966. Semigroups, Presburger Formulas, and Languages. *Pacific J. Math.* 16, 2 (1966), 285—-296.

Richard M. Karp and Raymond E. Miller. 1969. Parallel program schemata. *J. Comput. System Sci.* 3, 2 (1969), 147–195. doi:10.1016/S0022-0000(69)80011-5

Donald E. Knuth. 1968. Semantics of Context-Free Languages. *Math. Syst. Theory* 2, 2 (1968), 127–145. doi:10.1007/BF01692511

Yi Lu, Lei Shang, Xinwei Xie, and Jingling Xue. 2013. An incremental points-to analysis with CFL-Reachability. In *Proceedings of the 22nd International Conference on Compiler Construction* (Rome, Italy) *(CC'13)*. Springer-Verlag, Berlin, Heidelberg, 61–81. doi:10.1007/978-3-642-37051-9_4

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

Nikhil Pimpalkhare and Zachary Kincaid. 2024a. Monotone Procedure Summarization via Vector Addition Systems and Inductive Potentials. *Proc. ACM Program. Lang.* 8, OOPSLA2 (2024), 1873–1899. doi:10.1145/3689777

Nikhil Pimpalkhare and Zachary Kincaid. 2024b. Semi-linear VASR for Over-Approximate Semi-linear Transition System Reachability. In *Reachability Problems - 18th International Conference, RP 2024, Vienna, Austria, September 25-27, 2024, Proceedings (Lecture Notes in Computer Science, Vol. 15050)*, Laura Kovács and Ana Sokolova (Eds.). Springer, 154–166. doi:10.1007/978-3-031-72621-7_11

Polyvios Pratikakis, Jeffrey S. Foster, and Michael Hicks. 2006. Existential label flow inference via CFL reachability. In *Proceedings of the 13th International Conference on Static Analysis* (Seoul, Korea) *(SAS'06)*. Springer-Verlag, Berlin, Heidelberg, 88–106. doi:10.1007/11823230_7

Mojżesz Presburger. 1929. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves*.

Jakob Rehof and Manuel Fähndrich. 2001. Type-base flow analysis: From polymorphic subtyping to CFL-reachability. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (London, United Kingdom) *(POPL '01)*. Association for Computing Machinery, New York, NY, USA, 54–66. doi:10.1145/360204.360208

Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '95)*. Association for Computing Machinery, New York, NY, USA, 49–61. doi:10.1145/199448.199462

Thomas W. Reps. 1998. Program analysis via graph reachability. *Inf. Softw. Technol.* 40, 11-12 (1998), 701–726. doi:10.1016/S0950-5849(98)00093-7

Thomas W. Reps, Stefan Schwoon, Somesh Jha, and David Melski. 2005. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.* 58, 1-2 (2005), 206–263. doi:10.1016/J.SCICO.2005.02.009

Yu Su, Ding Ye, and Jingling Xue. 2014. Parallel Pointer Analysis with CFL-Reachability. In *2014 43rd International Conference on Parallel Processing*. 451–460. doi:10.1109/ICPP.2014.54

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.

Mihalis Yannakakis. 1990. Graph-Theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, Daniel J. Rosenkrantz and Yehoshua Sagiv (Eds.). ACM Press, 230–242. doi:10.1145/298514.298576

Received 2025-07-10; accepted 2025-11-06