A Calculus for Modular Loop Acceleration and Non-Termination Proofs

Florian Frohn[1] and Carsten Fuhs[2]

[1]Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany.
[1]LuFG Informatik 2, RWTH Aachen University, Aachen, Germany.
[2]Department of Computer Science and Information Systems, Birkbeck, University of London, London, UK.

Abstract

Loop acceleration can be used to prove safety, reachability, runtime bounds, and (non-)termination of programs. To this end, a variety of acceleration techniques has been proposed. However, so far all of them have been monolithic, i.e., a single loop could not be accelerated using a *combination of several different* acceleration techniques. In contrast, we present a calculus that allows for combining acceleration techniques in a modular way and we show how to integrate many existing acceleration techniques into our calculus. Moreover, we propose two novel acceleration techniques that can be incorporated into our calculus seamlessly. Some of these acceleration techniques apply only to non-terminating loops. Thus, combining them with our novel calculus results in a new, modular approach for proving non-termination. An empirical evaluation demonstrates the applicability of our approach, both for loop acceleration and for proving non-termination.

1 Introduction

In the last years, loop acceleration techniques have successfully been used to build static analyses for programs operating on integers [3, 10, 21, 26, 28, 43]. Essentially, such techniques extract a quantifier-free first-order formula ψ from a single-path loop $\mathcal{T}$, i.e., a loop without branching in its body, such that ψ under-approximates (or is equivalent to) $\mathcal{T}$. More specifically, each model of the resulting formula ψ corresponds to an execution of $\mathcal{T}$ (and vice versa). By integrating such techniques into a suitable program-analysis framework [4, 21, 26, 28, 37], whole programs can be transformed into first-order formulas which can then be analyzed by off-the-shelf solvers. Applications include proving safety [37] or reachability [37, 43], deducing bounds on the runtime complexity [26], and proving (non-)termination [10, 21].

However, existing acceleration techniques apply only if certain prerequisites are in place. So the power of static analyses built upon loop acceleration depends on the applicability of the underlying acceleration technique.

In this paper, we introduce a calculus which allows for combining several acceleration techniques modularly in order to accelerate a single loop. This not only allows for modular combinations of standalone techniques, but it also enables interactions between different acceleration techniques, allowing them to obtain better results together. Consequently, our calculus can handle classes of loops where all standalone techniques

fail. Moreover, we present two novel acceleration techniques and integrate them into our calculus.

One important application of loop acceleration is proving non-termination. As already observed in [21], certain properties of loops – in particular monotonicity of (parts of) the loop condition w.r.t. the loop body – are crucial for both loop acceleration and proving non-termination. In [21], this observation has been exploited to develop a technique for deducing invariants that are helpful to deal with non-terminating as well as terminating loops: For the former, they help to prove non-termination, and for the latter, they help to accelerate the loop.

In this paper, we take the next step by also unifying the actual techniques that are used for loop acceleration and for proving non-termination. To this end, we identify loop acceleration techniques that, if applied in isolation, give rise to non-termination proofs. Furthermore, we show that the combination of such non-termination techniques via our novel calculus for loop acceleration gives rise to non-termination proofs, too. In this way, we obtain a modular framework for combining several different non-termination techniques in order to prove non-termination of a single loop.

In the following, we introduce preliminaries in Section 2. Then, we discuss existing acceleration techniques in Section 3. In Section 4, we present our calculus to combine acceleration techniques and show how existing acceleration techniques can be integrated into our framework. Section 5 lifts existing acceleration techniques to *conditional* acceleration techniques, which provides additional power in the context of our framework by enabling interactions between different acceleration techniques. Next, we present two novel acceleration techniques and incorporate them into our calculus in Section 6. Then we adapt our calculus and certain acceleration techniques for proving non-termination in Section 7. After discussing related work in Section 8, we demonstrate the applicability of our approach via an empirical evaluation in Section 9 and conclude in Section 10.

A conference version of this paper was published in [22]. The present paper provides the following additional contributions:

- We present formal correctness proofs for all of our results, which were omitted in [22] for reasons of space.
- We present an improved version of the loop acceleration technique from [26, Thm. 3.8] and [27, Thm. 7] that yields simpler formulas.
- We prove an informal statement from [22] on using arbitrary existing acceleration techniques in our setting, resulting in the novel Lemma 1.
- The adaptation of our calculus and of certain acceleration techniques for proving non-termination (Section 7) is completely new.
- We extend the empirical evaluation from [22] with extensive experiments comparing the adaptation of our calculus for proving non-termination with other state-of-the-art tools for proving non-termination (Section 9.2).

2 Preliminaries

We use the notation $\vec{x}$, $\vec{y}$, $\vec{z}$, ... for vectors. Let $\mathscr{C}(\vec{z})$ be the set of *closed-form expressions* over the variables $\vec{z}$. So $\mathscr{C}(\vec{z})$ may, for example, be defined to be the smallest set containing all expressions built from $\vec{z}$, integer constants, and binary function symbols $\{+, -, \cdot, /, \exp\}$ for addition, subtraction, multiplication, division, and exponentiation. However, there is no widely accepted definition of "closed forms", and the results of the current paper are independent of the precise definition of $\mathscr{C}(\vec{z})$ (which may use other function symbols). Thus, we leave its definition open to avoid restricting our results unnecessarily. We consider loops of the form

$$\textbf{while } \varphi \textbf{ do } \vec{x} \leftarrow \vec{a} \qquad (\mathcal{T}_{loop})$$

where $\vec{x}$ is a vector of d pairwise different variables that range over the integers, the loop condition $\varphi \in FO_{QF}(\mathscr{C}(\vec{x}))$ (which we also call *guard*) is a finite quantifier-free first-order formula over the atoms $\{p > 0 \mid p \in \mathscr{C}(\vec{x})\}$, and $\vec{a} \in \mathscr{C}(\vec{x})^d$ such that the function[1] $\vec{x} \mapsto \vec{a}$ maps integers to integers. *Loop* denotes the set of all such loops.

We identify $\mathcal{T}_{loop}$ and the pair $\langle \varphi, \vec{a} \rangle$. Moreover, we identify $\vec{a}$ and the function $\vec{x} \mapsto \vec{a}$, where we sometimes write $\vec{a}(\vec{x})$ to make the variables $\vec{x}$ explicit. We use the same convention for other (vectors of) expressions. Similarly, we identify the formula $\varphi(\vec{x})$ (or just φ) with the predicate $\vec{x} \mapsto \varphi$. We use the standard integer-arithmetic semantics for the symbols occurring in formulas.

[1] i.e., the (anonymous) function that maps $\vec{x}$ to $\vec{a}$

Throughout this paper, let n be a designated variable ranging over $\mathbb{N} = \{0, 1, 2, \ldots\}$ and let:

$$\vec{a} := \begin{pmatrix} a_1 \\ \cdots \\ a_d \end{pmatrix} \quad \vec{x} := \begin{pmatrix} x_1 \\ \cdots \\ x_d \end{pmatrix} \quad \vec{x}' := \begin{pmatrix} x'_1 \\ \cdots \\ x'_d \end{pmatrix} \quad \vec{y} := \begin{pmatrix} \vec{x} \\ n \\ \vec{x}' \end{pmatrix}$$

Intuitively, the variable n represents the number of loop iterations and $\vec{x}'$ corresponds to the values of the program variables $\vec{x}$ after n iterations.

$\mathcal{T}_{loop}$ induces a relation $\longrightarrow_{\mathcal{T}_{loop}}$ on $\mathbb{Z}^d$:

$$\varphi(\vec{x}) \wedge \vec{x}' = \vec{a}(\vec{x}) \iff \vec{x} \longrightarrow_{\mathcal{T}_{loop}} \vec{x}'$$

3 Existing Acceleration Techniques

In the following (up to and including Section 6), our goal is to *accelerate* $\mathcal{T}_{loop}$, i.e., to find a formula $\psi \in FO_{QF}(\mathscr{C}(\vec{y}))$ such that

$$\psi \iff \vec{x} \longrightarrow^n_{\mathcal{T}_{loop}} \vec{x}' \qquad \text{for all } n > 0. \quad \text{(equiv)}$$

To see why we use $\mathscr{C}(\vec{y})$ instead of, e.g., polynomials, consider the loop

$$\textbf{while } x_1 > 0 \textbf{ do } \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leftarrow \begin{pmatrix} x_1 - 1 \\ 2 \cdot x_2 \end{pmatrix}. \qquad (\mathcal{T}_{exp})$$

Here, an acceleration technique synthesizes, e.g., the formula

$$\begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} x_1 - n \\ 2^n \cdot x_2 \end{pmatrix} \wedge x_1 - n + 1 > 0, \qquad (\psi_{exp})$$

where $\begin{pmatrix} x_1 - n \\ 2^n \cdot x_2 \end{pmatrix}$ is equivalent to the value of $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ after n iterations, and the inequation $x_1 - n + 1 > 0$ ensures that $\mathcal{T}_{exp}$ can be executed at least n times. Clearly, the growth of x_2 cannot be captured by a polynomial, i.e., even the behavior of quite simple loops is beyond the expressiveness of polynomial arithmetic.

In practice, one can restrict our approach to weaker classes of expressions to ease automation, but the presented results are independent of such considerations.

Some acceleration techniques cannot guarantee (equiv), but the resulting formula is an under-approximation of $\mathcal{T}_{loop}$, i.e., we have

$$\psi \implies \vec{x} \longrightarrow^n_{\mathcal{T}_{loop}} \vec{x}' \qquad \text{for all } n > 0. \quad \text{(approx)}$$

If (equiv) holds, then ψ is *equivalent* to $\mathcal{T}_{loop}$. Similarly, if (approx) holds, then ψ *approximates* $\mathcal{T}_{loop}$.[2]

Definition 1 (Acceleration Technique) An *acceleration technique* is a partial function

$$accel : Loop \rightharpoonup FO_{QF}(\mathscr{C}(\vec{y})).$$

It is *sound* if the formula $accel(\mathcal{T})$ approximates $\mathcal{T}$ for all $\mathcal{T} \in \mathrm{dom}(accel)$. It is *exact* if the formula $accel(\mathcal{T})$ is equivalent to $\mathcal{T}$ for all $\mathcal{T} \in \mathrm{dom}(accel)$.

We now recall several existing acceleration techniques. In Section 4 we will see how these techniques can be combined in a modular way. All of them first compute a *closed form* $\vec{c} \in \mathscr{C}(\vec{x}, n)^d$ for the values of the program variables after n iterations.

Definition 2 (Closed Form) We call $\vec{c} \in \mathscr{C}(\vec{x}, n)^d$ a *closed form* of $\mathcal{T}_{loop}$ if

$$\forall \vec{x} \in \mathbb{Z}^d, n \in \mathbb{N}.\ \vec{c} = \vec{a}^n(\vec{x}).$$

Here, $\vec{a}^n$ is the n-fold application of $\vec{a}$, i.e., $\vec{a}^0(\vec{x}) = \vec{x}$ and $\vec{a}^{n+1}(\vec{x}) = \vec{a}(\vec{a}^n(\vec{x}))$.

To find closed forms, one tries to solve the system of recurrence equations $\vec{x}^{(n)} = \vec{a}(\vec{x}^{(n-1)})$ with the initial condition $\vec{x}^{(0)} = \vec{x}$. In the sequel, we assume that we can represent $\vec{a}^n(\vec{x})$ in closed form. Note that one can always do so if $\vec{a}(\vec{x}) = A\vec{x} + \vec{b}$ with $A \in \mathbb{Z}^{d \times d}$ and $\vec{b} \in \mathbb{Z}^d$, i.e., if $\vec{a}$ is linear. To this end, one considers the matrix $B := \begin{pmatrix} A & \vec{b} \\ \vec{0}^T & 1 \end{pmatrix}$ and computes its Jordan normal form J where $B = T^{-1}JT$ and J is a block diagonal matrix (which has complex entries if B has complex eigenvalues). Then the closed form for J^n can be given directly (see, e.g., [48]), and $\vec{a}^n(\vec{x})$ is equal to the first d components of $T^{-1}J^nT\begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$. Moreover, one can compute a closed form if $\vec{a} = \begin{pmatrix} c_1 \cdot x_1 + p_1 \\ \cdots \\ c_d \cdot x_d + p_d \end{pmatrix}$ where $c_i \in \mathbb{N}$ and each p_i is a polynomial over $x_1, \ldots, x_{i-1}$ [25, 36].

[2] While there are also over-approximating acceleration techniques (see Section 8.1), in this paper we are interested only in under-approximations.

3.1 Acceleration via Decrease *or* Increase

The first acceleration technique discussed in this section exploits the following observation: If $\varphi(\vec{a}(\vec{x}))$ implies $\varphi(\vec{x})$ and if $\varphi(\vec{a}^{n-1}(\vec{x}))$ holds, then the loop condition φ of $\mathcal{T}_{loop}$ is satisfied throughout (at least) n loop iterations. So in other words, it requires that the indicator function (or characteristic function) $I_\varphi : \mathbb{Z}^d \to \{0,1\}$ of φ with $I_\varphi(\vec{x}) = 1 \iff \varphi(\vec{x})$ is monotonically decreasing w.r.t. $\vec{a}$, i.e., $I_\varphi(\vec{x}) \geq I_\varphi(\vec{a}(\vec{x}))$.

Theorem 1 (Acceleration via Monotonic Decrease [43]) *If*

$$\varphi(\vec{a}(\vec{x})) \implies \varphi(\vec{x}),$$

then the following acceleration technique is exact:

$$\mathcal{T}_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \varphi(\vec{a}^{n-1}(\vec{x}))$$

We will prove the more general Theorem 7 in Section 5.

So for example, Theorem 1 accelerates $\mathcal{T}_{exp}$ to ψ_{exp}. However, the requirement $\varphi(\vec{a}(\vec{x})) \implies \varphi(\vec{x})$ is often violated in practice. To see this, consider the loop

$$\textbf{while } x_1 > 0 \wedge x_2 > 0 \textbf{ do } \binom{x_1}{x_2} \leftarrow \binom{x_1 - 1}{x_2 + 1}. \quad (\mathcal{T}_{non\text{-}dec})$$

It cannot be accelerated with Theorem 1 as

$$x_1 - 1 > 0 \wedge x_2 + 1 > 0 \not\implies x_1 > 0 \wedge x_2 > 0.$$

A dual acceleration technique to Theorem 1 is obtained by "reversing" the implication in the prerequisites of the theorem, i.e., by requiring

$$\varphi(\vec{x}) \implies \varphi(\vec{a}(\vec{x})).$$

So the resulting dual acceleration technique applies iff φ is a loop invariant of $\mathcal{T}_{loop}$.[3] Then $\{\vec{x} \in \mathbb{Z}^d \mid \varphi(\vec{x})\}$ is a *recurrent set* [35] (see also Section 8.2) of $\mathcal{T}_{loop}$. In other words, this acceleration technique applies if I_φ is monotonically increasing w.r.t. $\vec{a}$.

Theorem 2 (Acceleration via Monotonic Increase) *If*

$$\varphi(\vec{x}) \implies \varphi(\vec{a}(\vec{x})),$$

then the following acceleration technique is exact:

$$\mathcal{T}_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \varphi(\vec{x})$$

We will prove the more general Theorem 8 in Section 5.

Example 1 As a minimal example, Theorem 2 accelerates

$$\textbf{while } x > 0 \textbf{ do } x \leftarrow x + 1 \quad (\mathcal{T}_{inc})$$

to $x' = x + n \wedge x > 0$. △

3.2 Acceleration via Decrease *and* Increase

Both acceleration techniques presented so far have been generalized in [21].

Theorem 3 (Acceleration via Monotonicity [21]) *If*

$$\begin{aligned} \varphi(\vec{x}) &\iff \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x}), \\ \varphi_1(\vec{x}) &\implies \varphi_1(\vec{a}(\vec{x})), \\ \varphi_1(\vec{x}) \wedge \varphi_2(\vec{a}(\vec{x})) &\implies \varphi_2(\vec{x}), \qquad \textit{and} \\ \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x}) &\implies \varphi_3(\vec{a}(\vec{x})), \end{aligned}$$

then the following acceleration technique is exact:

$$\mathcal{T}_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \varphi_1(\vec{x}) \wedge \varphi_2(\vec{a}^{n-1}(\vec{x})) \wedge \varphi_3(\vec{x})$$

Proof Immediate consequence of Theorem 5 and Remark 1, which will be proven in Sections 4 and 5. □

Here, φ_1 and φ_3 are again invariants of the loop. Thus, as in Theorem 2 it suffices to require that they hold before entering the loop. On the other hand, φ_2 needs to satisfy a similar condition as in Theorem 1, and thus it suffices to require that φ_2 holds before the last iteration. In such cases, i.e., if

$$\varphi_1(\vec{x}) \wedge \varphi_2(\vec{a}(\vec{x})) \implies \varphi_2(\vec{x})$$

is valid, we also say that φ_2 is a *converse invariant* (w.r.t. φ_1). It is easy to see that Theorem 3 is equivalent to Theorem 1 if $\varphi_1 \equiv \varphi_3 \equiv \top$ (where $\top$ denotes logical truth) and it is equivalent to Theorem 2 if $\varphi_2 \equiv \varphi_3 \equiv \top$.

Example 2 With Theorem 3, $\mathcal{T}_{non\text{-}dec}$ can be accelerated to

$$\binom{x_1'}{x_2'} = \binom{x_1 - n}{x_2 + n} \wedge x_2 > 0 \wedge x_1 - n + 1 > 0 \quad (\psi_{non\text{-}dec})$$

[3] We call a formula δ a *loop invariant* of a loop $\mathcal{T}_{loop}$ if $\varphi(\vec{x}) \wedge \delta(\vec{x}) \implies \delta(\vec{a}(\vec{x}))$ is valid.

by choosing $\varphi_1 := x_2 > 0$, $\varphi_2 := x_1 > 0$, and $\varphi_3 := \top$. △

Theorem 3 naturally raises the question: Why do we need *two* invariants? To see this, consider a restriction of Theorem 3 where $\varphi_3 := \top$. It would fail for a loop like

$$\textbf{while } x_1 > 0 \wedge x_2 > 0 \textbf{ do } \binom{x_1}{x_2} \leftarrow \binom{x_1+x_2}{x_2-1} \quad (\mathcal{T}_{2\text{-}invs})$$

which can easily be handled by Theorem 3 (by choosing $\varphi_1 := \top$, $\varphi_2 := x_2 > 0$, and $\varphi_3 := x_1 > 0$). The problem is that the converse invariant $x_2 > 0$ is needed to prove invariance of $x_1 > 0$. Similarly, a restriction of Theorem 3 where $\varphi_1 := \top$ would fail for the following variant of $\mathcal{T}_{2\text{-}invs}$:

$$\textbf{while } x_1 > 0 \wedge x_2 > 0 \textbf{ do } \binom{x_1}{x_2} \leftarrow \binom{x_1-x_2}{x_2+1}$$

Here, the problem is that the invariant $x_2 > 0$ is needed to prove converse invariance of $x_1 > 0$.

3.3 Acceleration via Metering Functions

Another approach for loop acceleration uses *metering functions*, a variation of classical *ranking functions* from termination and complexity analysis [27]. While ranking functions give rise to *upper* bounds on the runtime of loops, metering functions provide *lower* runtime bounds, i.e., the definition of a metering function $mf : \mathbb{Z}^d \to \mathbb{Q}$ ensures that for each $\vec{x} \in \mathbb{Z}^d$, the loop under consideration can be applied at least $\lceil mf(\vec{x}) \rceil$ times.

Definition 3 (Metering Function [27]) We call a function $mf : \mathbb{Z}^d \to \mathbb{Q}$ a *metering function* if the following holds:

$$\begin{aligned} \varphi(\vec{x}) &\implies mf(\vec{x}) - mf(\vec{a}(\vec{x})) \leq 1 \text{ and} \\ \neg\varphi(\vec{x}) &\implies mf(\vec{x}) \leq 0 \end{aligned} \qquad \text{(mf-bounded)}$$

We can use metering functions to accelerate loops as follows:

Theorem 4 (Acceleration via Metering Functions [26, 27]) *Let mf be a metering function for $\mathcal{T}_{loop}$. Then the following acceleration technique is sound:*

$$\mathcal{T}_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge n < mf(\vec{x}) + 1$$

We will prove the more general Theorem 9 in Section 5. In contrast to [26, Thm. 3.8] and [27, Thm. 7], the acceleration technique from Theorem 4 does not conjoin the loop condition φ to the result, which turned out to be superfluous. The reason is that $0 < n < mf(\vec{x}) + 1$ implies φ due to (mf-bounded).

Example 3 Using the metering function $(x_1, x_2) \mapsto x_1$, Theorem 4 accelerates $\mathcal{T}_{exp}$ to

$$\left(\binom{x_1'}{x_2'} = \binom{x_1 - n}{2^n \cdot x_2} \wedge n < x_1 + 1\right) \equiv \psi_{exp}.$$

△

However, synthesizing non-trivial (i.e., non-constant) metering functions is challenging. Moreover, unless the number of iterations of $\mathcal{T}_{loop}$ equals $\lceil mf(\vec{x}) \rceil$ for all $\vec{x} \in \mathbb{Z}^d$, *acceleration via metering functions* is not exact.

Linear metering functions can be synthesized via Farkas' Lemma and SMT solving [27]. However, many loops have only trivial linear metering functions. To see this, reconsider $\mathcal{T}_{non\text{-}dec}$. Here, $(x_1, x_2) \mapsto x_1$ is not a metering function as $\mathcal{T}_{non\text{-}dec}$ cannot be iterated at least x_1 times if $x_2 \leq 0$. Thus, [26] proposes a refinement of [27] based on metering functions of the form $\vec{x} \mapsto I_\xi(\vec{x}) \cdot f(\vec{x})$ where $\xi \in FO_{QF}(\mathscr{C}(\vec{x}))$ and f is linear. With this improvement, the metering function

$$(x_1, x_2) \mapsto I_{x_2 > 0}(x_2) \cdot x_1$$

can be used to accelerate $\mathcal{T}_{non\text{-}dec}$ to

$$\binom{x_1'}{x_2'} = \binom{x_1 - n}{x_2 + n} \wedge x_2 > 0 \wedge n < x_1 + 1.$$

4 A Calculus for Modular Loop Acceleration

All acceleration techniques presented so far are monolithic: Either they accelerate a loop successfully or they fail completely. In other words, we cannot *combine* several techniques to accelerate a single loop. To this end, we now present a calculus that repeatedly applies acceleration techniques to simplify an *acceleration problem* resulting from a loop $\mathcal{T}_{loop}$ until it is *solved* and hence gives rise to a suitable $\psi \in FO_{QF}(\mathscr{C}(\vec{y}))$ which approximates or is equivalent to $\mathcal{T}_{loop}$.

Definition 4 (Acceleration Problem) A tuple

$$[\![\psi \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}}$$

where $\psi \in FO_{QF}(\mathscr{C}(\vec{y}))$, $\breve{\varphi}, \widehat{\varphi} \in FO_{QF}(\mathscr{C}(\vec{x}))$, and $\vec{a} : \mathbb{Z}^d \to \mathbb{Z}^d$ is an *acceleration problem*. It is *consistent* if ψ approximates $\langle \breve{\varphi}, \vec{a} \rangle$, *exact* if ψ is equivalent to $\langle \breve{\varphi}, \vec{a} \rangle$, and *solved* if it is consistent and $\widehat{\varphi} \equiv \top$. The *canonical acceleration problem* of a loop $\mathcal{T}_{loop}$ is

$$[\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}}\,.$$

Example 4 The canonical acceleration problem of $\mathcal{T}_{non\text{-}dec}$ is

$$\left[\!\!\left[\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} x_1 - n \\ x_2 + n \end{pmatrix} \;\middle|\; \top \;\middle|\; x_1 > 0 \wedge x_2 > 0 \right]\!\!\right]_{\binom{x_1-1}{x_2+1}}.$$

△

The first component ψ of an acceleration problem $[\![\psi \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}}$ is the partial result that has been computed so far. The second component $\breve{\varphi}$ corresponds to the part of the loop condition that has already been processed successfully. As our calculus preserves consistency, ψ always approximates $\langle \breve{\varphi}, \vec{a} \rangle$. The third component is the part of the loop condition that remains to be processed, i.e., the loop $\langle \widehat{\varphi}, \vec{a} \rangle$ still needs to be accelerated. The goal of our calculus is to transform a canonical into a solved acceleration problem.

More specifically, whenever we have simplified a canonical acceleration problem

$$[\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}}$$

to

$$[\![\psi_1(\vec{y}) \mid \breve{\varphi}(\vec{x}) \mid \widehat{\varphi}(\vec{x})]\!]_{\vec{a}}\,,$$

then $\varphi \equiv \breve{\varphi} \wedge \widehat{\varphi}$ and

$$\psi_1 \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{x}'.$$

Then it suffices to find some $\psi_2 \in FO_{QF}(\mathscr{C}(\vec{y}))$ such that

$$\vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{x}' \wedge \psi_2 \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle \widehat{\varphi}, \vec{a} \rangle} \vec{x}'. \tag{1}$$

The reason is that we have $\longrightarrow_{\langle \breve{\varphi}, \vec{a} \rangle} \cap \longrightarrow_{\langle \widehat{\varphi}, \vec{a} \rangle} = \longrightarrow_{\langle \breve{\varphi} \wedge \widehat{\varphi}, \vec{a} \rangle} = \longrightarrow_{\langle \varphi, \vec{a} \rangle}$ and thus

$$\psi_1 \wedge \psi_2 \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle \varphi, \vec{a} \rangle} \vec{x}',$$

i.e., $\psi_1 \wedge \psi_2$ approximates $\mathcal{T}_{loop}$.

Note that the acceleration techniques presented so far would map $\langle \widehat{\varphi}, \vec{a} \rangle$ to some $\psi_2 \in FO_{QF}(\mathscr{C}(\vec{y}))$ such that

$$\psi_2 \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle \widehat{\varphi}, \vec{a} \rangle} \vec{x}', \tag{2}$$

which does not use the information that we have already accelerated $\langle \breve{\varphi}, \vec{a} \rangle$. In Section 5, we will adapt all acceleration techniques from Section 3 to search for some $\psi_2 \in FO_{QF}(\mathscr{C}(\vec{y}))$ that satisfies (1) instead of (2), i.e., we will turn them into *conditional acceleration techniques*.

Definition 5 (Conditional Acceleration) We call a partial function

$$accel : Loop \times FO_{QF}(\mathscr{C}(\vec{x})) \rightharpoonup FO_{QF}(\mathscr{C}(\vec{y}))$$

a *conditional acceleration technique*. It is *sound* if

$$\vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{x}' \wedge accel(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle \chi, \vec{a} \rangle} \vec{x}'$$

for all $(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \in \mathrm{dom}(accel)$, $\vec{x}, \vec{x}' \in \mathbb{Z}^d$, and $n > 0$. It is *exact* if additionally

$$\vec{x} \longrightarrow^n_{\langle \chi \wedge \breve{\varphi}, \vec{a} \rangle} \vec{x}' \quad \text{implies} \quad accel(\langle \chi, \vec{a} \rangle, \breve{\varphi})$$

for all $(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \in \mathrm{dom}(accel)$, $\vec{x}, \vec{x}' \in \mathbb{Z}^d$, and $n > 0$.

Note that every acceleration technique gives rise to a conditional acceleration technique in a straightforward way (by disregarding the second argument $\breve{\varphi}$ of *accel* in Definition 5). Soundness and exactness can be lifted directly to the conditional setting.

Lemma 1 (Acceleration as Conditional Acceleration) *Let $accel_0$ be an acceleration technique following Definition 1 such that $accel_0(\langle \chi, \vec{a} \rangle) \implies \vec{x}' = \vec{a}^n(\vec{x})$ is valid whenever $\langle \chi, \vec{a} \rangle \in \mathrm{dom}(accel_0)$. Then for the conditional acceleration technique accel given by $accel(\mathcal{T}, \breve{\varphi}) := accel_0(\mathcal{T})$, the following holds:*

1. *accel is sound if and only if $accel_0$ is sound*
2. *accel is exact if and only if $accel_0$ is exact*

Proof For the "if" direction of 1., we need to show that

$$\vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{x}' \wedge accel(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle \chi, \vec{a} \rangle} \vec{x}'$$

if $accel_0$ is a sound acceleration technique. Thus:

$$\begin{aligned} & \vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{x}' \wedge accel(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \\ \implies & accel(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \\ \iff & accel_0(\langle \chi, \vec{a} \rangle) \qquad \text{(by definition of } accel\text{)} \end{aligned}$$

$$\implies \vec{x} \longrightarrow^n_{\langle\chi,\vec{a}\rangle} \vec{x}' \qquad \text{(by soundness of } accel_0\text{)}$$

For the "only if" direction of 1., we need to show that

$$accel_0(\langle\varphi,\vec{a}\rangle) \quad \text{implies} \quad \vec{x} \longrightarrow^n_{\langle\varphi,\vec{a}\rangle} \vec{x}'$$

if *accel* is a sound conditional acceleration technique. Thus:

$$\begin{aligned}
&accel_0(\langle\varphi,\vec{a}\rangle)\\
\iff\ &accel_0(\langle\varphi,\vec{a}\rangle) \wedge \vec{x}' = \vec{a}^n(\vec{x})\\
&\qquad \text{(by the prerequisites of the lemma)}\\
\iff\ &\vec{x} \longrightarrow^n_{\langle\top,\vec{a}\rangle} \vec{x}' \wedge accel_0(\langle\varphi,\vec{a}\rangle)\\
\iff\ &\vec{x} \longrightarrow^n_{\langle\top,\vec{a}\rangle} \vec{x}' \wedge accel(\langle\varphi,\vec{a}\rangle,\top)\\
&\qquad \text{(by definition of } accel\text{)}\\
\implies\ &\vec{x} \longrightarrow^n_{\langle\varphi,\vec{a}\rangle} \vec{x}' \qquad \text{(by soundness of } accel\text{)}
\end{aligned}$$

For the "if" direction of 2., soundness of *accel* follows from 1. We still need to show that

$$\vec{x} \longrightarrow^n_{\langle\chi\wedge\breve{\varphi},\vec{a}\rangle} \vec{x}' \quad \text{implies} \quad accel(\langle\chi,\vec{a}\rangle,\breve{\varphi})$$

if $accel_0$ is an exact acceleration technique. Thus:

$$\begin{aligned}
&\vec{x} \longrightarrow^n_{\langle\chi\wedge\breve{\varphi},\vec{a}\rangle} \vec{x}'\\
\implies\ &\vec{x} \longrightarrow^n_{\langle\chi,\vec{a}\rangle} \vec{x}'\\
\iff\ &accel_0(\langle\chi,\vec{a}\rangle) \qquad \text{(by exactness of } accel_0\text{)}\\
\iff\ &accel(\langle\chi,\vec{a}\rangle,\breve{\varphi}) \qquad \text{(by definition of } accel\text{)}
\end{aligned}$$

For the "only if" direction of 2., soundness of $accel_0$ follows from 1. We still need to show that

$$\vec{x} \longrightarrow^n_{\langle\varphi,\vec{a}\rangle} \vec{x}' \quad \text{implies} \quad accel_0(\langle\varphi,\vec{a}\rangle)$$

if *accel* is an exact conditional acceleration technique. Thus:

$$\begin{aligned}
&\vec{x} \longrightarrow^n_{\langle\varphi,\vec{a}\rangle} \vec{x}'\\
\implies\ &accel(\langle\varphi,\vec{a}\rangle,\top) \qquad \text{(by exactness of } accel\text{)}\\
\iff\ &accel_0(\langle\varphi,\vec{a}\rangle) \qquad \text{(by definition of } accel\text{)}
\end{aligned}$$

□

We are now ready to present our *acceleration calculus*, which combines loop acceleration techniques in a modular way. In the following, w.l.o.g. we assume that formulas are in CNF, and we identify the formula $\bigwedge_{i=1}^k C_i$ with the set of clauses $\{C_i \mid 1 \le i \le k\}$.

Definition 6 (Acceleration Calculus) The relation $\rightsquigarrow$ on acceleration problems is defined by the rule

$$\frac{\emptyset \neq \chi \subseteq \widehat{\varphi} \quad accel(\langle\chi,\vec{a}\rangle,\breve{\varphi}) = \psi_2}{[\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}} \rightsquigarrow_{(e)} [\![\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi]\!]_{\vec{a}}}$$

where *accel* is a sound conditional acceleration technique. A $\rightsquigarrow$-step is *exact* (written $\rightsquigarrow_e$) if *accel* is exact.

So our calculus allows us to pick a subset χ (of clauses) from the yet unprocessed condition $\widehat{\varphi}$ and "move" it to $\breve{\varphi}$, which has already been processed successfully. To this end, $\langle\chi,\vec{a}\rangle$ needs to be accelerated by a conditional acceleration technique, i.e., when accelerating $\langle\chi,\vec{a}\rangle$ we may assume $\vec{x} \longrightarrow^n_{\langle\breve{\varphi},\vec{a}\rangle} \vec{x}'$.

With Lemma 1, our calculus allows for combining arbitrary existing acceleration techniques without adapting them. However, many acceleration techniques can easily be turned into more sophisticated conditional acceleration techniques (see Section 5), which increases the power of our approach.

Example 5 We continue Example 4, where we fix $\chi := x_1 > 0$ for the first acceleration step. Thus, we first need to accelerate the loop $\left\langle x_1 > 0, \binom{x_1-1}{x_2+1} \right\rangle$ to enable a first $\rightsquigarrow$-step, and we need to accelerate $\left\langle x_2 > 0, \binom{x_1-1}{x_2+1} \right\rangle$ afterwards. The resulting derivation is shown in Figure 1. Thus, we successfully constructed the formula $\psi_{non\text{-}dec}$, which is equivalent to $\mathcal{T}_{non\text{-}dec}$. Note that here neither of the two steps benefit from the component $\widehat{\varphi}$ of the acceleration problems. We will introduce more powerful conditional acceleration techniques that benefit from $\widehat{\varphi}$ in Section 5. △

The crucial property of our calculus is the following.

Lemma 2 *The relation $\rightsquigarrow$ preserves consistency, and the relation $\rightsquigarrow_e$ preserves exactness.*

Proof For the first part of the lemma, assume

$$[\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}} \rightsquigarrow [\![\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi]\!]_{\vec{a}}$$

where $[\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}}$ is consistent and

$$accel(\langle\chi,\vec{a}\rangle,\breve{\varphi}) = \psi_2.$$

We get

$$\begin{aligned}
&\psi_1 \wedge \psi_2\\
\implies\ &\vec{x} \longrightarrow^n_{\langle\breve{\varphi},\vec{a}\rangle} \vec{x}' \wedge \psi_2\\
\implies\ &\vec{x} \longrightarrow^n_{\langle\breve{\varphi},\vec{a}\rangle} \vec{x}' \wedge \vec{x} \longrightarrow^n_{\langle\chi,\vec{a}\rangle} \vec{x}'\\
\iff\ &\vec{x} \longrightarrow^n_{\langle\breve{\varphi}\wedge\chi,\vec{a}\rangle} \vec{x}'
\end{aligned}$$

$$
\begin{aligned}
&\left[\!\!\left[\psi^{init}_{non\text{-}dec} := \binom{x_1'}{x_2'} = \binom{x_1-n}{x_2+n} \;\middle|\; \top \;\middle|\; x_1 > 0 \wedge x_2 > 0\right]\!\!\right]_{\binom{x_1-1}{x_2+1}} \\
\rightsquigarrow_e &\left[\!\!\left[\psi^{init}_{non\text{-}dec} \wedge x_1 - n + 1 > 0 \;\middle|\; x_1 > 0 \;\middle|\; x_2 > 0\right]\!\!\right]_{\binom{x_1-1}{x_2+1}} && \text{(Theorem 1)} \\
\rightsquigarrow_e &\left[\!\!\left[\psi^{init}_{non\text{-}dec} \wedge x_1 - n + 1 > 0 \wedge x_2 > 0 \;\middle|\; x_1 > 0 \wedge x_2 > 0 \;\middle|\; \top\right]\!\!\right]_{\binom{x_1-1}{x_2+1}} && \text{(Theorem 2)} \\
= &\left[\!\!\left[\psi_{non\text{-}dec} \mid x_1 > 0 \wedge x_2 > 0 \mid \top\right]\!\!\right]_{\binom{x_1-1}{x_2+1}}
\end{aligned}
$$

Fig. 1: $\rightsquigarrow$-derivation for $\mathcal{T}_{non\text{-}dec}$

The first step holds since $[\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}}$ is consistent and the second step holds since *accel* is sound. This proves consistency of

$$
\begin{aligned}
&[\![\psi_1 \wedge \psi_2 \mid \breve{\varphi} \wedge \chi \mid \widehat{\varphi} \setminus \chi]\!]_{\vec{a}} \\
=&[\![\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi]\!]_{\vec{a}}\,.
\end{aligned}
$$

For the second part of the lemma, assume

$$[\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}} \rightsquigarrow_e [\![\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi]\!]_{\vec{a}}$$

where $[\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}}$ is exact and $accel(\langle \chi, \vec{a}\rangle, \breve{\varphi}) = \psi_2$. We get

$$
\begin{aligned}
&\vec{x} \longrightarrow^n_{\langle \breve{\varphi} \wedge \chi, \vec{a}\rangle} \vec{x}' \\
\iff &\vec{x} \longrightarrow^n_{\langle \breve{\varphi} \wedge \chi, \vec{a}\rangle} \vec{x}' \wedge \psi_2 \quad \text{(by exactness of } accel\text{)} \\
\iff &\vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{x}' \wedge \psi_2 \\
\iff &\psi_1 \wedge \psi_2 \qquad \text{(by exactness of } [\![\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}]\!]_{\vec{a}}\text{)}
\end{aligned}
$$

which proves exactness of

$$[\![\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi]\!]_{\vec{a}}\,.$$

□

Then the correctness of our calculus follows immediately. The reason is that

$$[\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}} \rightsquigarrow^*_{(e)} [\![\psi(\vec{y}) \mid \breve{\varphi}(\vec{x}) \mid \top]\!]_{\vec{a}}$$

implies $\varphi \equiv \breve{\varphi}$.

Theorem 5 (Correctness of $\rightsquigarrow$) *If* $[\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}} \rightsquigarrow^* [\![\psi(\vec{y}) \mid \breve{\varphi}(\vec{x}) \mid \top]\!]_{\vec{a}}$, *then* ψ *approximates* $\mathcal{T}_{loop}$. *If* $[\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}} \rightsquigarrow^*_e [\![\psi(\vec{y}) \mid \breve{\varphi}(\vec{x}) \mid \top]\!]_{\vec{a}}$, *then* ψ *is equivalent to* $\mathcal{T}_{loop}$.

Termination of our calculus is trivial, as the size of the third component $\widehat{\varphi}$ of the acceleration problem is decreasing.

Theorem 6 (Well-Foundedness of $\rightsquigarrow$) *The relation* $\rightsquigarrow$ *is well-founded.*

5 Conditional Acceleration Techniques

We now show how to turn the acceleration techniques from Section 3 into conditional acceleration techniques, starting with *acceleration via monotonic decrease*.

Theorem 7 (Conditional Acceleration via Monotonic Decrease) *If*

$$\breve{\varphi}(\vec{x}) \wedge \chi(\vec{a}(\vec{x})) \implies \chi(\vec{x}), \tag{3}$$

then the following conditional acceleration technique is exact:

$$(\langle \chi, \vec{a}\rangle, \breve{\varphi}) \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \chi(\vec{a}^{n-1}(\vec{x}))$$

Proof For soundness, we need to prove

$$
\begin{aligned}
&\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \chi(\vec{a}^{m-1}(\vec{x})) \\
&\qquad\qquad \implies \vec{x} \longrightarrow^m_{\langle \chi, \vec{a}\rangle} \vec{a}^m(\vec{x}) \qquad (4)
\end{aligned}
$$

for all $m > 0$. We use induction on m. If $m = 1$, then

$$
\begin{aligned}
&\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \chi(\vec{a}^{m-1}(\vec{x})) \\
\implies &\chi(\vec{x}) && \text{(as } m = 1\text{)} \\
\iff &\vec{x} \longrightarrow_{\langle \chi, \vec{a}\rangle} \vec{a}(\vec{x}) \\
\iff &\vec{x} \longrightarrow^m_{\langle \chi, \vec{a}\rangle} \vec{a}^m(\vec{x}). && \text{(as } m = 1\text{)}
\end{aligned}
$$

In the induction step, we have

$$
\begin{aligned}
&\vec{x} \longrightarrow^{m+1}_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{a}^{m+1}(\vec{x}) \wedge \chi(\vec{a}^m(\vec{x})) \\
\implies &\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \chi(\vec{a}^m(\vec{x})) \\
\iff &\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \breve{\varphi}(\vec{a}^{m-1}(\vec{x})) \wedge \chi(\vec{a}^m(\vec{x})) \\
&\qquad\qquad \text{(as } m > 0\text{)} \\
\implies &\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \chi(\vec{a}^{m-1}(\vec{x})) \wedge \chi(\vec{a}^m(\vec{x})) \\
&\qquad\qquad \text{(due to (3))} \\
\implies &\vec{x} \longrightarrow^m_{\langle \chi, \vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \chi(\vec{a}^m(\vec{x})) \\
&\qquad\qquad \text{(by the induction hypothesis (4))}
\end{aligned}
$$

$$\iff \vec{x} \longrightarrow^{m+1}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m+1}(\vec{x}).$$

For exactness, we need to prove

$$\vec{x} \longrightarrow^{m}_{\langle \chi \wedge \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \implies \chi(\vec{a}^{m-1}(\vec{x}))$$

for all $m > 0$, which is trivial. □

So we just add $\breve{\varphi}$ to the premise of the implication that needs to be checked to apply *acceleration via monotonic decrease*. Theorem 2 can be adapted analogously.

Theorem 8 (Conditional Acceleration via Monotonic Increase) *If*

$$\breve{\varphi}(\vec{x}) \wedge \chi(\vec{x}) \implies \chi(\vec{a}(\vec{x})), \tag{5}$$

then the following conditional acceleration technique is exact:

$$(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \mapsto \vec{x}' = \vec{a}^{n}(\vec{x}) \wedge \chi(\vec{x})$$

Proof For soundness, we need to prove

$$\vec{x} \longrightarrow^{m}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \wedge \chi(\vec{x}) \implies \vec{x} \longrightarrow^{m}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \tag{6}$$

for all $m > 0$. We use induction on m. If $m = 1$, then

$$\begin{aligned} & \vec{x} \longrightarrow^{m}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \wedge \chi(\vec{x}) \\ \implies & \vec{x} \longrightarrow_{\langle \chi, \vec{a} \rangle} \vec{a}(\vec{x}) \\ \iff & \vec{x} \longrightarrow^{m}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m}(\vec{x}). \qquad (\text{as } m = 1) \end{aligned}$$

In the induction step, we have

$$\begin{aligned} & \vec{x} \longrightarrow^{m+1}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m+1}(\vec{x}) \wedge \chi(\vec{x}) \\ \implies & \vec{x} \longrightarrow^{m}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \wedge \chi(\vec{x}) \\ \implies & \vec{x} \longrightarrow^{m}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \wedge \vec{x} \longrightarrow^{m}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \\ & \qquad (\text{by the induction hypothesis (6)}) \\ \implies & \vec{x} \longrightarrow^{m}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \wedge \breve{\varphi}(\vec{a}^{m-1}(\vec{x})) \wedge \chi(\vec{a}^{m-1}(\vec{x})) \\ & \qquad (\text{as } m > 0) \\ \implies & \vec{x} \longrightarrow^{m}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \wedge \chi(\vec{a}^{m}(\vec{x})) \qquad (\text{due to (5)}) \\ \iff & \vec{x} \longrightarrow^{m+1}_{\langle \chi, \vec{a} \rangle} \vec{a}^{m+1}(\vec{x}). \end{aligned}$$

For exactness, we need to prove

$$\vec{x} \longrightarrow^{m}_{\langle \chi \wedge \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m}(\vec{x}) \implies \chi(\vec{x}),$$

for all $m > 0$, which is trivial. □

Example 6 For the canonical acceleration problem of $\mathcal{T}_{2\text{-}invs}$, we obtain the derivation shown in Figure 2, where $\vec{a}_{2\text{-}invs} := \begin{pmatrix} x_1 + x_2 \\ x_2 - 1 \end{pmatrix}$. While we could also use Theorem 1 for the first step, Theorem 2 is inapplicable in the second step. The reason is that we need the converse invariant $x_2 > 0$ to prove invariance of $x_1 > 0$. △

It is not a coincidence that $\mathcal{T}_{2\text{-}invs}$, which could also be accelerated with *acceleration via monotonicity* (see Theorem 3) directly, can be handled by applying our novel calculus with Theorems 7 and 8.

Remark 1 If applying *acceleration via monotonicity* to $\mathcal{T}_{loop}$ yields ψ, then

$$[\![\vec{x}' = \vec{a}^{n}(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}} \rightsquigarrow_{e}^{\leq 3} [\![\psi(\vec{y}) \mid \varphi(\vec{x}) \mid \top]\!]_{\vec{a}}$$

where either Theorem 7 or Theorem 8 is applied in each $\rightsquigarrow_e$-step.

Proof As Theorem 3 applies, we have

$$\varphi(\vec{x}) \equiv \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x})$$

where

$$\begin{aligned} \varphi_1(\vec{x}) &\implies \varphi_1(\vec{a}(\vec{x})) \quad \wedge & (7) \\ \varphi_1(\vec{x}) \wedge \varphi_2(\vec{a}(\vec{x})) &\implies \varphi_2(\vec{x}) \quad \wedge & (8) \\ \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x}) &\implies \varphi_3(\vec{a}(\vec{x})). & (9) \end{aligned}$$

If $\varphi_1 \neq \top$, then Theorem 8 applies to $\langle \varphi_1, \vec{a} \rangle$ with $\breve{\varphi} := \top$ due to (7), and we obtain:

$$\begin{aligned} & [\![\vec{x}' = \vec{a}^{n}(\vec{x}) \mid \top \mid \varphi(\vec{x})]\!]_{\vec{a}} \\ = & [\![\vec{x}' = \vec{a}^{n}(\vec{x}) \mid \top \mid \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x})]\!]_{\vec{a}} \\ \rightsquigarrow_e & [\![\vec{x}' = \vec{a}^{n}(\vec{x}) \wedge \varphi_1(\vec{x}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x})]\!]_{\vec{a}} \\ = & [\![\psi_1(\vec{y}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x})]\!]_{\vec{a}} \end{aligned}$$

Next, if $\varphi_2 \neq \top$, then Theorem 7 applies to $\langle \varphi_2, \vec{a} \rangle$ with $\breve{\varphi} := \varphi_1$ due to (8) and we obtain:

$$\begin{aligned} & [\![\psi_1(\vec{y}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \wedge \varphi_3(\vec{x})]\!]_{\vec{a}} \\ \rightsquigarrow_e & \left[\!\!\left[\psi_1(\vec{y}) \wedge \varphi_2(\vec{a}^{n-1}(\vec{x})) \;\middle|\; \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \;\middle|\; \varphi_3(\vec{x}) \right]\!\!\right]_{\vec{a}} \\ = & [\![\psi_2(\vec{y}) \mid \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \mid \varphi_3(\vec{x})]\!]_{\vec{a}} \end{aligned}$$

Finally, if $\varphi_3 \neq \top$, then Theorem 8 applies to $\langle \varphi_3, \vec{a} \rangle$ with $\breve{\varphi} := \varphi_1 \wedge \varphi_2$ due to (9) and we obtain

$$\begin{aligned} & [\![\psi_2(\vec{y}) \mid \varphi_1(\vec{x}) \wedge \varphi_2(\vec{x}) \mid \varphi_3(\vec{x})]\!]_{\vec{a}} \\ \rightsquigarrow_e & [\![\psi_2(\vec{y}) \wedge \varphi_3(\vec{x}) \mid \varphi(\vec{x}) \mid \top]\!]_{\vec{a}} \\ = & [\![\psi(\vec{y}) \mid \varphi(\vec{x}) \mid \top]\!]_{\vec{a}}. \end{aligned}$$

□

Thus, there is no need for a conditional variant of *acceleration via monotonicity*. Note that combining Theorems 7 and 8 with our calculus is also useful for loops where *acceleration via monotonicity* is inapplicable.

$$
\begin{aligned}
&[\![\vec{x}' = \vec{a}^n_{2\text{-}invs}(\vec{x}) \mid \top \mid x_1 > 0 \wedge x_2 > 0]\!]_{\vec{a}_{2\text{-}invs}} \\
\rightsquigarrow_e\ &[\![\vec{x}' = \vec{a}^n_{2\text{-}invs}(\vec{x}) \wedge x_2 - n + 1 > 0 \mid x_2 > 0 \mid x_1 > 0]\!]_{\vec{a}_{2\text{-}invs}} && \text{(Theorem 7)} \\
\rightsquigarrow_e\ &[\![\vec{x}' = \vec{a}^n_{2\text{-}invs}(\vec{x}) \wedge x_2 - n + 1 > 0 \wedge x_1 > 0 \mid x_2 > 0 \wedge x_1 > 0 \mid \top]\!]_{\vec{a}_{2\text{-}invs}} && \text{(Theorem 8)}
\end{aligned}
$$

Fig. 2: $\rightsquigarrow$-derivation for $\mathcal{T}_{2\text{-}invs}$

$$
\begin{aligned}
&\left[\!\!\left[\psi^{init}_{2\text{-}c\text{-}invs} \;\middle|\; \top \;\middle|\; \varphi_{2\text{-}c\text{-}invs}\right]\!\!\right]_{\vec{a}_{2\text{-}c\text{-}invs}} \\
\rightsquigarrow_e\ &\left[\!\!\left[\psi^{init}_{2\text{-}c\text{-}invs} \wedge x_1^{(n-1)} > 0 \;\middle|\; x_1 > 0 \;\middle|\; x_2 > 0 \wedge x_3 > 0\right]\!\!\right]_{\vec{a}_{2\text{-}c\text{-}invs}} && \text{(Theorem 7)} \\
\rightsquigarrow_e\ &\left[\!\!\left[\psi^{init}_{2\text{-}c\text{-}invs} \wedge x_1^{(n-1)} > 0 \wedge x_2 > 0 \;\middle|\; x_1 > 0 \wedge x_2 > 0 \;\middle|\; x_3 > 0\right]\!\!\right]_{\vec{a}_{2\text{-}c\text{-}invs}} && \text{(Theorem 8)} \\
\rightsquigarrow_e\ &\left[\!\!\left[\psi^{init}_{2\text{-}c\text{-}invs} \wedge x_1^{(n-1)} > 0 \wedge x_2 > 0 \wedge x_3^{(n-1)} > 0 \;\middle|\; \varphi_{2\text{-}c\text{-}invs} \;\middle|\; \top\right]\!\!\right]_{\vec{a}_{2\text{-}c\text{-}invs}} && \text{(Theorem 7)}
\end{aligned}
$$

Fig. 3: $\rightsquigarrow$-derivation for $\mathcal{T}_{2\text{-}c\text{-}invs}$

Example 7 Consider the following loop, which can be accelerated by splitting its guard into one invariant and two converse invariants.

while $x_1 > 0 \wedge x_2 > 0 \wedge x_3 > 0$ **do**

$$
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leftarrow \begin{pmatrix} x_1 - 1 \\ x_2 + x_1 \\ x_3 - x_2 \end{pmatrix} \qquad (\mathcal{T}_{2\text{-}c\text{-}invs})
$$

Let

$$
\begin{aligned}
\varphi_{2\text{-}c\text{-}invs} &:= x_1 > 0 \wedge x_2 > 0 \wedge x_3 > 0, \\
\vec{a}_{2\text{-}c\text{-}invs} &:= \begin{pmatrix} x_1 - 1 \\ x_2 + x_1 \\ x_3 - x_2 \end{pmatrix}, \\
\psi^{init}_{2\text{-}c\text{-}invs} &:= \vec{x}' = \vec{a}^n_{2\text{-}c\text{-}invs}(\vec{x}),
\end{aligned}
$$

and let $x_i^{(m)}$ be the i^{th} component of $\vec{a}^m_{2\text{-}c\text{-}invs}(\vec{x})$. Starting with the canonical acceleration problem of $\mathcal{T}_{2\text{-}c\text{-}invs}$, we obtain the derivation shown in Figure 3. △

Finally, we present a variant of Theorem 4 for conditional acceleration. The idea is similar to the approach for deducing metering functions of the form $\vec{x} \mapsto I_{\breve{\varphi}}(\vec{x}) \cdot f(\vec{x})$ from [26] (see Section 3.3 for details). But in contrast to [26], in our setting the "conditional" part $\breve{\varphi}$ does not need to be an invariant of the loop.

Theorem 9 (Conditional Acceleration via Metering Functions) *Let* $mf : \mathbb{Z}^d \to \mathbb{Q}$. *If*

$$
\breve{\varphi}(\vec{x}) \wedge \chi(\vec{x}) \implies mf(\vec{x}) - mf(\vec{a}(\vec{x})) \leq 1 \text{ and } \tag{10}
$$

$$
\breve{\varphi}(\vec{x}) \wedge \neg\chi(\vec{x}) \implies mf(\vec{x}) \leq 0, \tag{11}
$$

then the following conditional acceleration technique is sound:

$$
(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge n < mf(\vec{x}) + 1
$$

Proof We need to prove

$$
\begin{aligned}
\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^m(\vec{x}) \wedge m < mf(\vec{x}) + 1 \\
\implies \vec{x} \longrightarrow^m_{\langle \chi, \vec{a} \rangle} \vec{a}^m(\vec{x})
\end{aligned} \tag{12}
$$

for all $m > 0$. Note that (11) is equivalent to

$$
mf(\vec{x}) > 0 \implies \neg\breve{\varphi}(\vec{x}) \vee \chi(\vec{x}). \tag{13}
$$

We use induction on m. If $m = 1$, then

$$
\begin{aligned}
&\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^m(\vec{x}) \wedge m < mf(\vec{x}) + 1 \\
\iff\ &\breve{\varphi}(\vec{x}) \wedge mf(\vec{x}) > 0 && \text{(as } m = 1\text{)} \\
\implies\ &\chi(\vec{x}) && \text{(due to (13))} \\
\iff\ &\vec{x} \longrightarrow_{\langle \chi, \vec{a} \rangle} \vec{a}(\vec{x}) \\
\iff\ &\vec{x} \longrightarrow^m_{\langle \chi, \vec{a} \rangle} \vec{a}^m(\vec{x}) && \text{(as } m = 1\text{)}
\end{aligned}
$$

In the induction step, assume

$$
\vec{x} \longrightarrow^{m+1}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m+1}(\vec{x}) \wedge m < mf(\vec{x}). \tag{14}
$$

Then we have:

$$
\begin{aligned}
&(14) \\
\implies\ &\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^m(\vec{x}) \wedge m < mf(\vec{x}) \\
\implies\ &\vec{x} \longrightarrow^m_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^m(\vec{x}) \wedge m < mf(\vec{x}) \wedge
\end{aligned}
$$

$$\vec{x} \longrightarrow^{m}_{\langle\chi,\vec{a}\rangle} \vec{a}^m(\vec{x})$$
(due to the induction hypothesis (12))
$$\iff m < mf(\vec{x}) \wedge \vec{x} \longrightarrow^{m}_{\langle\chi,\vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \forall i \in [0, m-1].\ \left(\breve{\varphi}(\vec{a}^i(\vec{x})) \wedge \chi(\vec{a}^i(\vec{x}))\right)$$
$$\implies m < mf(\vec{x}) \wedge \vec{x} \longrightarrow^{m}_{\langle\chi,\vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge mf(\vec{x}) - mf(\vec{a}^m(\vec{x})) \leq m \quad \text{(due to (10))}$$
$$\implies \vec{x} \longrightarrow^{m}_{\langle\chi,\vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge mf(\vec{a}^m(\vec{x})) > 0 \quad \text{(as } m < mf(\vec{x}))$$
$$\implies \vec{x} \longrightarrow^{m}_{\langle\chi,\vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge (\neg\breve{\varphi}(\vec{a}^m(\vec{x})) \vee \chi(\vec{a}^m(\vec{x}))) \quad \text{(due to (13))}$$
$$\iff \vec{x} \longrightarrow^{m}_{\langle\chi,\vec{a}\rangle} \vec{a}^m(\vec{x}) \wedge \chi(\vec{a}^m(\vec{x})) \quad \text{(as (14) implies } \breve{\varphi}(\vec{a}^m(\vec{x})))$$
$$\iff \vec{x} \longrightarrow^{m+1}_{\langle\chi,\vec{a}\rangle} \vec{a}^{m+1}(\vec{x})$$

□

6 Acceleration via Eventual Monotonicity

The combination of the calculus from Section 4 and the conditional acceleration techniques from Section 5 still fails to handle certain interesting classes of loops. Thus, to improve the applicability of our approach we now present two new acceleration techniques based on *eventual* monotonicity.

6.1 Acceleration via Eventual Decrease

All (combinations of) techniques presented so far fail for the following example.

$$\textbf{while } x_1 > 0 \textbf{ do } \binom{x_1}{x_2} \leftarrow \binom{x_1+x_2}{x_2-1} \qquad (\mathcal{T}_{ev\text{-}dec})$$

The reason is that x_1 does not behave monotonically, i.e., $x_1 > 0$ is neither an invariant nor a converse invariant. Essentially, $\mathcal{T}_{ev\text{-}dec}$ proceeds in two phases: In the first (optional) phase, x_2 is positive and hence the value of x_1 is monotonically increasing. In the second phase, x_2 is non-positive and consequently the value of x_1 decreases (weakly) monotonically. The crucial observation is that once the value of x_1 decreases, it can never increase again. Thus, despite the non-monotonic behavior of x_1, it suffices to require that $x_1 > 0$ holds before the first and before the n^{th} loop iteration to ensure that the loop can be iterated at least n times.

Theorem 10 (Acceleration via Eventual Decrease) *If $\varphi(\vec{x}) \equiv \bigwedge_{i=1}^{k} C_i$ where each clause C_i contains an inequation $e_i(\vec{x}) > 0$ such that*

$$e_i(\vec{x}) \geq e_i(\vec{a}(\vec{x})) \implies e_i(\vec{a}(\vec{x})) \geq e_i(\vec{a}^2(\vec{x})),$$

then the following acceleration technique is sound:

$$\mathcal{T}_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge e_i(\vec{a}^{n-1}(\vec{x})) > 0$$

If $C_i \equiv e_i > 0$ for all $i \in [1,k]$, then it is exact.

We will prove the more general Theorem 11 later in this section.

Example 8 With Theorem 10, we can accelerate $\mathcal{T}_{ev\text{-}dec}$ to

$$\binom{x_1'}{x_2'} = \binom{\frac{n-n^2}{2}+x_2\cdot n+x_1}{x_2-n}$$
$$\wedge\, x_1 > 0$$
$$\wedge\, \tfrac{n-1-(n-1)^2}{2} + x_2 \cdot (n-1) + x_1 > 0$$

as we have

$$(x_1 \geq x_1 + x_2) \equiv (0 \geq x_2) \implies (0 \geq x_2 - 1) \equiv (x_1 + x_2 \geq x_1 + x_2 + x_2 - 1).$$

△

Turning Theorem 10 into a conditional acceleration technique is straightforward.

Theorem 11 (Conditional Acceleration via Eventual Decrease) *If we have $\chi(\vec{x}) \equiv \bigwedge_{i=1}^{k} C_i$ where each clause C_i contains an inequation $e_i(\vec{x}) > 0$ such that*

$$\breve{\varphi}(\vec{x}) \wedge e_i(\vec{x}) \geq e_i(\vec{a}(\vec{x})) \implies e_i(\vec{a}(\vec{x})) \geq e_i(\vec{a}^2(\vec{x})), \quad (15)$$

then the following conditional acceleration technique is sound:

$$(\langle\chi,\vec{a}\rangle, \breve{\varphi}) \mapsto \Big(\vec{x}' = \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge e_i(\vec{a}^{n-1}(\vec{x})) > 0\Big) \quad (16)$$

If $C_i \equiv e_i > 0$ for all $i \in [1,k]$, then it is exact.

Proof For soundness, we need to show

$$\Big(\vec{x} \longrightarrow^{n}_{\langle\breve{\varphi},\vec{a}\rangle} \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge e_i(\vec{a}^{n-1}(\vec{x})) > 0\Big) \implies \vec{x} \longrightarrow^{n}_{\langle\chi,\vec{a}\rangle} \vec{a}^n(\vec{x}). \quad (17)$$

Assume

$$\vec{x} \longrightarrow^{n}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge e_i(\vec{a}^{n-1}(\vec{x})) > 0. \tag{18}$$

This implies

$$\bigwedge_{i=0}^{n-1} \breve{\varphi}(\vec{a}^i(\vec{x})). \tag{19}$$

In the following, we show

$$\bigwedge_{i=1}^{k} \bigwedge_{m=0}^{n-1} e_i(\vec{a}^m(\vec{x})) \geq \min(e_i(\vec{x}), e_i(\vec{a}^{n-1}(\vec{x}))). \tag{20}$$

Then the claim (17) follows, as we have

$$\begin{aligned}
&\bigwedge_{i=1}^{k} \bigwedge_{m=0}^{n-1} e_i(\vec{a}^m(\vec{x})) \geq \min(e_i(\vec{x}), e_i(\vec{a}^{n-1}(\vec{x}))) \\
\implies &\bigwedge_{i=1}^{k} \bigwedge_{m=0}^{n-1} e_i(\vec{a}^m(\vec{x})) > 0 && \text{(due to (18))} \\
\implies &\bigwedge_{m=0}^{n-1} \chi(\vec{a}^m(\vec{x})) && \text{(by definition of } e_i\text{)} \\
\iff &\vec{x} \longrightarrow^{n}_{\langle \chi, \vec{a} \rangle} \vec{a}^n(\vec{x}).
\end{aligned}$$

Let i be arbitrary but fixed, let $e = e_i$, and let j be the minimal natural number with

$$e(\vec{a}^j(\vec{x})) = \max\{e(\vec{a}^m(\vec{x})) \mid m \in [0, n-1]\}. \tag{21}$$

We first prove

$$e(\vec{a}^m(\vec{x})) < e(\vec{a}^{m+1}(\vec{x})) \tag{22}$$

for all $m \in [0, j-1]$ by backward induction on m. If $m = j-1$, then

$$\begin{aligned}
&e(\vec{a}^m(\vec{x})) \\
&= e(\vec{a}^{j-1}(\vec{x})) && \text{(as } m = j-1\text{)} \\
&< e(\vec{a}^j(\vec{x})) && \text{(due to (21) as } j \text{ is minimal)} \\
&= e(\vec{a}^{m+1}(\vec{x})). && \text{(as } m = j-1\text{)}
\end{aligned}$$

For the induction step, note that (15) implies

$$e(\vec{a}(\vec{x})) < e(\vec{a}^2(\vec{x})) \implies \neg\breve{\varphi}(\vec{x}) \vee e(\vec{x}) < e(\vec{a}(\vec{x})). \tag{23}$$

Then we have

$$\begin{aligned}
&e(\vec{a}^{m+1}(\vec{x})) < e(\vec{a}^{m+2}(\vec{x})) \\
&\qquad \text{(due to the induction hypothesis (22))} \\
\implies &\neg\breve{\varphi}(\vec{a}^m(\vec{x})) \vee e(\vec{a}^m(\vec{x})) < e(\vec{a}^{m+1}(\vec{x})) && \text{(by (23))} \\
\implies &e(\vec{a}^m(\vec{x})) < e(\vec{a}^{m+1}(\vec{x})). && \text{(by (19))}
\end{aligned}$$

Now we prove

$$e(\vec{a}^m(\vec{x})) \geq e(\vec{a}^{m+1}(\vec{x})) \tag{24}$$

for all $m \in [j, n-1]$ by induction on m. If $m = j$, then

$$\begin{aligned}
&e(\vec{a}^m(\vec{x})) \\
&= e(\vec{a}^j(\vec{x})) && \text{(as } m = j\text{)} \\
&= \max\{e(\vec{a}^m(\vec{x})) \mid m \in [0, n-1]\} && \text{(due to (21))} \\
&\geq e(\vec{a}^{j+1}(\vec{x})) \\
&= e(\vec{a}^{m+1}(\vec{x})). && \text{(as } m = j\text{)}
\end{aligned}$$

In the induction step, we have

$$\begin{aligned}
&e(\vec{a}^m(\vec{x})) \geq e(\vec{a}^{m+1}(\vec{x})) \\
&\qquad \text{(due to the induction hypothesis (24))} \\
\implies &e(\vec{a}^{m+1}(\vec{x})) \geq e(\vec{a}^{m+2}(\vec{x})). \\
&\qquad \text{(due to (19) and (15))}
\end{aligned}$$

As (22) and (24) imply

$$\bigwedge_{m=0}^{n-1} e(\vec{a}^m(\vec{x})) \geq \min(e(\vec{x}), e(\vec{a}^{n-1}(\vec{x}))),$$

this finishes the proof of (20) and hence shows (17).

For exactness, assume $\chi(\vec{x}) := \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0$. We have

$$\begin{aligned}
&\vec{x} \longrightarrow^{n}_{\langle \chi \wedge \breve{\varphi}, \vec{a} \rangle} \vec{a}^n(\vec{x}) \\
\implies &\chi(\vec{x}) \wedge \chi(\vec{a}^{n-1}(\vec{x})) \\
\iff &\bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge e_i(\vec{a}^{n-1}(\vec{x})) > 0.
\end{aligned}$$

□

Example 9 Consider the following variant of $\mathcal{T}_{ev\text{-}dec}$

$$\textbf{while } x_1 > 0 \wedge x_3 > 0 \textbf{ do } \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leftarrow \begin{pmatrix} x_1 + x_2 \\ x_2 - x_3 \\ x_3 \end{pmatrix},$$

i.e., we have $\vec{a} := \begin{pmatrix} x_1 + x_2 \\ x_2 - x_3 \\ x_3 \end{pmatrix}$. Starting with its canonical acceleration problem, we get the derivation shown in Figure 4, where the second step can be performed via Theorem 11 as

$$\begin{aligned}
&(\breve{\varphi}(\vec{x}) \wedge e(\vec{x}) \geq e(\vec{a}(\vec{x}))) \\
\equiv\ &(x_3 > 0 \wedge x_1 \geq x_1 + x_2) \\
\equiv\ &(x_3 > 0 \wedge 0 \geq x_2)
\end{aligned}$$

implies

$$\begin{aligned}
&(0 \geq x_2 - x_3) \\
\equiv\ &(x_1 + x_2 \geq x_1 + x_2 + x_2 - x_3) \\
\equiv\ &(e(\vec{a}(\vec{x})) \geq e(\vec{a}^2(\vec{x}))).
\end{aligned}$$

△

6.2 Acceleration via Eventual Increase

Still, all (combinations of) techniques presented so far fail for

$$\textbf{while } x_1 > 0 \textbf{ do } \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leftarrow \begin{pmatrix} x_1 + x_2 \\ x_2 + 1 \end{pmatrix}. \quad (\mathcal{T}_{ev\text{-}inc})$$

$$
\begin{aligned}
&[\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid x_1 > 0 \wedge x_3 > 0]\!]_{\vec{a}} \\
\rightsquigarrow_e\ &[\![\vec{x}' = \vec{a}^n(\vec{x}) \wedge x_3 > 0 \mid x_3 > 0 \mid x_1 > 0]\!]_{\vec{a}} && \text{(Theorem 8)} \\
\rightsquigarrow_e\ &\left[\!\!\left[\vec{x}' = \vec{a}^n(\vec{x}) \wedge x_3 > 0 \wedge x_1 > 0 \wedge x_1^{(n-1)} > 0 \;\middle|\; x_3 > 0 \wedge x_1 > 0 \;\middle|\; \top\right]\!\!\right]_{\vec{a}} && \text{(Theorem 11)}
\end{aligned}
$$

Fig. 4: $\rightsquigarrow$-derivation for Example 9

As in the case of $\mathcal{T}_{ev\text{-}dec}$, the value of x_1 does not behave monotonically, i.e., $x_1 > 0$ is neither an invariant nor a converse invariant. However, this time x_1 is eventually *increasing*, i.e., once x_1 starts to grow, it never decreases again. Thus, in this case it suffices to require that x_1 is positive and (weakly) increasing.

Theorem 12 (Acceleration via Eventual Increase) *If* $\varphi(\vec{x}) \equiv \bigwedge_{i=1}^{k} C_i$ *where each clause* C_i *contains an inequation* $e_i(\vec{x}) > 0$ *such that*

$$e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})) \implies e_i(\vec{a}(\vec{x})) \leq e_i(\vec{a}^2(\vec{x})),$$

then the following acceleration technique is sound:

$$\mathcal{T}_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} 0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x}))$$

We prove the more general Theorem 13 later in this section.

Example 10 With Theorem 12, we can accelerate $\mathcal{T}_{ev\text{-}inc}$ to

$$\binom{x_1'}{x_2'} = \binom{\frac{n^2-n}{2}+x_2\cdot n+x_1}{x_2+n} \wedge 0 < x_1 \leq x_1 + x_2 \qquad (\psi_{ev\text{-}inc})$$

as we have

$$(x_1 \leq x_1 + x_2) \equiv (0 \leq x_2) \implies (0 \leq x_2 + 1) \equiv (x_1 + x_2 \leq x_1 + x_2 + x_2 + 1).$$

△

However, Theorem 12 is *not* exact, as the resulting formula only covers program runs where each e_i behaves monotonically. So $\psi_{ev\text{-}inc}$ only covers those runs of $\mathcal{T}_{ev\text{-}inc}$ where the initial value of x_2 is non-negative.

Note that Theorem 12 can also lead to empty under-approximations. For example, Theorem 12 can be used to accelerate $\mathcal{T}_{exp}$, since the implication

$$x_1 \leq x_1 - 1 \implies x_1 - 1 \leq x_1 - 2$$

is valid. Then the resulting formula is

$$\binom{x_1'}{x_2'} = \binom{x_1 - n}{2^n \cdot x_2} \wedge 0 < x_1 \leq x_1 - 1,$$

which is unsatisfiable. Thus, when implementing Theorem 12 (or its conditional version Theorem 13), one has to check whether the resulting formula is satisfiable to avoid trivial (empty) under-approximations.

Again, turning Theorem 12 into a conditional acceleration technique is straightforward.

Theorem 13 (Conditional Acceleration via Eventual Increase) *If we have* $\chi(\vec{x}) \equiv \bigwedge_{i=1}^{k} C_i$ *where each clause* C_i *contains an inequation* $e_i(\vec{x}) > 0$ *such that*

$$\breve{\varphi}(\vec{x}) \wedge e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})) \implies e_i(\vec{a}(\vec{x})) \leq e_i(\vec{a}^2(\vec{x})), \quad (25)$$

then the following conditional acceleration technique is sound:

$$(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} 0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x}))$$

Proof We need to show

$$\vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^n(\vec{x}) \wedge \bigwedge_{i=1}^{k} 0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})) \implies \vec{x} \longrightarrow^n_{\langle \chi, \vec{a} \rangle} \vec{a}^n(\vec{x}).$$

Due to $\vec{x} \longrightarrow^n_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^n(\vec{x})$, we have

$$\bigwedge_{j=0}^{n-1} \breve{\varphi}(\vec{a}^j(\vec{x})). \quad (26)$$

Let i be arbitrary but fixed and assume

$$0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})). \quad (27)$$

We prove

$$e_i(\vec{a}^m(\vec{x})) \leq e_i(\vec{a}^{m+1}(\vec{x})) \quad (28)$$

for all $0 \leq m < n$ by induction on m. Then we get

$$0 < e_i(\vec{a}^m(\vec{x}))$$

and thus $\chi(\vec{a}^m(\vec{x}))$ for all $0 \leq m < n$ due to (27) and hence the claim follows. If $m = 0$, then

$$e_i(\vec{a}^m(\vec{x})) = e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})) = e_i(\vec{a}^{m+1}(\vec{x})).$$
(due to (27))

In the induction step, note that (26) implies

$$\breve{\varphi}(\vec{a}^m(\vec{x}))$$

as $0 \leq m < n$. Together with the induction hypothesis (28), we get

$$\breve{\varphi}(\vec{a}^m(\vec{x})) \wedge e_i(\vec{a}^m(\vec{x})) \leq e_i(\vec{a}^{m+1}(\vec{x})).$$

By (25), this implies

$$e_i(\vec{a}^{m+1}(\vec{x})) \leq e_i(\vec{a}^{m+2}(\vec{x})),$$

as desired. □

Example 11 Consider the following variant of $\mathcal{T}_{ev\text{-}inc}$.

$$\textbf{while } x_1 > 0 \wedge x_3 > 0 \textbf{ do } \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leftarrow \begin{pmatrix} x_1+x_2 \\ x_2+x_3 \\ x_3 \end{pmatrix}$$

So we have $\vec{a} := \begin{pmatrix} x_1+x_2 \\ x_2+x_3 \\ x_3 \end{pmatrix}$. Starting with the canonical acceleration problem, we get the derivation shown in Figure 5, where the second step can be performed via Theorem 13 as

$$\begin{aligned} &(\breve{\varphi}(\vec{x}) \wedge e(\vec{x}) \leq e(\vec{a}(\vec{x}))) \\ \equiv\ &(x_3 > 0 \wedge x_1 \leq x_1 + x_2) \\ \equiv\ &(x_3 > 0 \wedge 0 \leq x_2) \end{aligned}$$

implies

$$\begin{aligned} &(0 \leq x_2 + x_3) \\ \equiv\ &(x_1 + x_2 \leq x_1 + x_2 + x_2 + x_3) \\ \equiv\ &(e(\vec{a}(\vec{x})) \leq e(\vec{a}^2(\vec{x}))). \end{aligned}$$

△

We also considered versions of Theorems 11 and 13 where the inequations in (15) and (25) are strict, but this did not lead to an improvement in our experiments. Moreover, we experimented with a variant of Theorem 13 that splits the loop under consideration into two consecutive loops, accelerates them independently, and composes the results. While such an approach can accelerate loops like $\psi_{ev\text{-}inc}$ exactly, the impact on our experimental results was minimal. Thus, we postpone an in-depth investigation of this idea to future work.

7 Proving Non-Termination via Loop Acceleration

We now aim for proving *non-termination*.

Definition 7 ((Non-)Termination) We call a vector $\vec{x} \in \mathbb{Z}^d$ a *witness of non-termination* for $\mathcal{T}_{loop}$ (denoted $\vec{x} \longrightarrow^{\infty}_{\mathcal{T}_{loop}} \bot$) if

$$\forall n \in \mathbb{N}.\ \varphi(\vec{a}^n(\vec{x})).$$

If there is such a witness, then $\mathcal{T}_{loop}$ is *non-terminating*. Otherwise, $\mathcal{T}_{loop}$ *terminates*.

To this end, we search for a formula that characterizes a non-empty set of witnesses of non-termination, called a *certificate of non-termination*.

Definition 8 (Certificate of Non-Termination) We call a formula $\eta \in FO_{QF}(\mathscr{C}(\vec{x}))$ a *certificate of non-termination* for $\mathcal{T}_{loop}$ if η is satisfiable and

$$\forall \vec{x} \in \mathbb{Z}^d.\ \eta(\vec{x}) \implies \vec{x} \longrightarrow^{\infty}_{\mathcal{T}_{loop}} \bot.$$

Clearly, the loops $\mathcal{T}_{inc}$ and $\mathcal{T}_{ev\text{-}inc}$ that were used to motivate the acceleration techniques *Acceleration via Monotonic Increase* (Theorem 2) and *Acceleration via Eventual Increase* (Theorem 12) are non-terminating: $\mathcal{T}_{inc}$ diverges for all initial valuations that satisfy its guard $x > 0$ and $\mathcal{T}_{ev\text{-}inc}$ diverges if the initial values are sufficiently large, such that x_1 remains positive until x_2 becomes non-negative and hence x_1 starts to increase.

As we will see in the current section, this is not a coincidence: Unsurprisingly, all loops that can be accelerated with *Acceleration via Monotonic Increase* or *Acceleration via Eventual Increase* are non-terminating. More interestingly, the same holds for all loops that can be accelerated using our calculus from Section 4, as long as all $\rightsquigarrow$-steps use one of the conditional versions of the aforementioned acceleration techniques, i.e., *Conditional Acceleration via Monotonic Increase* (Theorem 8) or *Conditional Acceleration via Eventual Increase* (Theorem 13). Thus, we obtain a novel, modular technique for proving non-termination of loops $\mathcal{T}_{loop}$.

$$
\begin{aligned}
& [\![\vec{x}' = \vec{a}^n(\vec{x}) \mid \top \mid x_1 > 0 \wedge x_3 > 0]\!]_{\vec{a}} \\
\rightsquigarrow_e\ & [\![\vec{x}' = \vec{a}^n(\vec{x}) \wedge x_3 > 0 \mid x_3 > 0 \mid x_1 > 0]\!]_{\vec{a}} && \text{(Theorem 8)} \\
\rightsquigarrow\ & [\![\vec{x}' = \vec{a}^n(\vec{x}) \wedge x_3 > 0 \wedge 0 < x_1 \leq x_1 + x_2 \mid x_3 > 0 \wedge x_1 > 0 \mid \top]\!]_{\vec{a}} && \text{(Theorem 13)}
\end{aligned}
$$

Fig. 5: $\rightsquigarrow$-derivation for Example 11

Recall that derivations of our calculus from Section 4 start with *canonical acceleration problems* (Definition 4) whose first component is

$$\vec{x}' = \vec{a}^n(\vec{x}).$$

It relates the values of the variables before evaluating the loop ($\vec{x}$) to the values of the variables after evaluating the loop ($\vec{x}'$) using the closed form ($\vec{a}^n$). However, if we are interested in non-terminating runs, then the values of the variables after evaluating the loop are obviously irrelevant. Hence, attempts to prove non-termination operate on a variation of *acceleration problems*, which we call *non-termination problems*.

Definition 9 (Non-Termination Problem) A tuple

$$\|\psi \mid \breve{\varphi} \mid \widehat{\varphi}\|_{\vec{a}}$$

where $\psi, \breve{\varphi}, \widehat{\varphi} \in FO_{QF}(\mathscr{C}(\vec{x}))$ and $\vec{a} : \mathbb{Z}^d \to \mathbb{Z}^d$ is a *non-termination problem*. It is *consistent* if every model of ψ is a witness of non-termination for $\langle \breve{\varphi}, \vec{a} \rangle$ and *solved* if it is consistent and $\widehat{\varphi} \equiv \top$. The *canonical non-termination problem* of a loop $\mathcal{T}_{loop}$ is

$$\|\top \mid \top \mid \varphi\|_{\vec{a}}\,.$$

In particular, this means that the technique presented in the current section can also be applied to loops where $\vec{a}^n$ cannot be expressed in closed form.

Example 12 The canonical non-termination problem of $\mathcal{T}_{ev\text{-}inc}$ is

$$\|\top \mid \top \mid x_1 > 0\|_{\left(\begin{smallmatrix} x_1+x_2 \\ x_2+1 \end{smallmatrix}\right)}\,.$$

△

We use a variation of *conditional acceleration techniques* (Definition 5), which we call *conditional non-termination techniques*, to simplify the canonical non-termination problem of the analyzed loop.

Definition 10 (Conditional Non-Termination Technique) We call a partial function

$$nt : Loop \times FO_{QF}(\mathscr{C}(\vec{x})) \rightharpoonup FO_{QF}(\mathscr{C}(\vec{x}))$$

a *conditional non-termination technique* if

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge nt(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \quad \text{implies} \quad \vec{x} \longrightarrow^{\infty}_{\langle \chi, \vec{a} \rangle} \bot$$

for all $(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \in \mathrm{dom}(nt)$ and all $\vec{x} \in \mathbb{Z}^d$.

Thus, we obtain the following variation of our calculus from Section 4.

Definition 11 (Non-Termination Calculus) The relation $\rightsquigarrow_{nt}$ on non-termination problems is defined by the rule

$$\frac{\emptyset \neq \chi \subseteq \widehat{\varphi} \quad nt(\langle \chi, \vec{a} \rangle, \breve{\varphi}) = \psi_2}{\|\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}\|_{\vec{a}} \rightsquigarrow_{nt} \|\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi\|_{\vec{a}}}$$

where nt is a conditional non-termination technique.

Like $\rightsquigarrow$, the relation $\rightsquigarrow_{nt}$ preserves consistency. Hence, we obtain the following theorem, which shows that our calculus is indeed suitable for proving non-termination.

Theorem 14 (Correctness of $\rightsquigarrow_{nt}$) *If*

$$\|\top \mid \top \mid \varphi\|_{\vec{a}} \rightsquigarrow^*_{nt} \|\psi \mid \breve{\varphi} \mid \top\|_{\vec{a}}\,,$$

and ψ is satisfiable, then ψ is a certificate of non-termination for $\mathcal{T}_{loop}$.

Proof We prove that our calculus preserves consistency, then the claim follows immediately. Assume

$$\|\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}\|_{\vec{a}} \rightsquigarrow_{nt} \|\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi\|_{\vec{a}}$$

where $\|\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}\|_{\vec{a}}$ is consistent and

$$nt(\langle \chi, \vec{a} \rangle, \breve{\varphi}) = \psi_2.$$

We get

$$\begin{aligned} & \psi_1 \wedge \psi_2 \\ \Longrightarrow\ & \vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \psi_2 \end{aligned}$$

$$\implies \vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \vec{x} \longrightarrow^{\infty}_{\langle \chi, \vec{a} \rangle} \bot$$
$$\iff \vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi} \wedge \chi, \vec{a} \rangle} \bot$$

The first step holds since $\|\psi_1 \mid \breve{\varphi} \mid \widehat{\varphi}\|_{\vec{a}}$ is consistent and the second step holds since nt is a conditional non-termination technique. This proves consistency of

$$\|\psi_1 \wedge \psi_2 \mid \breve{\varphi} \wedge \chi \mid \widehat{\varphi} \setminus \chi\|_{\vec{a}}$$
$$= \|\psi_1 \cup \psi_2 \mid \breve{\varphi} \cup \chi \mid \widehat{\varphi} \setminus \chi\|_{\vec{a}} .$$

□

Analogously to well-foundedness of $\rightsquigarrow$, well-foundedness of $\rightsquigarrow_{nt}$ is trivial.

Theorem 15 (Well-Foundedness of $\rightsquigarrow_{nt}$) *The relation $\rightsquigarrow_{nt}$ is well-founded.*

It remains to present non-termination techniques that can be used with our novel calculus. We first derive a non-termination technique from *Conditional Acceleration via Monotonic Increase* (Theorem 8).

Theorem 16 (Non-Termination via Monotonic Increase) *If*

$$\breve{\varphi}(\vec{x}) \wedge \chi(\vec{x}) \implies \chi(\vec{a}(\vec{x})),$$

then

$$(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \mapsto \chi$$

is a conditional non-termination technique.

Proof We need to prove

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi(\vec{x}) \implies \vec{x} \longrightarrow^{\infty}_{\langle \chi, \vec{a} \rangle} \bot.$$

To this end, it suffices to prove

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi(\vec{x}) \implies \forall m \in \mathbb{N}.\ \chi(\vec{a}^m(\vec{x}))$$

by the definition of non-termination (Definition 7). Assume

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi(\vec{x}).$$

We prove $\chi(\vec{a}^m(\vec{x}))$ for all $m \in \mathbb{N}$ by induction on m. If $m = 0$, then the claim follows immediately. For the induction step, note that $\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot$ implies $\vec{x} \longrightarrow^{m+1}_{\langle \breve{\varphi}, \vec{a} \rangle} \vec{a}^{m+1}(\vec{x})$, which in turn implies $\breve{\varphi}(\vec{a}^m(\vec{x}))$. Together with the induction hypothesis $\chi(\vec{a}^m(\vec{x}))$, the claim follows from the prerequisites of the theorem.

□

Example 13 The canonical non-termination problem of $\mathcal{T}_{inc}$ is

$$\|\top \mid \top \mid x > 0\|_{(x+1)} .$$

Thus, in order to apply $\rightsquigarrow_{nt}$ with Theorem 16, the only possible choice for the formula χ is $x > 0$. Furthermore, we have $\breve{\varphi} := \top$ and $\vec{a} := (x + 1)$. Hence, Theorem 16 is applicable if the implication

$$\top \wedge x > 0 \implies x + 1 > 0$$

is valid, which is clearly the case. Thus, we get

$$\|\top \mid \top \mid x > 0\|_{(x+1)} \rightsquigarrow_{nt} \|x > 0 \mid x > 0 \mid \top\|_{(x+1)} .$$

Since the latter non-termination problem is solved and $x > 0$ is satisfiable, $x > 0$ is a certificate of non-termination for $\mathcal{T}_{inc}$ due to Theorem 14. △

Clearly, Theorem 16 is only applicable in very simple cases. To prove non-termination of more complex examples, we now derive a conditional non-termination technique from *Conditional Acceleration via Eventual Increase* (Theorem 13).

Theorem 17 (Non-Termination via Eventual Increase) *If we have $\chi(\vec{x}) \equiv \bigwedge_{i=1}^{k} C_i$ where each clause C_i contains an inequation $e_i(\vec{x}) > 0$ such that*

$$\breve{\varphi}(\vec{x}) \wedge e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})) \implies e_i(\vec{a}(\vec{x})) \leq e_i(\vec{a}^2(\vec{x})),$$

then

$$(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \mapsto \bigwedge_{i=1}^{k} 0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x}))$$

is a conditional non-termination technique.

Proof Let $\chi' := \bigwedge_{i=1}^{k} 0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x}))$. We need to prove

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi' \implies \vec{x} \longrightarrow^{\infty}_{\langle \chi, \vec{a} \rangle} \bot.$$

Then it suffices to prove

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi'(\vec{x}) \implies \vec{x} \longrightarrow^{\infty}_{\langle \chi', \vec{a} \rangle} \bot \qquad (29)$$

since χ' implies χ. By the prerequisites of the theorem, we have $\breve{\varphi} \wedge \chi'(\vec{x}) \implies \chi'(\vec{a}(\vec{x}))$. Thus, Theorem 16 applies to $\langle \chi', \vec{a} \rangle$. Hence, the claim (29) follows. □

Example 14 We continue Example 12. To apply $\rightsquigarrow_{nt}$ with Theorem 17 to the canonical non-termination problem of $\mathcal{T}_{ev\text{-}inc}$, the only possible choice for the formula χ is $x_1 > 0$. Moreover, we again have $\breve{\varphi} := \top$, and $\vec{a} := \begin{pmatrix} x_1 + x_2 \\ x_2 + 1 \end{pmatrix}$. Thus, Theorem 17 is applicable if

$$\top \wedge x_1 \leq x_1 + x_2 \implies x_1 + x_2 \leq x_1 + 2 \cdot x_2 + 1$$

is valid. Since we have $x_1 \leq x_1 + x_2 \iff x_2 \geq 0$ and $x_1 + x_2 \leq x_1 + 2 \cdot x_2 + 1 \iff x_2 + 1 \geq 0$, this is clearly the case. Hence, we get

$$\|\top \mid \top \mid x_1 > 0\|_{\vec{a}}$$

$$\rightsquigarrow_{nt} \|0 < x_1 \leq x_1 + x_2 \mid x_1 > 0 \mid \top\|_{\vec{a}}.$$

Since $0 < x_1 \leq x_1 + x_2 \equiv x_1 > 0 \wedge x_2 \geq 0$ is satisfiable, $x_1 > 0 \wedge x_2 \geq 0$ is a certificate of non-termination for $\mathcal{T}_{ev\text{-}inc}$ due to Theorem 14. △

Of course, some non-terminating loops do not behave (eventually) monotonically, as the following example illustrates.

Example 15 Consider the loop

$$\textbf{while } x_1 > 0 \textbf{ do } \binom{x_1}{x_2} \leftarrow \binom{x_2}{x_1}. \qquad (\mathcal{T}_{fixpoint})$$

Theorem 16 is inapplicable, since

$$x_1 > 0 \implies x_2 > 0.$$

Furthermore, Theorem 17 is inapplicable, since

$$x_1 > x_2 \implies x_2 > x_1.$$

△

However, $\mathcal{T}_{fixpoint}$ has *fixpoints*, i.e., there are valuations such that $\vec{x} = \vec{a}(\vec{x})$. Therefore, it can be handled by existing approaches like [21, Thm. 12]. As the following theorem shows, such techniques can also be embedded into our calculus.

Theorem 18 (Non-Termination via Fixpoints) *For each entity e, let $\mathcal{V}(e)$ denote the set of variables occurring in e. Moreover, we define*

$$closure_{\vec{a}}(e) := \bigcup_{n \in \mathbb{N}} \mathcal{V}(\vec{a}^n(e)).$$

Let $\chi(\vec{x}) \equiv \bigwedge_{i=1}^{k} C_i$, and for each $i \in [1, k]$, assume that $e_i(\vec{x}) > 0$ is an inequation that occurs in C_i. Then

$$(\langle \chi, \vec{a} \rangle, \breve{\varphi}) \mapsto \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge \bigwedge_{x_j \in closure_{\vec{a}}(e_i)} x_j = \vec{a}(\vec{x})_j$$

is a conditional non-termination technique.

Proof Let

$$\chi' := \bigwedge_{i=1}^{k} e_i(\vec{x}) > 0 \wedge \bigwedge_{x_j \in closure_{\vec{a}}(e_i)} x_j = \vec{a}(\vec{x})_j.$$

We need to prove

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi' \implies \vec{x} \longrightarrow^{\infty}_{\langle \chi, \vec{a} \rangle} \bot.$$

Then it suffices to prove

$$\vec{x} \longrightarrow^{\infty}_{\langle \breve{\varphi}, \vec{a} \rangle} \bot \wedge \chi'(\vec{x}) \implies \vec{x} \longrightarrow^{\infty}_{\langle \chi', \vec{a} \rangle} \bot \qquad (30)$$

since χ' implies χ. By construction, we have

$$\chi'(\vec{x}) \implies \chi'(\vec{a}(\vec{x})).$$

Thus, Theorem 16 applies to $\langle \chi', \vec{a} \rangle$. Hence, the claim (30) follows. □

Example 16 We continue Example 15 by showing how to apply Theorem 18 to $\mathcal{T}_{fixpoint}$, i.e., we have $\chi := x_1 > 0$, $\breve{\varphi} := \top$, and $\vec{a} := \binom{x_2}{x_1}$. Thus, we get

$$closure_{\vec{a}}(x_1 > 0) = \{x_1, x_2\}.$$

So starting with the canonical non-termination problem of $\mathcal{T}_{fixpoint}$, we get

$$\|\top \mid \top \mid x_1 > 0\|_{\binom{x_2}{x_1}}$$
$$\rightsquigarrow_{nt} \|x_1 > 0 \wedge x_1 = x_2 \mid x_1 > 0 \mid \top\|_{\binom{x_2}{x_1}}.$$

Since the formula $x_1 > 0 \wedge x_1 = x_2$ is satisfiable, $x_1 > 0 \wedge x_1 = x_2$ is a certificate of non-termination for $\mathcal{T}_{fixpoint}$ by Theorem 14. △

Like the acceleration techniques from Theorems 12 and 13, the non-termination techniques from Theorems 17 and 18 can result in empty under-approximations. Thus, when integrating these techniques into our calculus, one should check the resulting formula for satisfiability after each step to detect fruitless derivations early.

We conclude this section with a more complex example, which shows how the presented non-termination techniques can be combined to find certificates of non-termination.

Example 17 Consider the following loop:

$$\textbf{while } x_1 > 0 \wedge x_3 > 0 \wedge x_4 + 1 > 0 \textbf{ do}$$
$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \leftarrow \begin{pmatrix} 1 \\ x_2 + x_1 \\ x_3 + x_2 \\ -x_4 \end{pmatrix}$$

So we have

$$\varphi := x_1 > 0 \wedge x_3 > 0 \wedge x_4 + 1 > 0$$

and

$$\vec{a} := \begin{pmatrix} 1 \\ x_2 + x_1 \\ x_3 + x_2 \\ -x_4 \end{pmatrix}.$$

Then the canonical non-termination problem is

$$\|\top \mid \top \mid x_1 > 0 \wedge x_3 > 0 \wedge x_4 + 1 > 0\|_{\vec{a}}.$$

First, our implementation applies Theorem 16 to $x_1 > 0$ (as $x_1 > 0 \implies 1 > 0$), resulting in

$$\|x_1 > 0 \mid x_1 > 0 \mid x_3 > 0 \wedge x_4 + 1 > 0\|_{\vec{a}}.$$

Next, it applies Theorem 17 to $x_3 > 0$, which is possible since

$$x_1 > 0 \wedge x_3 \leq x_3 + x_2 \implies x_3 + x_2 \leq x_3 + 2 \cdot x_2 + x_1$$

is valid. Note that this implication breaks if one removes $x_1 > 0$ from the premise, i.e., Theorem 17 does not apply to $x_3 > 0$ without applying Theorem 16 to $x_1 > 0$ before. This shows that our calculus is

more powerful than "the sum" of the underlying non-termination techniques. Hence, we obtain the following non-termination problem:

$$\|x_1 > 0 \wedge x_2 \geq 0 \wedge x_3 > 0 \mid x_1 > 0 \wedge x_3 > 0 \mid x_4 + 1 > 0\|_{\vec{a}}$$

Here, we simplified

$$0 < x_3 \leq x_3 + x_2$$

to

$$x_2 \geq 0 \wedge x_3 > 0.$$

Finally, neither Theorem 16 nor Theorem 17 applies to $x_4 + 1 > 0$, since x_4 does not behave (eventually) monotonically: Its value after n iterations is given by $(-1)^n \cdot x_4^{init}$, where x_4^{init} denotes the initial value of x_4. Thus, we apply Theorem 18 and we get

$$\|x_1 > 0 \wedge x_2 \geq 0 \wedge x_3 > 0 \wedge x_4 = 0 \mid \varphi \mid \top\|_{\vec{a}},$$

which is solved. Here, we simplified the sub-formula $x_4 + 1 > 0 \wedge x_4 = -x_4$ that results from the last acceleration step to $x_4 = 0$.

This shows that our calculus allows for applying Theorem 18 to loops that do not have a fixpoint. The reason is that it suffices to require that a *subset* of the program variables remain unchanged, whereas the values of other variables may still change.

Since

$$x_1 > 0 \wedge x_2 \geq 0 \wedge x_3 > 0 \wedge x_4 = 0$$

is satisfiable, it is a certificate of non-termination due to Theorem 14. △

8 Related Work

The related work for our paper splits into papers on acceleration and papers on methods specifically designed to prove non-termination. In both cases, one major difference between our approach and the approaches in the literature is that we enable a *modular* analysis that allows for combining completely unrelated acceleration or non-termination techniques to process a loop in an iterative way and to reuse information obtained by earlier proof steps.

8.1 Related Work on Acceleration

Acceleration-like techniques are also used in *over-approximating* settings (see, e.g., [20, 32, 33, 40, 41, 46, 49, 51]), whereas we consider *exact* and *under-approximating* loop acceleration techniques. As many related approaches have already been discussed in Section 3, we only mention three more techniques here.

First, [6, 9] presents an exact acceleration technique for *finite monoid affine transformations* (FMATs), i.e., loops with linear arithmetic whose body is of the form $\vec{x} \leftarrow A\vec{x} + \vec{b}$ where $\{A^i \mid i \in \mathbb{N}\}$ is finite. For such loops, Presburger-Arithmetic is sufficient to construct an equivalent formula ψ, i.e., it can be expressed in a decidable logic. In general, this is clearly not the case for the techniques presented in the current paper (which may even synthesize non-polynomial closed forms, see $\mathcal{T}_{exp}$). As a consequence and in contrast to our technique, this approach cannot handle loops where the values of variables grow super-linearly (i.e., it cannot handle examples like $\mathcal{T}_{2\text{-}invs}$). Implementations are available in the tools FAST [3] and Flata [38]. Further theoretical results on linear transformations whose n-fold closure is definable in (extensions of) Presburger-Arithmetic can be found in [7].

Second, [8] shows that *octagonal relations* can be accelerated exactly. Such relations are defined by a finite conjunction ξ of inequations of the form $\pm x \pm y \leq c$, $x, y \in \vec{x} \cup \vec{x}'$, $c \in \mathbb{Z}$. Then ξ induces the relation $\vec{x} \longrightarrow_\xi \vec{x}' \iff \xi(\vec{x}, \vec{x}')$. In [42], it is proven that such relations can even be accelerated in polynomial time. This generalizes earlier results for *difference bound constraints* [17]. As in the case of FMATs, the resulting formula can be expressed in Presburger-Arithmetic. In contrast to the loops considered in the current paper where $\vec{x}'$ is uniquely determined by $\vec{x}$, octagonal relations can represent non-deterministic programs. Therefore and due to the restricted form of octagonal relations, the work from [8, 42] is orthogonal to ours.

Third, Albert et al. recently presented a technique to find metering functions via *loop specialization*, which is automated via MAX-SMT solving [1]. This approach could be integrated into our framework via Theorem 9. However, the technique from [1] focuses on multi-path loops, whereas we focus on single-path loops. One of the main reasons for our restriction to single-path loops is that their closed form (Definition 2) can often be computed automatically in practice. In contrast, for multi-path loops, it is less clear how to obtain closed forms.

8.2 Related Work on Proving Non-Termination

In the following, we focus on approaches for proving non-termination of programs that operate on (unbounded) integer numbers as data.

Many approaches to proving non-termination are based on finding recurrent sets [35]. A recurrent set describes program configurations from which one can take a step to a configuration that is also in the recurrent set. Thus, a recurrent set that includes an initial configuration implies non-termination of the program. In the setting of this paper, a certificate of non-termination $\psi(\vec{x})$ for a loop $\langle \varphi, \vec{a} \rangle$ induces the recurrent set

$$\{\vec{a}^n(\vec{x}) \mid n \in \mathbb{N} \wedge \vec{x} \in \mathbb{Z}^d \wedge \psi(\vec{x})\}.$$

As long as our calculus is used with the non-termination techniques presented in the current paper only (i.e., Theorems 16 to 18), it even holds that $\{\vec{x} \in \mathbb{Z}^d \mid \psi(\vec{x})\}$ is a recurrent set. Conversely, a formula characterizing a non-empty recurrent set of a loop is also a certificate of non-termination. Thus, our calculus could also make use of other non-termination techniques that produce recurrent sets expressed by formulas in $FO_{QF}(\mathscr{C}(\vec{x}))$.

Recurrent sets are often synthesized by searching for suitable parameter values for template formulas [14, 18, 35, 44, 45, 55] or for safety proofs [16, 34, 54]. In contrast to these search-based techniques, our current techniques use constraint solving only to check implications. As also indicated by our experiments (Section 9), this aspect leads to low runtimes and an efficient analysis.

Many proof techniques for proving non-termination of programs [11, 18, 35, 45] work by proving that some loop is non-terminating and (often separately) proving that a witness of non-termination for this loop is reachable from an initial program configuration. This captures *lasso-shaped* counterexamples to program termination. A *lasso* consists of a *stem* of straight-line code (which could be expressed as a single update), followed by a loop with a single update in its body. Techniques for proving non-termination of loops that provide witnesses of non-termination can thus be lifted to techniques for lassos by checking reachability of the found witnesses of non-termination from an initial program configuration. While the presentation in this paper focuses on loops, our implementation in LoAT can also prove that the found certificate of non-termination for the loop is reachable from an initial program configuration. If a loop cannot be proven non-terminating, it can still be possible to accelerate it and use the accelerated loop as part of the stem of a lasso for another loop. Like this, LoAT can analyze programs with more complex control flow than just single loops.

Several techniques for proving *aperiodic* non-termination, i.e., non-termination of programs that do not have any lasso-shaped counterexamples to termination, have been proposed [11, 14, 16, 34, 44]. By integrating our calculus into a suitable program-analysis framework [21, 26], it can be used to prove aperiodic non-termination as well.

Loop termination was recently proven to be decidable for the subclass of loops in which the guards and updates use only linear arithmetic and the guards are restricted to conjunctions of atoms [24, 39]. Our approach is less restrictive regarding the input loops: we allow for non-linear guards and updates, and we allow for arbitrary Boolean structure in the guards. In future work, one could investigate the use of such decision procedures as conditional non-termination techniques in our calculus to make them applicable to larger classes of loops. For practical applications to larger programs, it is important to obtain a certificate of non-termination for a loop when proving its non-termination, corresponding to a large, ideally infinite set of witnesses of non-termination. The reason is that some witness of non-termination for the loop must be reachable from an initial program configuration so that the non-termination proof carries over from the loop to the input program. However, the decision procedures in [24, 39] are not optimized to this end: They produce a certificate of *eventual non-termination*, i.e., a formula that describes initial configurations that give rise to witnesses of non-termination by applying the loop body a finite, but unknown number of times. For example, the most general certificate of non-termination for the loop $\mathcal{T}_{inc}$ is $x > 0$, whereas the most general certificate of eventual non-termination is $\top$. The reason is that, for any initial valuation $-c$ of x (where c is a natural number), one obtains a witness of non-termination by applying the body of the loop $c + 1$ times while ignoring the loop condition. The problem of transforming a *single* witness of eventual non-termination into a witness of non-termination has

been solved partially in [36]. The problem of transforming certificates of eventual non-termination that describe *infinite* sets of configurations into certificates of non-termination is, to the best of our knowledge, still open. In contrast, the conditional non-termination techniques presented in Section 7 aim to identify a potentially infinite set of witnesses of non-termination.

For a subclass of loops involving non-linearity and arbitrary Boolean structures, decidability of termination has recently been proven, too [25]. However, the decidability proof from [25] only applies to loops where the variables range over $\mathbb{R}$. For loops over $\mathbb{Z}$, termination of such programs is trivially undecidable (due to Hilbert's tenth problem).

Ben-Amram, Doménech, and Genaim [5] show a connection between the non-existence of *multiphase-linear ranking functions* as termination arguments for linear loops and *monotonic* recurrent sets. A recurrent set

$$R := \left\{ \vec{x} \in \mathbb{Z}^d \;\middle|\; \bigwedge_{i=1}^{m} e_i(\vec{x}) > 0 \right\}$$

is monotonic if we have $e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x}))$ for all $i \in [1, m]$ and all $\vec{x} \in R$. In particular, they propose a procedure that, if it terminates, returns either a multiphase-linear ranking function as a termination proof or a set of program states that could not be proven terminating. If the input loop has a linear update function with only integer coefficients and if the procedure returns a non-empty set of states that includes integer values, this set is a monotonic recurrent set and proves non-termination. This approach is implemented in the iRankFinder tool.

Leike and Heizmann [45] propose a method to find geometric non-termination arguments that allow for expressing the values of the variables after the n^{th} loop iteration. In this sense, their approach can also be seen as a use of loop acceleration to express a non-termination argument. This approach is implemented in the Ultimate tool. While our technique for loop acceleration also relies on closed forms, our technique for proving non-termination does not need closed forms. Hence our approach also applies to loops where closed forms cannot be computed, or contain subexpressions that make further analyses difficult, like the imaginary unit.

Finally, Frohn and Giesl [21] have already used loop acceleration for proving non-termination. However, they use loop acceleration (more specifically, Theorem 3) solely for proving reachability of non-terminating loops. To prove non-termination of loops, they used unconditional, standalone versions of Theorems 16 and 18. So their approach does not allow for combining different acceleration or non-termination techniques when analyzing loops. However, they already exploited similarities between non-termination proving and loop acceleration: Both their loop acceleration technique (Theorem 3) and their main non-termination technique (Theorem 16) are based on certain monotonicity properties of the loop condition. Starting from this observation, they developed a technique for deducing invariants that may be helpful for both proving non-termination and accelerating loops. This idea is orthogonal to our approach, which could, of course, be combined with techniques for invariant inference.

9 Implementation and Experiments

We implemented our approach in our open-source Loop Acceleration Tool LoAT [21, 26]:
https://aprove-developers.github.io/LoAT
It uses Z3 [47] and Yices2 [19] to check implications and PURRS [2] to compute closed forms. While LoAT can synthesize formulas with non-polynomial arithmetic, it cannot yet parse them, i.e., the input is restricted to polynomials. Moreover, LoAT does not yet support disjunctive loop conditions.

To evaluate our approach, we extracted 1511 loops with conjunctive guards from the category *Termination of Integer Transition Systems* of the *Termination Problems Database* [53], the benchmark collection which is used at the annual *Termination and Complexity Competition* [30], as follows:

1. We parsed all examples with LoAT and extracted each single-path loop with conjunctive guard (resulting in 3829 benchmarks).
2. We removed duplicates by checking syntactic equality (resulting in 2825 benchmarks).
3. We removed loops whose runtime is trivially constant using an incomplete check (resulting in 1733 benchmarks).

4. We removed loops which do not admit any terminating runs, i.e., loops where Theorem 2 applies (resulting in 1511 benchmarks).

All tests have been run on StarExec [52] (Intel Xeon E5-2609, 2.40GHz, 264GB RAM [50]). For our benchmark collection, more details about the results of our evaluation, and a pre-compiled binary (Linux, 64 bit) we refer to [23].

9.1 Loop Acceleration

For technical reasons, the closed forms computed by LoAT are valid only if $n > 0$, whereas Definition 2 requires them to be valid for all $n \in \mathbb{N}$. The reason is that PURRS has only limited support for initial conditions. Thus, LoAT's results are correct only for all $n > 1$ (instead of all $n > 0$). Moreover, LoAT can currently compute closed forms only if the loop body is *triangular*, meaning that each a_i is an expression over $x_1, \dots, x_i$. The reason is that PURRS cannot solve *systems* of recurrence equations, but only a single recurrence equation at a time. While systems of recurrence equations can be transformed into a single recurrence equation of higher order, LoAT does not yet implement this transformation. However, LoAT failed to compute closed forms for just 26 out of 1511 loops in our experiments, i.e., this appears to be a minor restriction in practice. Furthermore, the implementation of our calculus does not use *conditional acceleration via metering functions*. The reason is that we are not aware of examples where our monotonicity-based acceleration techniques fail, but our technique for finding metering functions (based on Farkas' Lemma) may succeed.

Apart from these differences, our implementation closely follows the current paper. It applies the conditional acceleration techniques from Sections 5 and 6 with the following priorities: Theorem 8 > Theorem 7 > Theorem 11 > Theorem 13.

We compared our implementation with LoAT's implementation of *acceleration via monotonicity* (Theorem 3, [21]) and its implementation of *acceleration via metering functions* (Theorem 4, [27]), which also incorporates the improvements proposed in [26]. We did not include the techniques from Theorems 1 and 2 in our evaluation, as they are subsumed by *acceleration via monotonicity*.

Furthermore, we compared with Flata [38], which implements the techniques to accelerate FMATs and octagonal relations discussed in Section 8. To this end, we used a straightforward transformation from LoAT's native input format[4] (which is also used in the category *Complexity of Integer Transition Systems* of the *Termination and Complexity Competition*) to Flata's input format. Note that our benchmark collection contains 16 loops with non-linear arithmetic where Flata is bound to fail, since it supports only linear arithmetic. We did not compare with FAST [3], which uses a similar approach as the more recent tool Flata. We used a wall clock timeout of 60s per example and a memory limit of 128GB for each tool.

The results can be seen in Table 1, where the information regarding the runtime includes all examples, not just solved examples. They show that our novel calculus was superior to the competing techniques in our experiments. In all but 7 cases where our calculus successfully accelerated the given loop, the resulting formula was polynomial. Thus, integrating our approach into existing acceleration-based verification techniques should not present major obstacles w.r.t. automation.

Furthermore, we evaluated the impact of our new acceleration techniques from Section 6 independently. To this end, we ran experiments with three configurations where we disabled *acceleration via eventual increase*, *acceleration via eventual decrease*, and both of them. The results can be seen in Table 2. They show that our calculus does not improve over *acceleration via monotonicity* if both *acceleration via eventual increase* and *acceleration via eventual decrease* are disabled (i.e., our benchmark collection does not contain examples like $\mathcal{T}_{\textit{2-c-invs}}$). However, enabling either *acceleration via eventual decrease* or *acceleration via eventual increase* resulted in a significant improvement. Interestingly, there are many examples that can be accelerated with either of these two techniques: When both of them were enabled, LoAT (exactly or approximately) accelerated 1482 loops. When only one of them was enabled, it accelerated 1444 and 1338 loops, respectively. But when none of them was enabled, it accelerated only 845 loops. We believe that this is due to examples like

$$\textbf{while } x_1 > 0 \wedge \dots \textbf{ do } \begin{pmatrix} x_1 \\ x_2 \\ \dots \end{pmatrix} \leftarrow \begin{pmatrix} x_2 \\ x_2 \\ \dots \end{pmatrix} \tag{31}$$

[4] https://github.com/aprove-developers/LoAT#koat

	LoAT	Monot.	Meter	Flata
exact	1444	845	0[5]	1231
approx	38	0	733	0
fail	29	666	778	280
avg rt	0.16s	0.11s	0.09s	0.47s
median rt	0.09s	0.09s	0.09s	0.40s
std dev rt	0.18s	0.09s	0.03s	0.50s

Table 1: LoAT vs. other techniques

	~~Ev-Inc~~	~~Ev-Dec~~	~~Ev-Mon~~
exact	1444	845	845
approx	0	493	0
fail	67	173	666
avg rt	0.15s	0.14s	0.09s
median rt	0.08s	0.08s	0.07s
std dev rt	0.17s	0.17s	0.06s

Table 2: Impact of our new acceleration techniques

LoAT:	Acceleration calculus + Theorems 7, 8, 11 and 13
Monot.:	*Acceleration via Monotonicity*, Theorem 3
Meter:	*Acceleration via Metering Functions*, Theorem 4
Flata:	http://nts.imag.fr/index.php/Flata
~~Ev-Inc~~:	Acceleration calculus + Theorems 7, 8 and 11
~~Ev-Dec~~:	Acceleration calculus + Theorems 7, 8 and 13
~~Ev-Mon~~:	Acceleration calculus + Theorems 7 and 8
exact:	Examples that were accelerated *exactly*
approx:	Examples that were accelerated *approximately*
fail:	Examples that could not be accelerated
avg rt:	Average wall clock runtime
median rt:	Median wall clock runtime
st dev rt:	Standard deviation of wall clock runtime

where Theorem 11 *and* Theorem 13 are applicable (since $x_1 \leq x_2$ implies $x_2 \leq x_2$ and $x_1 \geq x_2$ implies $x_2 \geq x_2$).

Flata exactly accelerated 49 loops where LoAT failed or approximated and LoAT exactly accelerated 262 loops where Flata failed. So there were only 18 loops where both Flata and the full version of our calculus failed to compute an exact result. Among them were the only 3 examples where our implementation found a closed form, but failed anyway. One of them was[6]

$$\textbf{while } x_3 > 0 \textbf{ do } \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leftarrow \begin{pmatrix} x_1+1 \\ x_2-x_1 \\ x_3+x_2 \end{pmatrix}.$$

Here, the updated value of x_1 depends on x_1, the update of x_2 depends on x_1 and x_2, and the update of x_3 depends on x_2 and x_3. Hence, the closed form of x_1 is linear, the closed form of x_2 is quadratic, and the closed form of x_3 is cubic:

$$x_3^{(n)} = -\tfrac{1}{6} \cdot n^3 + \tfrac{1-x_1}{2} \cdot n^2 + \left(\tfrac{x_1}{2} + x_2 - \tfrac{1}{3}\right) \cdot n + x_3$$

So when x_1, x_2, and x_3 have been fixed, $x_3^{(n)}$ has up to 2 extrema, i.e., its monotonicity may change twice. However, our techniques based on eventual monotonicity require that the respective expressions behave monotonically once they start to de- or increase, so these techniques only allow one change of monotonicity.

This raises the question if our approach can accelerate *every* loop with conjunctive guard and linear arithmetic whose closed form is a vector of (at most) quadratic polynomials with rational coefficients. We leave this to future work.

9.2 Non-Termination

To prove non-termination, our implementation applies the conditional non-termination techniques from Section 7 with the following priorities: Theorem 16 > Theorem 17 > Theorem 18. To evaluate our approach, we compared it with several leading tools for proving non-termination of integer programs: AProVE [29], iRankFinder [5], RevTerm [14], Ultimate [15], and VeryMax [44]. Note that AProVE uses, among other techniques, the tool T2 [13] as backend for proving non-termination, so we refrained from including T2 separately in our evaluation.

To compare with AProVE, RevTerm, and Ultimate, we transformed all examples into the format which is used in the category *Termination of C Integer Programs* of the *Termination and Complexity Competition*.[7] For iRankFinder and VeryMax, we transformed them into the format from the category *Termination of Integer Transition Systems* of the *Termination and Complexity Competition* [12]. The latter format is also supported by LoAT, so besides iRankFinder and VeryMax, we also used it to evaluate LoAT.

For the tools iRankFinder, Ultimate, and VeryMax, we used the versions of their last participations in the *Termination and Complexity Competition* (2019 for VeryMax and 2021 for iRankFinder and Ultimate), as suggested by the developers.

[5] While acceleration via metering functions may be exact in some cases (see the discussion after Theorem 4), our implementation cannot check whether this is the case.

[6] The other two are structurally similar, but more complex.

[7] http://termination-portal.org/wiki/C_Integer_Programs

For AProVE, the developers provided an up-to-date version. For RevTerm, we followed the build instruction from [31] and sequentially executed the following command lines, as suggested by the developers:

```
RevTerm.sh prog.c -linear part1 mathsat 2 1
RevTerm.sh prog.c -linear part2 mathsat 2 1
```

It is important to note that all tools but RevTerm and LoAT also try to prove termination. Thus, a comparison between the runtimes of LoAT and those other tools is of limited significance. Therefore, we also compared LoAT with configurations of the tools that only try to prove non-termination. For AProVE, such a configuration was kindly provided by its developers (named AProVE NT in Table 3). In the case of iRankFinder, the excellent documentation allowed us to easily build such a configuration ourselves (named iRankFinder NT in Table 3). In the case of Ultimate, its developers pointed out that a comparison w.r.t. runtime is meaningless, as it is dominated by Ultimate's startup-time of ∼10s on small examples. For VeryMax, it is not possible to disable termination-proving, according to its authors.

We again used a wall clock timeout of 60s and a memory limit of 128GB for each tool. For RevTerm, we used a timeout of 30s for the first invocation, and the remaining time for the second invocation.

The results can be seen in Table 3. They show that our novel calculus is competitive with state-of-the-art tools. Both iRankFinder and Ultimate can prove non-termination of precisely the same 205 examples. LoAT can prove non-termination of these examples, too. In addition, it solves one benchmark that cannot be handled by any other tool:[8]

$$\textbf{while } x > 9 \wedge x_1 \geq 0 \textbf{ do } \begin{pmatrix} x \\ x_1 \end{pmatrix} \leftarrow \begin{pmatrix} x_1^2 + 2 \cdot x_1 + 1 \\ x_1 + 1 \end{pmatrix}$$

Most likely, the other tools fail for this example due to the presence of non-linear arithmetic. Our calculus from Section 7 just needs to check implications, so as long as the underlying SMT-solver supports non-linearity, it can be applied to non-linear examples, too. It is worth mentioning that LoAT subsumes all other tools w.r.t. proving non-termination. There are only 4 examples where none of the tools can prove termination or non-termination. Termination of one of these examples can be proven by an experimental, unpublished module of LoAT, which is inspired by the calculi presented in this paper. A manual investigation revealed that the 3 remaining examples are terminating, too.

To investigate the impact of the different non-termination techniques, we also tested configurations where one of the non-termination techniques from Theorems 16 to 18 was disabled, respectively. The results can be seen in Table 4. First, note that disabling Theorem 16 does not reduce LoAT's power. The reason is that if the left-hand side p of an inequation $p > 0$ is monotonically increasing (such that Theorem 16 applies), then it is also eventually monotonically increasing (such that Theorem 17 applies). However, since Theorem 16 yields simpler certificates of non-termination than Theorem 17, LoAT still uses both techniques. Interestingly, ~~Ev-Inc~~ and ~~FP~~ are almost equally powerful: Without Theorem 17, LoAT still proves non-termination of 203 examples and without Theorem 18, LoAT solves 205 examples. Presumably, the reason is again due to examples like (31), where Theorem 17 finds the recurrent set $x_2 > 0$ and Theorem 18 finds the recurrent set $x_1 > 0 \wedge x_1 = x_2$. So even though both non-termination techniques are applicable in such cases, the recurrent set deduced via Theorem 17 is clearly more general and thus preferable in practice. Note that LoAT cannot solve a single example when *both* Theorem 17 and Theorem 18 are disabled (~~Ev-Inc, FP~~ in Table 4). Then the only remaining non-termination technique is *Non-Termination via Monotonic Increase*. Examples where this technique suffices to prove non-termination trivially diverge whenever the loop condition is satisfied, and hence they were filtered from our benchmark set (as explained at the beginning of Section 9).

Regarding the runtime, we think that LoAT is faster than the competing tools due to the fact that the technique presented in Section 7 requires very little search, whereas many other non-termination techniques are heavily search-based (e.g., due to the use of *templates*, as it is exercised by RevTerm). In our setting, the inequations that eventually constitute a certificate of non-termination immediately arise from the given loop. In this regard, iRankFinder's approach for proving non-termination is similar to ours, as it also requires little search.

[8] 1567523105272726.koat.smt2

	LoAT	AProVE	AProVE NT	iRankFinder	iRankFinder NT	RevTerm	Ultimate	VeryMax
NO	206	200	200	205	205	133	205	175
YES	0	1301	0	1298	0	0	919	1299
fail	1305	10	1311	8	1306	1378	387	37
avg rt	0.03s	16.09s	25.69s	1.40s	1.03s	38.85s	23.30s	3.17s
avg rt NO	0.03s	10.65s	9.67s	1.34s	0.96s	4.63s	7.99s	14.52s
median rt	0.02s	13.41s	15.74s	1.11s	0.91s	34.93s	11.57s	0.03
median rt NO	0.02s	8.51s	6.91s	1.17s	0.90s	1.88s	8.01s	5.36s
std dev rt	0.03s	10.09s	19.95s	2.25s	0.30s	16.61s	21.60s	10.76s
std dev rt NO	0.06s	5.80s	5.80s	0.92s	0.12s	11.08s	1.88s	13.24s

Table 3: Comparison of LoAT with competing tools

	LoAT	~~Inc~~	~~Ev-Inc~~	~~FP~~	~~Ev-Inc, FP~~
NO	206	206	203	205	0
fail	1305	1305	1308	1306	1511
avg rt	0.03s	0.03s	0.03s	0.03s	0.02s
avg rt NO	0.03s	0.02s	0.02s	0.02s	–
median rt	0.02s	0.02s	0.02s	0.02s	0.02s
median rt NO	0.02s	0.02s	0.02s	0.02s	–
std dev rt	0.03s	0.02s	0.02s	0.02s	0.02s
std dev rt NO	0.06s	0.03s	0.03s	0.04s	–

Table 4: Comparison of LoAT versions

LoAT:	Calculus from Section 7
AProVE:	https://aprove.informatik.rwth-aachen.de
AProVE NT:	Configuration of AProVE that does not try to prove termination
iRankFinder:	http://irankfinder.loopkiller.com
iRankFinder NT:	Configuration of iRankFinder that does not try to prove termination
RevTerm:	https://github.com/ekgma/RevTerm
Ultimate:	https://monteverdi.informatik.uni-freiburg.de/tomcat/Website
VeryMax:	https://www.cs.upc.edu/~albert/VeryMax.html
~~Inc~~:	Calculus from Section 7 without Theorem 16
~~Ev-Inc~~:	Calculus from Section 7 without Theorem 17
~~FP~~:	Calculus from Section 7 without Theorem 18
~~Ev-Inc, FP~~:	Calculus from Section 7 without Theorems 17 and 18
NO:	Number of non-termination proofs
YES:	Number of termination proofs
fail:	Number of examples where (non-)termination could not be proven
avg rt:	Average wall clock runtime
avg rt NO:	Average wall clock runtime when non-termination was proven
median rt:	Median wall clock runtime
median rt NO:	Median wall clock runtime when non-termination was proven
st dev rt:	Standard deviation of wall clock runtime
st dev rt NO:	Standard deviation of wall clock runtime when non-termination was proven

This is also reflected in our experiments, where iRankFinder is the second fastest tool.

It should also be taken into account that iRankFinder is implemented in Python, AProVE, RevTerm, and Ultimate are implemented in Java, and LoAT and VeryMax are implemented in C++. Thus, the difference in runtime is in parts due to the different performances of the respective programming language implementations.

Another interesting aspect of our evaluation is the result of RevTerm, which outperformed all other tools in the evaluation of [14]. The reason for this discrepancy is the following: In [14], 300 different configurations of RevTerm have been tested,

and a benchmark has been considered to be solved if at least one of these configurations was able to prove non-termination. In contrast, we ran two configurations of RevTerm, one for each of the two non-termination checks proposed in [14]. So essentially, RevTerm's results from [14] correspond to a highly parallel setting, whereas the results from our evaluation correspond to a sequential setting.

10 Conclusion and Future Work

We discussed existing acceleration techniques (Section 3) and presented a calculus to combine acceleration techniques modularly (Section 4). Then we showed how to combine existing (Section 5) and two novel (Section 6) acceleration techniques with our calculus. This improves over prior approaches, where acceleration techniques were used independently, and may thus improve acceleration-based verification techniques [8, 9, 21, 26, 28, 43] in the future. An empirical evaluation (Section 9.1) shows that our approach is more powerful than state-of-the-art acceleration techniques. Moreover, if it is able to accelerate a loop, then the result is exact (instead of just an under-approximation) in most cases. Thus, our calculus can be used for under-approximating techniques (e.g., to find bugs or counterexamples) as well as in over-approximating settings (e.g., to prove safety or termination).

Furthermore, we showed how our calculus from Section 4 can be adapted for proving non-termination in Section 7, where we also presented three non-termination techniques that can be combined with our novel calculus. While two of them (Theorems 16 and 18) are straightforward adaptions of existing non-termination techniques to our modular setting, the third one (Theorem 17) is, to the best of our knowledge, new and might also be of interest independently from our calculus.

Actually, the two calculi presented in this paper are so similar that they do not require separate implementations. In our tool LoAT, both of them are implemented together, such that we can handle loops uniformly: If our implementation of the calculi yields a certificate of non-termination, then it suffices to prove reachability of one of the corresponding witnesses of non-termination from an initial program state afterwards to finish the proof of non-termination. If our implementation of the calculi successfully accelerates the loop under consideration, this may help to prove reachability of other, potentially non-terminating configurations later on. If our implementation of the calculi fails, then LoAT continues its search with other program paths. The success of this strategy is demonstrated at the annual *Termination and Complexity Competition*, where LoAT has been the most powerful tool for proving non-termination of *Integer Transition Systems* since its first participation in 2020.

Regarding future work, we are actively working on support for disjunctive loop conditions. Moreover, our experiments indicate that integrating specialized techniques for FMATs (see Section 8) would improve the power of our approach for loop acceleration, as Flata exactly accelerated 49 loops where LoAT failed to do so (see Section 9). Furthermore, we plan to extend our approach to richer classes of loops, e.g., loops operating on both integers and arrays, non-deterministic loops, or loops operating on bitvectors (as opposed to mathematical integers).

Acknowledgments

We thank Marcel Hark, Sophie Tourret, and the anonymous reviewers for helpful feedback and comments. Moreover, we thank Jera Hensel for her help with AProVE, Radu Iosif and Filip Konečný for their help with Flata, Samir Genaim for his help with iRankFinder, Matthias Heizmann for his help with Ultimate, Ehsan Goharshady for his help with RevTerm, and Albert Rubio for his help with VeryMax.

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2), and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 389792660 as part of TRR 248 (see https://perspicuous-computing.science).

References

[1] Albert, E., Genaim, S., Martin-Martin, E., Merayo, A., Rubio, A.: Lower-bound synthesis using loop specialization and max-SMT. In: CAV '21. pp. 863–886. LNCS 12760 (2021). https://doi.org/10.1007/978-3-030-81688-9_40

[2] Bagnara, R., Pescetti, A., Zaccagnini, A., Zaffanella, E.: PURRS: Towards computer algebra support for fully automatic worst-case complexity analysis (2005), arXiv:cs/0512056 [cs.MS]

[3] Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Acceleration from theory to practice. Int. J. Softw. Tools Technol. Transf. **10**(5), 401–424 (2008). https://doi.org/10.1007/s10009-008-0064-3

[4] Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P.: Flat acceleration in symbolic model checking. In: ATVA '05. pp. 474–488. LNCS 3707 (2005). https://doi.org/10.1007/11562948_35

[5] Ben-Amram, A.M., Doménech, J.J., Genaim, S.: Multiphase-linear ranking functions and their relation to recurrent sets. In: SAS '19. pp. 459–480. LNCS 11822 (2019). https://doi.org/10.1007/978-3-030-32304-2_22

[6] Boigelot, B.: Symbolic Methods for Exploring Infinite State Spaces. Ph.D. thesis, Université de Liège (1999), https://orbi.uliege.be/bitstream/2268/74874/1/Boigelot98.pdf

[7] Boigelot, B.: On iterating linear transformations over recognizable sets of integers. Theor. Comput. Sci. **309**(1-3), 413–468 (2003). https://doi.org/10.1016/S0304-3975(03)00314-1

[8] Bozga, M., Gîrlea, C., Iosif, R.: Iterating octagons. In: TACAS '09. pp. 337–351. LNCS 5505 (2009). https://doi.org/10.1007/978-3-642-00768-2_29

[9] Bozga, M., Iosif, R., Konečný, F.: Fast acceleration of ultimately periodic relations. In: CAV '10. pp. 227–242. LNCS 6174 (2010). https://doi.org/10.1007/978-3-642-14295-6_23

[10] Bozga, M., Iosif, R., Konečný, F.: Deciding conditional termination. Log. Methods Comput. Sci. **10**(3) (2014). https://doi.org/10.2168/LMCS-10(3:8)2014

[11] Brockschmidt, M., Ströder, T., Otto, C., Giesl, J.: Automated detection of non-termination and `NullPointerExceptions` for Java Bytecode. In: FoVeOOS '11. pp. 123–141. LNCS 7421 (2012). https://doi.org/10.1007/978-3-642-31762-0_9

[12] Brockschmidt, M., Rybalchenko, A.: TermComp proposal: Pushdown systems as a model for programs with procedures (2014), https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/SMTPushdownPrograms.pdf

[13] Brockschmidt, M., Cook, B., Ishtiaq, S., Khlaaf, H., Piterman, N.: T2: temporal property verification. In: TACAS '16. pp. 387–393. LNCS 9636 (2016). https://doi.org/10.1007/978-3-662-49674-9_22

[14] Chatterjee, K., Goharshady, E.K., Novotný, P., Zikelic, D.: Proving non-termination by program reversal. In: PLDI '21. pp. 1033–1048 (2021). https://doi.org/10.1145/3453483.3454093

[15] Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M., Turrini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In: PLDI '18. pp. 135–150 (2018). https://doi.org/10.1145/3192366.3192405

[16] Chen, H., Cook, B., Fuhs, C., Nimkar, K., O'Hearn, P.W.: Proving nontermination via safety. In: TACAS '14. pp. 156–171. LNCS 8413 (2014). https://doi.org/10.1007/978-3-642-54862-8_11

[17] Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and Presburger arithmetic. In: CAV '98. pp. 268–279. LNCS 1427 (1998). https://doi.org/10.1007/BFb0028751

[18] Cook, B., Fuhs, C., Nimkar, K., O'Hearn, P.W.: Disproving termination with overapproximation. In: FMCAD '14. pp. 67–74 (2014). https://doi.org/10.1109/FMCAD.2014.6987597

[19] Dutertre, B.: Yices 2.2. In: CAV '14. pp. 737–744. LNCS 8559 (2014). https://doi.org/10.1007/978-3-319-08867-9_49

[20] Farzan, A., Kincaid, Z.: Compositional recurrence analysis. In: FMCAD '15. pp. 57–64 (2015). https://doi.org/10.1109/FMCAD.2015.7542253

[21] Frohn, F., Giesl, J.: Proving non-termination via loop acceleration. In: FMCAD '19. pp. 221–230 (2019). https://doi.org/10.23919/FMCAD.2019.8894271

[22] Frohn, F.: A calculus for modular loop acceleration. In: TACAS '20. pp. 58–76. LNCS 12078 (2020). https://doi.org/10.1007/978-3-030-45190-5_4

[23] Frohn, F., Fuhs, C.: Empirical evaluation of "A calculus for modular loop acceleration (and non-termination proofs)" (2022), https://ffrohn.github.io/acceleration-calculus

[24] Frohn, F., Giesl, J.: Termination of triangular integer loops is decidable. In: CAV '19. pp. 426–444. LNCS 11562 (2019). https://doi.org/10.1007/978-3-030-25543-5_24

[25] Frohn, F., Hark, M., Giesl, J.: Termination of polynomial loops. In: SAS '20. pp. 89–112. LNCS 12389 (2020). https://doi.org/10.1007/978-3-030-65474-0_5

[26] Frohn, F., Naaf, M., Brockschmidt, M., Giesl, J.: Inferring lower runtime bounds for integer programs. ACM Trans. Program. Lang. Syst. **42**(3), 13:1–13:50 (2020). https://doi.org/10.1145/3410331

[27] Frohn, F., Naaf, M., Hensel, J., Brockschmidt, M., Giesl, J.: Lower runtime bounds for integer programs. In: IJCAR '16. pp. 550–567. LNCS 9706 (2016). https://doi.org/10.1007/978-3-319-40229-1_37

[28] Ganty, P., Iosif, R., Konečný, F.: Under-approximation of procedure summaries for integer programs. Int. J. Softw. Tools Technol. Transf. **19**(5), 565–584 (2017). https://doi.org/10.1007/s10009-016-0420-7

[29] Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. J. Autom. Reasoning **58**(1), 3–31 (2017). https://doi.org/10.1007/s10817-016-9388-y

[30] Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: TACAS '19. pp. 156–166. LNCS 11429 (2019). https://doi.org/10.1007/978-3-030-17502-3_10

[31] Goharshady, E.K.: RevTerm on GitHub (2021), https://github.com/ekgma/RevTerm

[32] Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: SAS '06. pp. 144–160. LNCS 4134 (2006). https://doi.org/10.1007/11823230_10

[33] Gonnord, L., Schrammel, P.: Abstract acceleration in linear relation analysis. Sci. Comput. Program. **93**, 125–153 (2014). https://doi.org/10.1016/j.scico.2013.09.016

[34] Gulwani, S., Srivastava, S., Venkatesan, R.: Program analysis as constraint solving. In: PLDI '08. pp. 281–292 (2008). https://doi.org/10.1145/1375581.1375616

[35] Gupta, A., Henzinger, T.A., Majumdar, R., Rybalchenko, A., Xu, R.: Proving non-termination. In: POPL '08. pp. 147–158 (2008). https://doi.org/10.1145/1328438.1328459

[36] Hark, M., Frohn, F., Giesl, J.: Polynomial loops: Beyond termination. In: LPAR '20. pp. 279–297. EPiC Series in Computing 73 (2020). https://doi.org/10.29007/nxv1

[37] Hojjat, H., Iosif, R., Konečný, F., Kuncak, V., Rümmer, P.: Accelerating interpolants. In: ATVA '12. pp. 187–202. LNCS 7561 (2012). https://doi.org/10.1007/978-3-642-33386-6_16

[38] Hojjat, H., Konečný, F., Garnier, F., Iosif, R., Kuncak, V., Rümmer, P.: A verification toolkit for numerical transition systems - tool paper. In: FM '12. pp. 247–251. LNCS 7436 (2012). https://doi.org/10.1007/

978-3-642-32759-9_21

[39] Hosseini, M., Ouaknine, J., Worrell, J.: Termination of linear loops over the integers. In: ICALP '19. pp. 118:1–118:13. LIPIcs 132 (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.118

[40] Jeannet, B., Schrammel, P., Sankaranarayanan, S.: Abstract acceleration of general linear loops. In: POPL '14. pp. 529–540 (2014). https://doi.org/10.1145/2535838.2535843

[41] Kincaid, Z., Breck, J., Boroujeni, A.F., Reps, T.W.: Compositional recurrence analysis revisited. In: PLDI '17. pp. 248–262 (2017). https://doi.org/10.1145/3062341.3062373

[42] Konečný, F.: PTIME computation of transitive closures of octagonal relations. In: TACAS '16. pp. 645–661. LNCS 9636 (2016). https://doi.org/10.1007/978-3-662-49674-9_42

[43] Kroening, D., Lewis, M., Weissenbacher, G.: Under-approximating loops in C programs for fast counterexample detection. Formal Methods Syst. Des. **47**(1), 75–92 (2015). https://doi.org/10.1007/s10703-015-0228-1

[44] Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using max-SMT. In: CAV '14. pp. 779–796. LNCS 8559 (2014). https://doi.org/10.1007/978-3-319-08867-9_52

[45] Leike, J., Heizmann, M.: Geometric nontermination arguments. In: TACAS '18. pp. 266–283. LNCS 10806 (2018). https://doi.org/10.1007/978-3-319-89963-3_16

[46] Madhukar, K., Wachter, B., Kroening, D., Lewis, M., Srivas, M.K.: Accelerating invariant generation. In: FMCAD '15. pp. 105–111 (2015). https://doi.org/10.1109/FMCAD.2015.7542259

[47] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS '08. pp. 337–340. LNCS 4963 (2008). https://doi.org/10.1007/978-3-540-78800-3_24

[48] Ouaknine, J., Pinto, J.S., Worrell, J.: On termination of integer linear loops. In: SODA '15. pp. 957–969 (2015). https://doi.org/10.1137/1.9781611973730.65

[49] Silverman, J., Kincaid, Z.: Loop summarization with rational vector addition systems. In: CAV '19. pp. 97–115. LNCS 11562 (2019). https://doi.org/10.1007/978-3-030-25543-5_7

[50] StarExec hardware specifications (2022), https://www.starexec.org/starexec/public/machine-specs.txt

[51] Strejcek, J., Trtík, M.: Abstracting path conditions. In: ISSTA '12. pp. 155–165 (2012). https://doi.org/10.1145/2338965.2336772

[52] Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: IJCAR '14. pp. 367–373. LNCS 8562 (2014). https://doi.org/10.1007/978-3-319-08587-6_28

[53] Termination problems data base (TPDB), http://termination-portal.org/wiki/TPDB

[54] Urban, C., Gurfinkel, A., Kahsai, T.: Synthesizing ranking functions from bits and pieces. In: TACAS '16. pp. 54–70. LNCS 9636 (2016). https://doi.org/10.1007/978-3-662-49674-9_4

[55] Velroyen, H., Rümmer, P.: Non-termination checking for imperative programs. In: TAP '08. pp. 154–170. LNCS 4966 (2008). https://doi.org/10.1007/978-3-540-79124-9_11