Frohn, F. and Fuhs, Carsten (2022) A calculus for modular loop acceleration and non-termination proofs. International Journal on Software Tools for Technology Transfer 24, pp. 691-715. ISSN 1433-2779.

Downloaded from:

Usage Guidelines:
Please refer to usage guidelines at
contact lib-eprints@bbk.ac.uk.
A calculus for modular loop acceleration and non-termination proofs

Florian Frohn 1,2 · Carsten Fuhs 3

Accepted: 29 August 2022 / Published online: 7 October 2022 © The Author(s) 2022

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 have 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.

Keywords Loop acceleration · Non-termination · Integer transition systems · Program verification · Static analysis

1 Introduction
In the last years, loop acceleration techniques have successfully been used to build static analyses for programs operating on integers [3,10,22,27,29,44]. Essentially, such techniques extract a quantifier-free first-order formula $\psi$ from a single-path loop $T$, i.e., a loop without branching in its body, such that $\psi$ under-approximates (or is equivalent to) $T$. More specifically, each model of the resulting formula $\psi$ corresponds to an execution of $T$ (and vice versa). By integrating such techniques into a suitable program analysis framework [4,22,27,29,38], whole programs can be transformed into first-order formulas which can then be analyzed by off-the-shelf solvers. Applications include proving safety [38] or reachability [38,44], deducing bounds on the runtime complexity [27], and proving (non-)termination [10,22].

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 [22], 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 [22], 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 Sect. 2. Then, we discuss existing acceleration techniques in Sect. 3. In Sect. 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 \( \text{conditional} \) 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 Sect. 6. Then, we adapt our calculus and certain acceleration techniques for proving non-termination in Sect. 7. After discussing related work in Sect. 8, we demonstrate the applicability of our approach via an empirical evaluation in Sect. 9 and conclude in Sect. 10.

A conference version of this paper was published in [23]. The present paper provides the following additional contributions:

- We present formal correctness proofs for all of our results, which were omitted in [23] for reasons of space.
- We present an improved version of the loop acceleration technique from [27, Thm. 3.8] and [28, Thm. 7] that yields simpler formulas.
- We prove an informal statement from [23] 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 (Sect. 7) is completely new.
- We extend the empirical evaluation from [23] with extensive experiments comparing the adaptation of our calculus for proving non-termination with other state-of-the-art tools for proving non-termination (Sect. 9.2).

2 Preliminaries

We use the notation \( \vec{x}, \vec{y}, \vec{z}, \ldots \) for vectors. Let \( \mathcal{C}(\vec{z}) \) be the set of \( \text{closed-form expressions} \) over the variables \( \vec{z} \). So \( \mathcal{C}(\vec{z}) \) 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 \( \mathcal{C}(\vec{z}) \) (which may use other function symbols). Thus, we leave its definition open to avoid restricting our results unnecessarily. We consider loops of the form

\[
\text{while } \varphi \text{ do } \vec{x} \leftarrow \vec{a} \quad (T_{\text{loop}})
\]

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

We identify \( T_{\text{loop}} \) 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 \( \varphi \)) with the predicate \( \vec{x} \mapsto \varphi \). We use the standard integer–arithmetic semantics for the symbols occurring in formulas.

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

\[
\vec{a} := \left( \frac{a_1}{a_d} \right), \quad \vec{x} := \left( \frac{x_1}{x_d} \right), \quad \vec{x}' := \left( \frac{x'_1}{x'_d} \right), \quad \vec{y} := \left( \frac{y}{n} \right).
\]

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.

\( T_{\text{loop}} \) induces a relation \( \rightarrow_{T_{\text{loop}}} \) on \( \mathbb{Z}^d \):

\[
\varphi(\vec{x}) \land \vec{x}' = \vec{a}(\vec{x}) \iff \vec{x} \rightarrow_{T_{\text{loop}}} \vec{x}'
\]

3 Existing acceleration techniques

In the following (up to and including Sect. 6), our goal is to \( \text{accelerate} \) \( T_{\text{loop}} \), i.e., to find a formula \( \psi \in \text{FOQF}(\mathcal{C}(\vec{y})) \) such that

\[
\psi \iff \vec{x} \rightarrow_{T_{\text{loop}}}^n \vec{x}' \quad \text{for all } n > 0. \quad \text{(equiv)}
\]

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

\[
\text{while } x_1 > 0 \text{ do } \left( \frac{x_1}{x_2} \right) \leftarrow \left( \frac{x_1 - 1}{2 \cdot x_2} \right). \quad (T_{\text{exp}})
\]

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

\[
\left( \frac{x_1}{x_2} \right) = \left( \frac{x_1 - n}{2 \cdot x_2} \right) \land x_1 - n + 1 > 0, \quad (\psi_{\text{exp}})
\]

\( ^1 \) i.e., the (anonymous) function that maps \( \vec{x} \) to \( \vec{a} \).
where \( \binom{x_1 - n}{2^i x_2} \) is equivalent to the value of \( \binom{x_1}{1} \) after \( n \) iterations, and the inequation \( x_1 - n + 1 > 0 \) ensures that \( T_{exp} \) 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 \( T_{loop} \), i.e., we have

\[
\psi \implies \vec{x} \xrightarrow{n} T_{loop} \vec{x}' \quad \text{for all } n > 0. 
\]

If (equiv) holds, then \( \psi \) is equivalent to \( T_{loop} \). Similarly, if (approx) holds, then \( \psi \) approximates \( T_{loop} \).

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

\[
\text{accel} : \text{Loop} \rightarrow \text{FO}_{OF}(\mathcal{C}(\vec{y})).
\]

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

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

**Definition 2 (Closed Form)** We call \( \vec{c} \in \mathcal{C}(\vec{x}, n)^d \) a closed form of \( T_{loop} \) 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 \( B := \left( \begin{array}{cc} A & \vec{b} \\ \vec{0}^T & 1 \end{array} \right) \) 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 \( \vec{a}^n \) can be given directly (see, e.g., [48]), and \( \vec{a}^n(\vec{x}) \) is equal to the first \( d \) components of \( T^{-1}J^nT \left( \frac{1}{\vec{1}} \right) \). Moreover, one can compute a closed form if \( \vec{a} = \left( \frac{c_1 x_1 + p_1}{c_d x_d + p_d} \right) \) where \( c_i \in \mathbb{N} \) and each \( p_i \) is a polynomial over \( x_1, \ldots, x_{i-1} \) [26,37].

### 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 \( \varphi \) of \( T_{loop} \) is satisfied throughout (at least) \( n \) loop iterations. So in other words, it requires that the indicator function (or characteristic function) \( I_\varphi : \mathbb{Z}^d \rightarrow \{0, 1\} \) of \( \varphi \) with \( I_\varphi(\vec{x}) = 1 \iff \varphi(\vec{x}) \) is monotonically decreasing w.r.t. \( \vec{a} \), i.e., \( I_\varphi(\vec{a}(\vec{x})) \geq I_\varphi(\vec{x}) \).

**Theorem 1** (Acceleration via Monotonic Decrease [44]) If

\[
\varphi(\vec{a}(\vec{x})) \implies \varphi(\vec{x}),
\]

then the following acceleration technique is exact:

\[
T_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \land \varphi(\vec{a}^{n-1}(\vec{x}))
\]

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

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

\[
\text{while } x_1 > 0 \land x_2 > 0 \text{ do } (\frac{x_1}{x_2}) \leftarrow (\frac{x_1 - 1}{x_2 + 1}). \quad (T_{non-dec})
\]

It cannot be accelerated with Theorem 1 as

\[
x_1 - 1 > 0 \land x_2 + 1 > 0 \nrightarrow x_1 > 0 \land x_2 > 0.
\]

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 \( \varphi \) is a loop invariant of \( T_{loop} \). Then \( \{\vec{x} \in \mathbb{Z}^d | \varphi(\vec{x})\} \) is a recurrent set [36] (see also Sect. 8.2 of \( T_{loop} \)). In other words, this acceleration technique applies if \( I_\varphi \) 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:

\[
T_{loop} \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \land \varphi(\vec{x})
\]

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

\[3\] We call a formula \( \delta \) a loop invariant of a loop \( T_{loop} \) if \( \varphi(\vec{x}) \land \delta(\vec{x}) \implies \delta(\vec{a}(\vec{x})) \) is valid.
We will prove the more general Theorem 8 in Sect. 5.

**Example 1** As a minimal example, Theorem 2 accelerates

while \( x > 0 \) do \( x \leftarrow x + 1 \)

\((T_{\text{inc}})\)
to \( x' = x + n \land x > 0 \).

### 3.2 Acceleration via decrease and increase

Both acceleration techniques presented so far have been generalized in [22].

**Theorem 3** (Acceleration via Monotonicity [22]) If

\[
\begin{align*}
\varphi_1(\bar{x}) &\iff \varphi_1(\bar{x}) \land \varphi_2(\bar{x}) \land \varphi_3(\bar{x}), \\
\varphi_1(\bar{x}) &\implies \varphi_1(\bar{a}(\bar{x})), \\
\varphi_1(\bar{x}) \land \varphi_2(\bar{a}(\bar{x})) &\implies \varphi_2(\bar{x}), \\
\varphi_1(\bar{x}) \land \varphi_2(\bar{x}) \land \varphi_3(\bar{x}) &\implies \varphi_3(\bar{a}(\bar{x})),
\end{align*}
\]

then the following acceleration technique is exact:

\( T_{\text{loop}} \mapsto \bar{x}' = \bar{a}^n(\bar{x}) \land \varphi_1(\bar{x}) \land \varphi_2(\bar{a}^{n-1}(\bar{x})) \land \varphi_3(\bar{x}) \)

**Proof** Immediate consequence of Theorem 5 and Remark 1, which are proved in Sects. 4 and 5.

Here, \( \varphi_1 \) and \( \varphi_3 \) 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, \( \varphi_2 \) needs to satisfy a similar condition as in Theorem 1, and thus, it suffices to require that \( \varphi_2 \) holds before the last iteration. In such cases, i.e., if

\[
\begin{align*}
\varphi_1(\bar{x}) \land \varphi_2(\bar{a}(\bar{x})) &\implies \varphi_2(\bar{x})
\end{align*}
\]
is valid, we also say that \( \varphi_2 \) is a converse invariant (w.r.t. \( \varphi_1 \)). 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, \( T_{\text{non-dec}} \) can be accelerated to

\[
\begin{align*}
\left( \frac{x_1}{x_2} \right) &= \left( \frac{x_1 - n}{x_2 + n} \right) \land x_2 > 0 \land x_1 - n + 1 > 0 \quad (\text{psi}_{\text{non-dec}})
\end{align*}
\]

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

while \( x_1 > 0 \land x_2 > 0 \) do \( \left( \frac{x_1}{x_2} \right) \leftarrow \left( \frac{x_1 + x_2}{x_2 - 1} \right) \) \((T_{2-\text{inv})})

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 \( T_{2-\text{inv}} \):

while \( x_1 > 0 \land x_2 > 0 \) do \( \left( \frac{x_1}{x_2} \right) \leftarrow \left( \frac{x_1 - x_2}{x_2 + 1} \right) \)

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 [28]. 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 \( \bar{x} \in \mathbb{Z}^d \), the loop under consideration can be applied at least \( \lceil mf(\bar{x}) \rceil \) times.

**Definition 3** (Metering Function [28]) We call a function \( mf : \mathbb{Z}^d \to \mathbb{Q} \) a metering function if the following holds:

\[
\begin{align*}
\varphi(\bar{x}) &\implies mf(\bar{x}) - mf(\bar{a}(\bar{x})) \leq 1 \\
\neg \varphi(\bar{x}) &\implies mf(\bar{x}) \leq 0
\end{align*}
\]

We can use metering functions to accelerate loops as follows:

**Theorem 4** (Acceleration via Metering Functions [27,28])

Let \( mf \) be a metering function for \( T_{\text{loop}} \). Then, the following acceleration technique is sound:

\( T_{\text{loop}} \mapsto \bar{x}' = \bar{a}^n(\bar{x}) \land n < mf(\bar{x}) + 1 \)

We will prove the more general Theorem 9 in Sect. 5. In contrast to [27, Thm. 3.8] and [28, Thm. 7], the acceleration technique from Theorem 4 does not conjoin the loop condition \( \varphi \) to the result, which turned out to be superfluous. The reason is that \( 0 < n < mf(\bar{x}) + 1 \) implies \( \varphi \) due to (mf-bounded).

**Example 3** Using the metering function \( \left( x_1, x_2 \right) \mapsto x_1 \), Theorem 4 accelerates \( T_{\text{exp}} \) to

\[
\begin{align*}
\left( \frac{x_1}{x_2} \right) &= \left( \frac{x_1 - n}{2^n x_2} \right) \land n < x_1 + 1 \quad \equiv \psi_{\text{exp}}
\end{align*}
\]

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

Linear metering functions can be synthesized via Farkas’ Lemma and SMT solving [28]. However, many loops have
only trivial linear metering functions. To see this, reconsider
\( T_{\text{non-dec}} \). Here, \((x_1, x_2) \mapsto x_1\) is not a metering function as
\( T_{\text{non-dec}} \) cannot be iterated at least \( x_1 \) times if \( x_2 \leq 0 \). Thus,
[27] proposes a refinement of [28] based on metering functions
of the form \( x \mapsto I_\xi(I_\xi(x)) \cdot f(x) \) where \( \xi \in FOQF(\mathcal{E}(x)) \)
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 \( T_{\text{non-dec}} \) to
\[
\left( \frac{x_1}{x_2} \right) = \left( \frac{x_1 - n}{x_2 + n} \right) \wedge x_2 > 0 \wedge n < x_1 + 1.
\]

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
\( T_{\text{loop}} \) until it is solved and hence gives rise to a suitable
\( \psi \in FOQF(\mathcal{E}(\bar{y})) \) which approximates or is equivalent to
\( T_{\text{loop}} \).

Definition 4 (Acceleration Problem) A tuple
\[
[\psi \mid \hat{\psi} \mid \hat{\phi}]_a
\]
where \( \psi \in FOQF(\mathcal{E}(\bar{y})), \hat{\psi}, \hat{\phi} \in FOQF(\mathcal{E}(x)), \) and \( a : \mathbb{Z}^d \rightarrow \mathbb{Z}^d \) is an acceleration problem. It is consistent if \( \psi \)
approximates \( (\hat{\psi}, \hat{a}) \), exact if \( \psi \) is equivalent to \( (\hat{\psi}, \hat{a}) \), and
solved if it is consistent and \( \hat{\phi} \equiv T \). The canonical acceleration problem
of a loop \( T_{\text{loop}} \) is
\[
[\hat{x}' = a^n(\bar{x}) \mid T \mid \phi(\bar{x})]_a.
\]

Example 4 The canonical acceleration problem of \( T_{\text{non-dec}} \) is
\[
\left[\left( \frac{x_1}{x_2} \right) = \left( \frac{x_1 - n}{x_2 + n} \right) \wedge x_2 > 0 \right]_{x_1 > 0}.
\]

The first component \( \psi \) of an acceleration problem
\( [\psi \mid \hat{\psi} \mid \hat{\phi}]_a \) is the partial result that has been computed so far.
The second component \( \hat{\psi} \) corresponds to the part of the loop condition that has already been processed successfully.
As our calculus preserves consistency, \( \psi \) always approximates \( (\hat{\psi}, \hat{a}) \).
The third component is the part of the loop condition that remains to be processed, i.e., the loop \( (\hat{\psi}, \hat{a}) \)
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
\[
[\hat{x}' = a^n(\bar{x}) \mid T \mid \phi(\bar{x})]_a
\]
to
\[
[\psi(\bar{x}) \mid \hat{\phi}(\bar{x}) \mid \hat{\phi}(\bar{x})]_a,
\]
then \( \psi \equiv \hat{\phi} \wedge \hat{\phi} \) and
\[
\psi_1 \text{ implies } \bar{x} \rightarrow^n_{(\hat{\psi}, \hat{a})} \bar{x}'.
\]

Then it suffices to find some \( \psi_2 \in FOQF(\mathcal{E}(\bar{y})) \) such that
\[
\bar{x} \rightarrow^n_{(\hat{\psi}, \hat{a})} \bar{x}' \text{ implies } \bar{x} \rightarrow^n_{(\hat{\phi}, \hat{a})} \bar{x}'. \quad (1)
\]
The reason is that we have \( \rightarrow (\hat{\psi}, \hat{a}) \cap \rightarrow (\hat{\phi}, \hat{a}) = \rightarrow (\hat{\phi}, \hat{a}) \)
and thus
\[
\psi_1 \wedge \psi_2 \text{ implies } \bar{x} \rightarrow^n_{(\hat{\phi}, \hat{a})} \bar{x}'.
\]
i.e., \( \psi_1 \wedge \psi_2 \) approximates \( T_{\text{loop}} \).

Note that the acceleration techniques presented so far
would map \( (\hat{\psi}, \hat{a}) \) to some \( \psi_2 \in FOQF(\mathcal{E}(\bar{y})) \) such that
\[
\psi_2 \text{ implies } \bar{x} \rightarrow^n_{(\hat{\phi}, \hat{a})} \bar{x}', \quad (2)
\]
which does not use the information that we have already accelerated \( (\hat{\phi}, \hat{a}) \). In Sect. 5, we will adapt all acceleration
techniques from Sect. 3 to search for some \( \psi_2 \in FOQF(\mathcal{E}(\bar{y})) \) 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
\[
accel : \text{Loop} \times FOQF(\mathcal{E}(\bar{y})) \rightarrow FOQF(\mathcal{E}(\bar{y}))
\]
a conditional acceleration technique. It is sound if
\[
\bar{x} \rightarrow^n_{(\hat{\chi}, \hat{a})} \bar{x}' \text{ and } accel((\hat{\chi}, \hat{a}), \hat{\phi}) \text{ implies } \bar{x} \rightarrow^n_{(\hat{\chi}, \hat{a})} \bar{x}'
\]
for all \((\hat{\chi}, \hat{a}), \hat{\phi}) \in \text{dom}(accel), \bar{x}, \bar{x}' \in \mathbb{Z}^d, \) and \( n > 0 \). It is
exact if additionally
\[
\bar{x} \rightarrow^n_{(\hat{\chi} \wedge \hat{\phi}, \hat{a})} \bar{x}' \text{ implies } accel((\hat{\chi}, \hat{a}), \hat{\phi})
\]
for all \((\hat{\chi}, \hat{a}), \hat{\phi}) \in \text{dom}(accel), \bar{x}, \bar{x}' \in \mathbb{Z}^d, \) and \( n > 0 \).

Note that every acceleration technique gives rise to a con-
ditional acceleration technique in a straightforward way (by
disregarding the second argument \( \hat{\phi} \) of \( accel \) in Definition 5).
Soundness and exactness can be lifted directly to the con-
ditional setting.
Lemma 1 (Acceleration as Conditional Acceleration) Let \( \text{accel}_0 \) be an acceleration technique following Definition 1 such that \( \text{accel}_0((\chi, \vec{a})) \implies \vec{x}' = \vec{a}^n(\vec{x}) \) is valid whenever \( (\chi, \vec{a}) \in \text{dom(accel)} \). Then for the conditional acceleration technique \( \text{accel} \) given by \( \text{accel}(\mathcal{T}, \hat{\varphi}) := \text{accel}_0(\mathcal{T}) \), the following holds:

1. \( \text{accel} \) is sound if and only if \( \text{accel}_0 \) is sound
2. \( \text{accel} \) is exact if and only if \( \text{accel}_0 \) is exact

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

\[
\vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}' \land \text{accel}((\chi, \vec{a}), \hat{\varphi}) \implies \vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}'
\]

if \( \text{accel}_0 \) is a sound acceleration technique. Thus:

\[
\vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}' \land \text{accel}((\chi, \vec{a}), \hat{\varphi}) \implies \text{accel}((\chi, \vec{a}), \hat{\varphi})
\]

\[
\iff \text{accel}_0((\chi, \vec{a})) \quad \text{(by definition of accel)}
\]

\[
\iff \vec{x} \rightarrow n_{(\chi, \vec{a})} \vec{x}' \quad \text{(by soundness of accel)}
\]

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

\[
\text{accel}_0((\varphi, \vec{a})) \implies \vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}'
\]

if \( \text{accel} \) is a sound conditional acceleration technique. Thus:

\[
\text{accel}_0((\varphi, \vec{a}))
\]

\[
\iff \text{accel}_0((\varphi, \vec{a})) \land \vec{x}' = \vec{a}^n(\vec{x})
\]

(by the prerequisites of the lemma)

\[
\iff \vec{x} \rightarrow n_{(\mathcal{T}, \vec{a})} \vec{x}' \land \text{accel}_0((\varphi, \vec{a}))
\]

\[
\iff \vec{x} \rightarrow n_{(\mathcal{T}, \vec{a})} \vec{x}' \land \text{accel}((\varphi, \vec{a}), \mathcal{T})
\]

(by definition of accel)

\[
\iff \vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}' \quad \text{(by soundness of accel)}
\]

For the “if” direction of 2., soundness of \( \text{accel} \) follows from 1. We still need to show that

\[
\vec{x} \rightarrow n_{(\chi, \varphi, \vec{a})} \vec{x}' \implies \text{accel}((\chi, \vec{a}), \hat{\varphi})
\]

if \( \text{accel}_0 \) is an exact acceleration technique. Thus:

\[
\vec{x} \rightarrow n_{(\chi, \varphi, \vec{a})} \vec{x}'
\]

\[
\iff \vec{x} \rightarrow n_{(\chi, \vec{a})} \vec{x}'
\]

\[
\iff \text{accel}_0((\chi, \vec{a})) \quad \text{(by exactness of accel)}
\]

\[
\iff \text{accel}((\chi, \vec{a}), \hat{\varphi}) \quad \text{(by definition of accel)}
\]

For the “only if” direction of 2., soundness of \( \text{accel}_0 \) follows from 1. We still need to show that

\[
\vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}' \implies \text{accel}_0((\varphi, \vec{a}))
\]

if \( \text{accel} \) is an exact conditional acceleration technique. Thus:

\[
\vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}'
\]

\[
\iff \text{accel}_0((\varphi, \vec{a})) \quad \text{(by exactness of accel)}
\]

\[
\iff \text{accel}((\varphi, \vec{a})) \quad \text{(by definition of accel)}
\]

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 \( \land_{i=1}^{k} C_i \) with the set of clauses \( \{C_i \mid 1 \leq i \leq k\} \).

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

\[
\emptyset \neq \chi \subseteq \varphi \text{ accel}((\chi, \vec{a}), \hat{\varphi}) = \psi_2 \quad \overline{\psi_1 | \hat{\varphi} | \hat{\varphi}_\varepsilon \rightarrow (e) \ \psi_1 \cup \psi_2 \cup \chi \ \hat{\varphi} \ \hat{\varphi}_\varepsilon \rightarrow}
\]

where \( \text{accel} \) is a sound conditional acceleration technique.

A \( \rightarrow \)-step is exact (written \( \rightarrow_e \)) if \( \text{accel} \) is exact.

So our calculus allows us to pick a subset \( \chi \) (of clauses) from the yet unprocessed condition \( \hat{\varphi} \) and “move” it to \( \hat{\varphi} \), which has already been processed successfully. To this end, \( (\chi, \vec{a}) \) needs to be accelerated by a conditional acceleration technique, i.e., when accelerating \( (\chi, \vec{a}) \) we may assume

\[
\vec{x} \rightarrow n_{(\varphi, \vec{a})} \vec{x}'.
\]

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 Sect. 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 \( \{x_1 > 0, \left(\frac{x_1 + 1}{x_2 + 1}\right)\} \) to enable a first \( \rightarrow \)-step, and we need to accelerate \( \{x_2 > 0, \left(\frac{x_1 + 1}{x_2 + 1}\right)\} \) afterward. The resulting derivation is shown in Fig. 1. Thus, we successfully constructed the formula \( \psi_{\text{non-dec}} \), which is equivalent to \( T_{\text{non-dec}} \). Note that here neither of the two steps benefit from the component \( \hat{\varphi} \) of the acceleration problems. We will introduce more powerful conditional acceleration techniques that benefit from \( \hat{\varphi} \) in Sect. 5.

The crucial property of our calculus is the following.

Lemma 2 The relation \( \rightarrow \) preserves consistency, and the relation \( \rightarrow_e \) preserves exactness.
Fig. 1  \( \leadsto \)-derivation for \( \mathcal{T}_{\text{non-dec}} \)

\[
\begin{align*}
\llbracket \psi_{\text{non-dec}}^{\text{init}} \rrbracket := (x_1^n) & = (x_1 - n) + 1) | x_1 > 0 \land x_2 > 0 \end{align*}
\]

\( \leadsto e \llbracket \psi_{\text{non-dec}}^{\text{init}} \rrbracket \land x_1 - n > 0 | x_1 > 0 | x_2 > 0 \llbracket (x_1^n) + 1) \) (Theorem 1)

\( \leadsto e \llbracket \psi_{\text{non-dec}}^{\text{init}} \rrbracket \land x_1 - n > 0 \land x_2 > 0 | x_1 > 0 \land x_2 > 0 \rrbracket \llbracket (x_1^n) + 1) \) (Theorem 2)

\[\llbracket \psi_{\text{non-dec}} \rrbracket | x_1 > 0 \land x_2 > 0 \rrbracket \llbracket (x_1^n) + 1) \]

**Proof** For the first part of the lemma, assume

\[\llbracket \psi_1 | \phi \mid \phi \rrbracket \leadsto \llbracket \psi_1 \cup \psi_2 | \phi \cup \chi \mid \phi \rrbracket \]

where \( \llbracket \psi_1 | \phi \mid \phi \rrbracket \) is consistent and

\[\operatorname{accel}(\chi, \alpha), \phi) = \psi_2.\]

We get

\[\begin{align*}
\psi_1 & \land \psi_2 \\
\implies & \exists n (x_1) \land \psi_2 \\
\implies & \exists n (\phi, x_1) \land \psi_2 \\
\implies & \exists n (\phi, x_1) \land \psi_2
\end{align*}\]

The first step holds since \( \llbracket \psi_1 | \phi \mid \phi \rrbracket \) is consistent and the second step holds since \( \operatorname{accel} \) is sound. This proves consistency of

\[\llbracket \psi_1 \land \psi_2 | \phi \cup \chi \mid \phi \rrbracket \]

\[\llbracket \psi_1 \cup \psi_2 | \phi \cup \chi \mid \phi \rrbracket \]

For the second part of the lemma, assume

\[\llbracket \psi_1 | \phi \mid \phi \rrbracket \leadsto e \llbracket \psi_1 \cup \psi_2 | \phi \cup \chi \mid \phi \rrbracket \]

where \( \llbracket \psi_1 | \phi \mid \phi \rrbracket \) is exact and \( \operatorname{accel}(\chi, \alpha), \phi) \rightarrow \psi_2. \)

We get

\[\begin{align*}
\exists n (\phi, x_1) \land \psi_2 \\
\implies & \exists n (\phi, x_1) \land \psi_2 \\
\implies & \exists n (\phi, x_1) \land \psi_2
\end{align*}\]

which proves exactness of

\[\llbracket \psi_1 \cup \psi_2 | \phi \cup \chi \mid \phi \rrbracket \]

Then, the correctness of our calculus follows immediately.

The reason is that

\[\llbracket \phi \rightarrow \phi \rrbracket \]

implies \( \phi \equiv \phi \).

**Theorem 5** (Correctness of \( \leadsto \)) If

\[\llbracket \phi \rightarrow \phi \rrbracket \]

then \( \phi \) approximates \( \mathcal{T}_\text{loop} \). If

\[\llbracket \phi \rightarrow \phi \rrbracket \]

then \( \phi \) is equivalent to \( \mathcal{T}_\text{loop} \).

Termination of our calculus is trivial, as the size of the third component \( \phi \) of the acceleration problem is decreasing.

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

### 5 Conditional acceleration techniques

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

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

\[\psi(x) \land \chi (\alpha (x)) \implies \chi (x),\]

then the following conditional acceleration technique is exact:

\[(\chi, \alpha, \psi) \mapsto \bar{x}' = \bar{a}^n (\bar{x}) \land \chi (\bar{a}^{n-1} (\bar{x}))\]

**Proof** For soundness, we need to prove

\[\bar{x} \rightarrow_{(\bar{x}, \bar{a})} \bar{a}^m (\bar{x}) \land \chi (\bar{a}^{m-1} (\bar{x}))\]

\[\bar{x} \rightarrow_{(\bar{x}, \bar{a})} \bar{a}^m (\bar{x})\]

\[\square\]

\[\square\]
Fig. 2 \(\rightsquigarrow\) derivation for \(T_{\text{2-inv}}\)

\[
[\vec{x}^n = \vec{a}_{\Delta}^{n} \vec{x}^n | \top | x_1 > 0 \land x_2 > 0 | x_2 > 0 | x_2 > 0 | x_1 > 0 | \top]_{\vec{a}_{\Delta}^{n}}
\]

\(\rightsquigarrow\) [\(\vec{x}^n = \vec{a}_{\Delta}^{n} \vec{x}^n | x_2 - n + 1 > 0 | x_2 > 0 | x_2 > 0 | x_1 > 0 | \top]_{\vec{a}_{\Delta}^{n}}\)

\(\rightsquigarrow\) \(\vec{x}^n = \vec{a}_{\Delta}^{n} \vec{x}^n | x_2 - n + 1 > 0 \land x_1 > 0 | x_2 > 0 \land x_1 > 0 | \top]_{\vec{a}_{\Delta}^{n}}\)

(Theorem 7)

(Theorem 8)

for all \(m > 0\). We use induction on \(m\). If \(m = 1\), then

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{a}^{m-1} (\vec{x}))
\]

\[
\implies \chi (\vec{x}) \quad \text{(as } m = 1)\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x})\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}). \quad \text{(as } m = 1)\)

In the induction step, we have

\[
\vec{x} \xrightarrow{m+1, \langle \vec{a}, \vec{x} \rangle} \vec{a}^{m+1} (\vec{x}) \land \chi (\vec{a}^m (\vec{x}))
\]

\[
\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{a}^m (\vec{x}))
\]

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{a}^m (\vec{x}))\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{a}^m (\vec{x}))\)

\(\text{(as } m > 0)\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{a}^m (\vec{x}))\)

\(\text{(due to (3))}\)

\(\implies \vec{x} \xrightarrow{m+1, \langle \vec{a}, \vec{x} \rangle} \vec{a}^{m+1} (\vec{x})\)

\(\text{(by the induction hypothesis (4))}\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^{m+1} (\vec{x})\).

For exactness, we need to prove

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \implies \chi (\vec{a}^{m-1} (\vec{x}))
\]

for all \(m > 0\), which is trivial. \(\Box\)

So we just add \(\vec{a}\) 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

\[
\vec{a} (\vec{x}) \land \chi (\vec{x}) \implies \chi (\vec{a} (\vec{x})),
\]

then the following conditional acceleration technique is exact:

\[
\langle \vec{a}, \vec{x} \rangle \xrightarrow{T_{\text{2-inv}}} \vec{x} \xrightarrow{\vec{a}^m} \vec{x} \land \chi (\vec{x})
\]

**Proof** For soundness, we need to prove

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{x}) \implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x})
\]

\(\text{for all } m > 0\). We use induction on \(m\). If \(m = 1\), then

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{x})
\]

\(
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{x})
\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x})\)

\(\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}).\)

\(\text{(as } m = 1)\)

In the induction step, we have

\[
\vec{x} \xrightarrow{m+1, \langle \vec{a}, \vec{x} \rangle} \vec{a}^{m+1} (\vec{x}) \land \chi (\vec{x})
\]

\[
\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{x})
\]

\[
\implies \vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{x})\]

\(\text{(by the induction hypothesis (6))}\)

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \land \chi (\vec{a}^{m-1} (\vec{x}))\)

\(\text{(as } m > 0)\)

\[
\vec{x} \xrightarrow{m+1, \langle \vec{a}, \vec{x} \rangle} \vec{a}^{m+1} (\vec{x})\)

\(\text{(due to (5))}\)

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^{m+1} (\vec{x})\).

For exactness, we need to prove

\[
\vec{x} \xrightarrow{m, \langle \vec{a}, \vec{x} \rangle} \vec{a}^m (\vec{x}) \implies \chi (\vec{a}^{m-1} (\vec{x}))
\]

for all \(m > 0\), which is trivial. \(\Box\)

**Example 6** For the canonical acceleration problem of \(T_{\text{2-inv}}\), we obtain the derivation shown in Fig. 2, where \(\vec{a}_{\Delta} \vec{x} \vec{a} \vec{x} = (x_1+x_2)\). 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 \(T_{\text{2-inv}}\), 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 \(T_{\text{loop}}\) yields \(\psi\), then

\[
[\vec{x}^n = \vec{a}^n (\vec{x}) | \top | \phi (\vec{x})] \xrightarrow{T_{\text{loop}}} [\psi (\vec{y}) | \varphi (\vec{x}) | \top] \vec{a}
\]

where either Theorem 7 or Theorem 8 is applied in each \(\rightsquigarrow\)-step.

**Proof** As Theorem 3 applies, we have

\[
\phi (\vec{x}) \equiv \varphi_1 (\vec{x}) \land \varphi_2 (\vec{x}) \land \varphi_3 (\vec{x})
\]

\(\text{where}\)

\[
\varphi_1 (\vec{x}) \implies \varphi_1 (\vec{a} (\vec{x})),
\]

\(\Box\) Springer
\( \varphi_1(\vec{x}) \land \varphi_2(\vec{a}(\vec{x})) \implies \varphi_2(\vec{x}) \) \quad (8)
\( \varphi_1(\vec{x}) \land \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \implies \varphi_3(\vec{a}(\vec{x})). \) \quad (9)

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

\[
\begin{align*}
\vec{x}' &= \vec{a}^n(\vec{x}) \mid \top \mid \varphi(\vec{x}) \equiv_t e \\
\vec{x}' &= \vec{a}^n(\vec{x}) \mid \top \mid \varphi_1(\vec{x}) \land \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\vec{x}' &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\end{align*}
\]

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

\[
\begin{align*}
\vec{y} &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\vec{y} &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\vec{y} &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \mid \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\end{align*}
\]

Finally, if \( \varphi_3 \neq \top \), then Theorem 8 applies to \( (\varphi_3, \vec{a}) \) with \( \vec{\psi} := \varphi_1 \land \varphi_2 \) due to (9) and we obtain:

\[
\begin{align*}
\vec{y} &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \land \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\vec{y} &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \land \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\vec{y} &= \vec{a}^n(\vec{x}) \mid \varphi_1(\vec{x}) \land \varphi_2(\vec{x}) \land \varphi_3(\vec{x}) \equiv_t e \\
\end{align*}
\]

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.

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

\[
\begin{align*}
\textbf{while } & x_1 > 0 \land x_2 > 0 \land x_3 > 0 \textbf{ do} & (T_{2-c-inv}) \\
& \left( \begin{array}{c}
x_1 \\
\_x_2 \\
\_x_3 \\
\end{array} \right) \leftarrow \left( \begin{array}{c}
\_x_1 - x_1 + x_1 \\
x_2 + x_2 \\
x_3 + x_3 \\
\end{array} \right) \\
& \text{Let } \psi_{2-c-inv} := x_1 > 0 \land x_2 > 0 \land x_3 > 0, \\
& \vec{a}_{2-c-inv} := \left( \begin{array}{c}
x_1 - x_1 \\
x_2 + x_2 \\
x_3 + x_3 \\
\end{array} \right), \\
& \psi_{2-c-inv} \implies \vec{x}' = \vec{a}_{2-c-inv}(\vec{x}), \\
& \text{and let } x_i^{(m)} \text{ be the } i\text{th component of } \vec{a}_{2-c-inv}^{(m)}(\vec{x}). \text{ Starting with the canonical acceleration problem of } T_{2-c-inv}, \text{ we obtain the derivation shown in Fig. 3.}
\end{align*}
\]

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 \( \vec{x} \mapsto f(\vec{x}) \) from [27] (see Sect. 3.3 for details). But in contrast to [27], in our setting the “conditional” part \( \vec{\psi} \) does not need to be an invariant of the loop.

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

If

\[
\vec{\psi}(\vec{x}) \land \chi(\vec{x}) \implies mf(\vec{x}) - mf(\vec{a}(\vec{x})) \leq 1 \quad \text{and} \quad \vec{\psi}(\vec{x}) \land \neg \chi(\vec{x}) \implies mf(\vec{x}) \leq 0,
\]

then the following conditional acceleration technique is sound:

\[
(\chi, \vec{a}, \vec{\psi}) \mapsto \vec{x}' = \vec{a}^n(\vec{x}) \land n < mf(\vec{x}) + 1
\]

**Proof** We need to prove

\[
\vec{x} \rightarrow^m (\vec{\psi}, \vec{a}) \vec{a}^m(\vec{x}) \land m < mf(\vec{x}) + 1
\]

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

\[
mf(\vec{x}) > 0 \implies \neg \vec{\psi}(\vec{x}) \lor \chi(\vec{x}).
\]

We use induction on \( m \). If \( m = 1 \), then

\[
\vec{x} \rightarrow^m (\vec{\psi}, \vec{a}) \vec{a}^m(\vec{x}) \land m < mf(\vec{x}) + 1
\]

\[
\iff \vec{\psi}(\vec{x}) \land mf(\vec{x}) > 0
\]

(as \( m = 1 \))

\[
\iff \chi(\vec{x})
\]

(due to (13))

\[
\iff \vec{x} \rightarrow^m (\chi, \vec{a}) \vec{a}^m(\vec{x})
\]

(as \( m = 1 \))

In the induction step, assume

\[
\vec{x} \rightarrow^m (\vec{\psi}, \vec{a}) \vec{a}^m(\vec{x}) \land m < mf(\vec{x})
\]

Then, we have:

\[
\vec{x} \rightarrow^m (\vec{\psi}, \vec{a}) \vec{a}^m(\vec{x}) \land m < mf(\vec{x})
\]

\[
\vec{x} \rightarrow^m (\chi, \vec{a}) \vec{a}^m(\vec{x}) \land m < mf(\vec{x}) \land
\]

\[
\vec{x} \rightarrow^m (\vec{\psi}, \vec{a}) \vec{a}^m(\vec{x})
\]

(due to the induction hypothesis (12))
The combination of the calculus from Sect. 4 and the conditional acceleration techniques from Sect. 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.

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

$$
\begin{bmatrix}
    x_1 \\
    x_2
\end{bmatrix} = \left( \frac{n-n^2}{2} + x_2 \cdot (n-1) + x_1 > 0 \\
    (0 \geq x_2 - 1) \equiv (x_1 + x_2 \geq x_1 + x_2 + x_2 - 1) \\
\end{bmatrix}
$$

The reason is that $x_1$ does not behave monotonically, i.e., $x_1 > 0$ is neither an invariant nor a converse invariant. Essentially, $\mathcal{T}_{\text{ev-dec}}$ 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 $x_1$ weakly 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 $\varphi(\bar{x}) \equiv \bigwedge_{i=1}^{k} C_i$ where each clause $C_i$ contains an inequation $e_i(\bar{x}) > 0$ such that

$$
eq e_i(\bar{x}) \geq e_i(\bar{a}(\bar{x})) \implies e_i(\bar{a}(\bar{x})) \geq e_i(\bar{a}^2(\bar{x})).$$

then the following acceleration technique is sound:

$$
\mathcal{T}_{\text{loop}} \mapsto \bar{x}' = \bar{a}^n(\bar{x}) \land \bigwedge_{i=1}^{k} e_i(\bar{x}) > 0 \\
\text{If } C_i \equiv e_i > 0 \text{ for all } i \in [1, k], \text{ then it is exact.}
$$

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

Turning Theorem 10 into a conditional acceleration technique is straightforward.

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

$$
\varphi(\bar{x}) \land e_i(\bar{a}(\bar{x})) \implies e_i(\bar{a}(\bar{x})) \geq e_i(\bar{a}^2(\bar{x})).$$

then the following conditional acceleration technique is sound:

$$
((\chi, \bar{a}), \varphi) \mapsto \left( \bar{x}' = \bar{a}^n(\bar{x}) \land \bigwedge_{i=1}^{k} e_i(\bar{x}) > 0 \land e_i(\bar{a}^{n-1}(\bar{x})) > 0 \right)
$$

If $C_i \equiv e_i > 0 \text{ for all } i \in [1, k], \text{ then it is exact.}$
Proof. For soundness, we need to show
\[
\left( \bar{x} \stackrel{n}{\longrightarrow}_{(\bar{\phi}, \bar{a})} \bar{a}^n(\bar{x}) \right)
\land \bigwedge_{i=1}^{k} e_i(\bar{x}) > 0 \land e_i(\bar{a}^{n-1}(\bar{x})) > 0
\quad \implies \quad \bar{x} \stackrel{n}{\longrightarrow}_{(\bar{\chi}, \bar{a})} \bar{a}^n(\bar{x}).
\quad \tag{17}
\]
Assume
\[
\bar{x} \stackrel{n}{\longrightarrow}_{(\bar{\phi}, \bar{a})} \bar{a}^n(\bar{x}) \land \bigwedge_{i=1}^{k} e_i(\bar{x}) > 0 \land e_i(\bar{a}^{n-1}(\bar{x})) > 0.
\quad \tag{18}
\]
This implies
\[
\bigwedge_{i=0}^{n-1} \bar{\phi}(\bar{a}^i(\bar{x})�)
\quad \tag{19}
\]
In the following, we show
\[
\bigwedge_{i=1}^{k} \bigwedge_{m=0}^{n-1} e_i(\bar{a}^m(\bar{x})) \geq \min(e_i(\bar{x}), e_i(\bar{a}^{n-1}(\bar{x}))).
\quad \tag{20}
\]
Then, the claim (17) follows, as we have
\[
\bigwedge_{i=1}^{k} \bigwedge_{m=0}^{n-1} e_i(\bar{a}^m(\bar{x})) \geq \min(e_i(\bar{x}), e_i(\bar{a}^{n-1}(\bar{x})))
\implies \bigwedge_{i=1}^{k} \bigwedge_{m=0}^{n-1} e_i(\bar{a}^m(\bar{x})) > 0
\quad \text{(due to (18))}
\implies \bigwedge_{m=0}^{n-1} \chi(\bar{a}^m(\bar{x}))
\quad \text{(by definition of } e_i)
\iff \bar{x} \stackrel{n}{\longrightarrow}_{(\bar{\chi}, \bar{a})} \bar{a}^n(\bar{x}).
\]
Let \( i \) be arbitrary but fixed, let \( e = e_i \), and let \( j \) be the minimum natural number with
\[
e(\bar{a}^j(\bar{x})) = \max \{e(\bar{a}^m(\bar{x})) \mid m \in [0, n-1] \}.
\quad \tag{21}
\]
We first prove
\[
e(\bar{a}^m(\bar{x})) < e(\bar{a}^{m+1}(\bar{x}))
\quad \tag{22}
\]
for all \( m \in [0, j-1] \) by backward induction on \( m \). If \( m = j-1 \), then \( e(\bar{a}^m(\bar{x})) \).
Example 9 Consider the following variant of $T_{ev-dec}$

while $x_1 > 0 \land x_3 > 0$ do $(\frac{x_1}{x_3}) \leftarrow (\frac{x_1 + x_2}{x_2 + x_3})$.

i.e., we have $\bar{a} := (\frac{x_1 + x_2}{x_2 + x_3})$. Starting with its canonical acceleration problem, we get the derivation shown in Fig. 4, where the second step can be performed via Theorem 11 as

$$(\bar{\psi}(\bar{x}) \land e(\bar{x}) \geq e(\bar{a}(\bar{x})))$$

$$(x_3 > 0 \land x_1 \geq x_1 + x_2)$$

implies

$$(0 \geq x_2 - x_3)$$

$$(x_1 + x_2 \geq x_1 + x_2 - x_3)$$

Thus, in this case it suffices to require that $x_1$ monotonically, i.e.,

$$T := a := (\frac{x_1 + x_2}{x_2 + x_3})$$

Example 10 With Theorem 12, we can accelerate $T_{ev-inc}$ to

$$(\frac{x_1}{x_2}) = \left(\frac{n + n x_2 + x_1}{x_2 + 1}\right) \land 0 < x_1 \leq x_1 + x_2 \quad (\psi_{ev-inc})$$

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 $\psi_{ev-inc}$ only covers those runs of $T_{ev-inc}$ 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 $T_{exp}$, since the implication

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

is valid. Then the resulting formula is

$$(\frac{x_1}{x_2}) = \left(\frac{x_1 - n}{x_2 + n}\right) \land 0 < x_1 \leq x_1 - 1,$$

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 $\chi(\bar{x}) \equiv \bigwedge_{i=1}^{k} C_i$ where each clause $C_i$ contains an inequation $e_i(\bar{x}) > 0$ such that

$$\bar{\psi}(\bar{x}) \land e_i(\bar{x}) \leq e_i(\bar{a}(\bar{x})) \implies e_i(\bar{a}(\bar{x})) \leq e_i(\bar{a}^2(\bar{x})),$$  \hspace{1cm} (25)$$

then the following conditional acceleration technique is sound:

$$((\chi, \bar{a}), \bar{\psi}) \mapsto \bar{x}' = \bar{a}^\alpha(\bar{x}) \land \bigwedge_{i=1}^{k} \bigwedge_{i=1}^{n} 0 < e_i(\bar{x}) \leq e_i(\bar{a}(\bar{x}))$$

Proof We need to show

$$\bar{x} \longrightarrow_{n(\bar{\psi}, \bar{a})} \bar{a}^\alpha(\bar{x}) \land \bigwedge_{i=1}^{k} \bigwedge_{i=1}^{n} 0 < e_i(\bar{x}) \leq e_i(\bar{a}(\bar{x}))$$

$$\implies \bar{x} \longrightarrow_{n(\chi, \bar{a})} \bar{a}^\alpha(\bar{x}).$$

Due to $\bar{x} \longrightarrow_{n(\bar{\psi}, \bar{a})} \bar{a}^\alpha(\bar{x})$, we have

$$\bigwedge_{j=0}^{n-1} \bar{\psi}(\bar{a}(\bar{x})).$$

\[\Box\] Springer
Let $i$ be arbitrary but fixed and assume

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

We prove

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

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

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

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

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

In the induction step, note that (26) implies

$$\vec{\phi}(\vec{\alpha}^m(\vec{x}))$$

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

$$\vec{\phi}(\vec{\alpha}^m(\vec{x})) \land e_i(\vec{\alpha}^m(\vec{x})) \leq e_i(\vec{\alpha}^{m+1}(\vec{x})).$$

By (25), this implies

$$e_i(\vec{\alpha}^{m+1}(\vec{x})) \leq e_i(\vec{\alpha}^{m+2}(\vec{x})), \quad \text{as desired.}$$

**Example 11** Consider the following variant of $T_{ev-inc}$.

**while** $x_1 > 0 \land x_3 > 0$ **do** $\left(\frac{x_1}{x_2} \xrightarrow{\frac{x_1}{x_2}} \frac{x_1 + x_2}{x_2 + x_3}\right)$

So we have $\vec{\alpha} := \left(\frac{x_1 + x_2}{x_2 + x_3}\right)$. Starting with the canonical acceleration problem, we get the derivation shown in Fig. 5, where the second step can be performed via Theorem 13 as

$$(\vec{\phi}(\vec{\alpha})) \land e(\vec{\alpha}) \leq e(\vec{\alpha}(\vec{\alpha})))$$

$$\equiv (x_3 > 0 \land x_1 \leq x_1 + x_2)$$

implies

$$0 \leq x_2 + x_3$$

$$\equiv (x_1 + x_2 \leq x_1 + x_2 + x_3)$$

$$\equiv (e(\vec{\alpha}(\vec{\alpha}))) \leq e(\vec{\alpha}^2(\vec{\alpha}))).$$

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 $\vec{\psi}_{ev-inc}$ 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 $T_{\text{loop}}$ (denoted $\vec{x} \leadsto^\infty T_{\text{loop}} \bot$) if

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

If there is such a witness, then $T_{\text{loop}}$ is non-terminating. Otherwise, $T_{\text{loop}}$ 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 $\eta \in FO_{\text{QF}}(\vec{\phi}(\vec{x}))$ a certificate of non-termination for $T_{\text{loop}}$ if $\eta$ is satisfiable and

$$\forall \vec{x} \in \mathbb{Z}^d. \, \eta(\vec{x}) \rightarrow^\infty \vec{x} \leadsto^\infty T_{\text{loop}} \bot.$$
(Theorem 12) are non-terminating: \( T_{\text{inc}} \) diverges for all initial valuations that satisfy its guard \( x > 0 \) and \( T_{\text{ev-inc}} \) 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 Sect. 4, as long as all \( \sim \) 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 \( T_{\text{loop}} \).

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

\[ \tilde{x} = \tilde{a}^n(\tilde{x}). \]

It relates the values of the variables before evaluating the loop \( \tilde{x} \) to the values of the variables after evaluating the loop \( \tilde{x}' \) using the closed form \( \tilde{a}^n \). 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

\[ \| \psi | \hat{\psi} | \hat{\phi} \|_{\tilde{a}} \]

where \( \psi, \hat{\psi}, \hat{\phi} \in FO_{QF}(\mathcal{C}(\tilde{x})) \) and \( \tilde{a} : \mathbb{Z}^d \to \mathbb{Z}^d \) is a non-termination problem. It is consistent if every model of \( \psi \) is consistent. It is solved if it is consistent and \( \hat{\phi} \equiv \top \). The canonical non-termination problem of a loop \( T_{\text{loop}} \) is

\[ \| T \mid T \mid \varphi \|_{\tilde{a}}. \]

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

**Example 12** The canonical non-termination problem of \( T_{\text{ev-inc}} \) is

\[ \| T \mid T \mid x_1 > 0 \|_{\left( \frac{x_1 + x_2}{x_2 + 1} \right)}. \]

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

\[ nt : \text{Loop} \times FO_{QF}(\mathcal{C}(\tilde{x})) \to FO_{QF}(\mathcal{C}(\tilde{x})) \]

a conditional non-termination technique if \( \tilde{x} \to^{\infty}_{(\hat{\psi}, \hat{a})} \bot \land nt((\chi, \hat{a}), \hat{\phi}) \) implies \( \tilde{x} \to^{\infty}_{(\chi, \hat{a})} \bot \)

for all \( ((\chi, \hat{a}), \hat{\phi}) \in \text{dom}(nt) \) and all \( \tilde{x} \in \mathbb{Z}^d \).

Thus, we obtain the following variant of the calculus from Sect. 4.

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

\[ \emptyset \neq \chi \subseteq \hat{\psi} nt((\chi, \hat{a}), \hat{\phi}) = \psi_2 \]

\[ \| \psi_1 | \hat{\psi} | \hat{\phi} \|_{\tilde{a}} \sim_{nt} \| \psi_1 \cup \psi_2 \|_{\tilde{a}}, \]

where \( nt \) is a conditional non-termination technique.

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

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

\[ \| T \mid T \mid \varphi \|_{\tilde{a}} \sim^*_t \| \psi | \hat{\psi} | T \|_{\tilde{a}}, \]

and \( \psi \) is satisfiable, then \( \psi \) is a certificate of non-termination for \( T_{\text{loop}} \).

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

\[ \| \psi_1 | \hat{\psi} | \hat{\phi} \|_{\tilde{a}} \sim_{nt} \| \psi_1 \cup \psi_2 \|_{\tilde{a}}. \]

where \( \| \psi_1 | \hat{\psi} | \hat{\phi} \|_{\tilde{a}} \) is consistent and \( nt((\chi, \hat{a}), \hat{\phi}) = \psi_2 \).

We get

\[ \psi_1 \land \psi_2 \]

\[ \tilde{x} \to^{\infty}_{(\hat{\psi}, \hat{a})} \bot \land \psi_2 \]

\[ \tilde{x} \to^{\infty}_{(\hat{\psi}, \hat{a})} \bot \land \tilde{x} \to^{\infty}_{(\chi, \hat{a})} \bot \]

\[ \tilde{x} \to^{\infty}_{(\hat{\psi}, \hat{a})} \bot \]

The first step holds since \( \| \psi_1 | \hat{\psi} | \hat{\phi} \|_{\tilde{a}} \) is consistent and the second step holds since \( nt \) is a conditional non-termination technique.
This proves consistency of
\[
\| \psi_1 \land \psi_2 | \phi \land \chi | \phi \setminus \chi \|_{\tilde{\alpha}}
\]
\[
\equiv \| \psi_1 \cup \psi_2 | \phi \lor \chi | \phi \setminus \chi \|_{\tilde{\alpha}}.
\]
\[\Box\]

Analogously to well-foundedness of \(\leadsto\), well-foundedness of \(\leadsto_{nt}\) is trivial.

**Theorem 15** (Well-Foundedness of \(\leadsto_{nt}\)) The relation \(\leadsto_{nt}\) 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
\[
\hat{\phi}(\vec{x}) \land \chi(\vec{x}) \implies \chi(\vec{a}(\vec{x})),
\]
then
\[
(\chi, \vec{a}, \hat{\phi}) \mapsto \chi
\]
is a conditional non-termination technique.

**Proof** We need to prove
\[
\vec{x} \leadsto_{(\chi, \vec{a})} \perp \land \chi(\vec{x}) \implies \vec{x} \leadsto_{(\chi, \vec{a})} \perp.
\]

To this end, it suffices to prove
\[
\vec{x} \leadsto_{(\chi, \vec{a})} \perp \land \chi(\vec{x}) \implies \forall m \in \mathbb{N}, \chi(\vec{a}(\vec{x}))
\]
by the definition of non-termination (Definition 7). Assume
\[
\vec{x} \leadsto_{(\chi, \vec{a})} \perp \land \chi(\vec{x}).
\]

We prove \(\chi(\vec{a}(\vec{x}))\) for all \(m \in \mathbb{N}\) by induction on \(m\). If \(m = 0\), then the claim follows immediately. For the induction step, note that \(\vec{x} \leadsto_{(\chi, \vec{a})} \perp\) implies \(\vec{x} \leadsto_{(\chi, \vec{a})} \vec{a} \leadsto_{(\chi, \vec{a})} \perp\), which in turn implies \(\phi(\vec{a}(\vec{x}))\). Together with the induction hypothesis \(\chi(\vec{a}(\vec{x}))\), the claim follows from the prerequisites of the theorem. \[\Box\]

**Example 13** The canonical non-termination problem of \(\mathcal{T}_{inc}\) is
\[
\| \top | \top | x > 0 \|_{(x+1)}.
\]

Thus, in order to apply \(\leadsto_{nt}\) with Theorem 16, the only possible choice for the formula \(\chi\) is \(x > 0\). Furthermore, we have \(\phi := \top\) and \(\vec{a} := (x + 1)\). Hence, Theorem 16 is applicable if the implication
\[
\top \land x > 0 \implies x + 1 > 0
\]
is valid, which is clearly the case. Thus, we get
\[
\| \top | \top | x > 0 \|_{(x+1)} \equiv x > 0 | x > 0 | \top \|_{(x+1)}.
\]

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
\[
\hat{\phi}(\vec{x}) \land e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x})) \implies e_i(\vec{a}(\vec{x})) \leq e_i(\vec{a}(\vec{x})),
\]
then
\[
(\chi, \vec{a}, \hat{\phi}) \mapsto \bigwedge_{i=1}^{k} 0 < e_i(\vec{x}) \leq e_i(\vec{a}(\vec{x}))
\]
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
\[
\vec{x} \leadsto_{(\chi', \vec{a})} \perp \land \chi' \implies \vec{x} \leadsto_{(\chi', \vec{a})} \perp.
\]

Then it suffices to prove
\[
\vec{x} \leadsto_{(\chi', \vec{a})} \perp \land \chi' \implies \vec{x} \leadsto_{(\chi', \vec{a})} \perp
\]
(29)
since \(\chi'\) implies \(\chi\). By the prerequisites of the theorem, we have \(\phi \land \chi(\vec{x}) \implies \chi'(\vec{a}(\vec{x}))\). Thus, Theorem 16 applies to \((\chi', \vec{a})\). Hence, the claim (29) follows. \[\Box\]

**Example 14** We continue Example 12. To apply \(\leadsto_{nt}\) with Theorem 17 to the canonical non-termination problem of \(\mathcal{T}_{ev-inc}\), the only possible choice for the formula \(\chi\) is \(x_1 > 0\). Moreover, we again have \(\hat{\phi} := \top\), and \(\vec{a} := (x_1, x_2)\). Thus, Theorem 17 is applicable if
\[
\top \land x_1 \leq x_1 + x_2 \implies x_1 + x_2 \leq x_1 + 2 \cdot x_2 + 1
\]
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

$$\|T \mid T \mid x_1 > 0\|_a$$

$$\not\sim_{nt} \|0 < x_1 \leq x_1 + x_2 \mid x_1 > 0 \mid T\|_a.$$ 

Since $0 < x_1 \leq x_1 + x_2 \equiv x_1 > 0 \land x_2 \geq 0$ is satisfiable, $x_1 > 0 \land x_2 \geq 0$ is a certificate of non-termination for $\mathcal{T}_{ev-inc}$ 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

while $x_1 > 0$ do $(\frac{x_1}{x_2}) \leftarrow (\frac{x_2}{x_1})$. 

Theorem 16 is inapplicable, since

$x_1 > 0 \not\Rightarrow x_2 > 0$.

Furthermore, Theorem 17 is inapplicable, since

$x_1 > x_2 \not\Rightarrow x_2 > x_1$.

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 [22, 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

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

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,

$$((\chi, \vec{a}), \vec{\phi}) \mapsto \bigwedge_{i=1}^k e_i(\vec{x}) > 0 \land \bigwedge_{j \in \text{closure}_a(e_i)} x_j = \vec{a}(\vec{x})_j$$

is a conditional non-termination technique.

**Proof** Let

$$\chi' := \bigwedge_{i=1}^k e_i(\vec{x}) > 0 \land \bigwedge_{j \in \text{closure}_a(e_i)} x_j = \vec{a}(\vec{x})_j.$$ 

We need to prove

$$\vec{x} \rightarrow_{(\chi, \vec{a})} \chi' \iff \vec{x} \rightarrow_{(\chi', \vec{a})} \chi.'$$

Then, it suffices to prove

$$\vec{x} \rightarrow_{(\chi, \vec{a})} \chi' \iff \vec{x} \rightarrow_{(\chi', \vec{a})} \chi.'$$

(30)

since $\chi'$ implies $\chi$. By construction, we have

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

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

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

$$\text{closure}_a(x_1 > 0) = \{x_1, x_2\}.$$ 

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

$$\|T \mid T \mid x_1 > 0\|_{x_2}$$

$$\not\sim_{nt} \|x_1 > 0 \land x_1 = x_2 \mid x_1 > 0 \mid T\|_{x_2}.$$ 

Since the formula $x_1 > 0 \land x_1 = x_2$ is satisfiable, $x_1 > 0 \land x_1 = x_2$ 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:

while $x_1 > 0 \land x_3 > 0 \land x_4 + 1 > 0$ do

$$\left(\frac{x_1}{x_2}, \frac{x_3}{x_4}\right) \leftarrow \left(\frac{1}{x_2+x_1, x_3+x_1, -x_4}\right)$$

So we have

$$\varphi := x_1 > 0 \land x_3 > 0 \land x_4 + 1 > 0.$$
and
\[
\vec{a} := \left( \frac{1}{x_2 + x_1} \right).
\]

Then, the canonical non-termination problem is
\[
\| T | T | 0 \land x_3 > 0 \land x_4 + 1 > 0 \|_{\vec{a}}.
\]

First, our implementation applies Theorem 16 to \( x_1 > 0 \) (as \( x_1 > 0 \implies 1 > 0 \)), resulting in
\[
\| x_1 > 0 | x_1 > 0 | x_3 > 0 \land x_4 + 1 > 0 \|_{\vec{a}}.
\]

Next, it applies Theorem 17 to \( x_3 > 0 \), which is possible since
\[
x_1 > 0 \land x_3 \leq x_3 + x_2 \implies x_3 + x_2 \leq x_3 + 2 \cdot x_2 + x_1
\]
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:
\[
\| x_1 > 0 \land x_2 > 0 \land x_3 > 0 | x_1 > 0 \land x_3 > 0 \land x_4 + 1 > 0 \|_{\vec{a}}
\]
Here, we simplified
\[
0 < x_3 \leq x_3 + x_2
\]
to
\[
x_2 \geq 0 \land x_3 > 0.
\]

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 \((-1)^n \cdot x_4^{\text{init}}\), where \( x_4^{\text{init}} \) denotes the initial value of \( x_4 \). Thus, we apply Theorem 18 and we get
\[
\| x_1 > 0 \land x_2 > 0 \land x_3 > 0 \land x_4 = 0 | \varphi | T \|_{\vec{a}},
\]
which is solved. Here, we simplified the sub-formula \( x_4 + 1 > 0 \land x_4 = -x_4 \) 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
\[
x_1 > 0 \land x_2 \geq 0 \land x_3 > 0 \land x_4 = 0
\]
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., [21,33,34,41,42,47,49,51]), whereas we consider exact and under-approximating loop acceleration techniques. As many related approaches are discussed in Sect. 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 \( \{ A^i | i \in \mathbb{N} \} \) is finite. For such loops, Presburger Arithmetic is sufficient to construct an equivalent formula \( \psi \), 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 \( T_{exp} \)). 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 \( T_{2-incr} \)). Implementations are available in the tools FAST [3] and Flata [39]. 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 \( \xi \) of inequations of the form \( \pm x \pm y \leq c, x, y \in \overline{x, x'}, c \in \mathbb{Z} \). Then \( \xi \) induces the relation \( \vec{x} \mapsto \xi \vec{x}' \iff \xi(\vec{x}, \vec{x'}) \). In [43], 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 [8,43] 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 [36]. 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

\[ \{ \vec{a}^n(\vec{x}) \mid n \in \mathbb{N} \land \vec{x} \in \mathbb{Z}^d \land \psi(\vec{x}) \} . \]

As long as our calculus is used with the non-termination techniques presented in the current paper only (i.e., Theorems 16–18), 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 \( \text{FO}_{\text{OF}}(\psi(\vec{x})) \).

Recurrent sets are often synthesized by searching for suitable parameter values for template formulas [14,18,36,45,46,55] or for safety proofs [16,35,54]. In contrast to these search-based techniques, our current techniques use constraint solving only to check implications. As also indicated by our experiments (Sect. 9), this aspect leads to low runtimes and an efficient analysis.

Many proof techniques for proving non-termination of programs [11,18,36,46] 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 [11,14,16,35,45]. By integrating our calculus into a suitable program-analysis framework [22,27], 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 [25,40]. 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 [25,40] 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 \( T_{\text{inc}} \) 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 [37]. 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 Sect. 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 [26]. However, the decidability proof from [26] 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, Domènech, 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

\[
R := \left\{ \vec{x} \in \mathbb{Z}^d \mid \bigwedge_{i=1}^m e_i(\vec{x}) > 0 \right\}
\]

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 [46] 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 Theorem 2 applies (resulting in 1511 benchmarks). The reason is that PURRS has only limited support for initial conditions. Thus, LoAT’s results are correct only for all \( n > 0 \) (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 \( x_1, \ldots, x_t \). 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

9 Implementation and experiments

We implemented our approach in our open-source Loop Acceleration Tool LoAT [22,27]:

https://aprove-developers.github.io/LoAT

It uses Z3 [20] 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 [31], 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 [24].

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 \( x_1, \ldots, x_t \). 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 Sects. 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, [22]) and its implementation of acceleration via metering functions (Theorem 4, [28]), which also incorporates the improvements proposed in [27]. 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 [39], which implements the techniques to accelerate FMATs and octagonal relations discussed in Sect. 8. To this end, we used a straightforward transformation from LoAT’s native input format4 (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 60 s per example and a memory limit of 128 GB for each tool.

The results are shown 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 Sect. 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 are shown 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 $T_{2-\text{incdec}}$). 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

\[
\text{while } x_1 > 0 \land \ldots \text{ do } \left( \begin{array}{c} x_1 \\ x_2 \\ \ldots \end{array} \right) \leftarrow \left( \begin{array}{c} x_2 \\ \ldots \end{array} \right) \tag{31}
\]

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

LoAT 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

---

4 https://github.com/aprove-developers/LoAT#koat.
Our implementation found a closed form, but failed anyway. One of them was

\[x_3 > 0 \text{ do } \left( \frac{x_1}{x_2} \right) \left( \frac{x_1 + 1}{x_2 + 1} \right) \left( \frac{x_3}{x_3 + 1} \right).\]

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:

\[x_3^{(n)} = -\frac{1}{6} \cdot n^3 + \frac{1 - 3}{2} \cdot n^2 + \left( \frac{1}{2} + x_2 - \frac{1}{2} \right) \cdot n + x_3\]

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 decrease 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 Sect. 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 [30], iRankFinder [5], RevTerm [14], Ultimate [15], and VeryMax [45]. 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. 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. For AProVE, the developers provided an up-to-date version. For RevTerm, we followed the build instruction from [32] and sequentially executed the following command lines, as suggested by the developers:

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

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 \(\sim 10\) s 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 60 s and a memory limit of 128 GB for each tool. For RevTerm, we used a timeout of

---

5 The other two are structurally similar, but more complex.

6 [http://termination-portal.org/wiki/C_Integer_Programs](http://termination-portal.org/wiki/C_Integer_Programs)
30 s for the first invocation, and the remaining time for the second invocation.

The results are shown 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: 7

while \( x > 9 \land x_1 \geq 0 \) do \( \left( \frac{x}{x_1} \right) \left\langle \frac{x^2 + 2x_1 + 1}{x_1 + 1} \right\rangle \)

Most likely, the other tools fail for this example due to the presence of non-linear arithmetic. Our calculus from Sect. 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 different non-termination techniques, we also tested configurations where one of the non-termination techniques from Theorems 16 and 18 was disabled, respectively. The results are shown 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, \( \text{Ev-Inc} \) and \( \text{FP} \) 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 \( x_1 > 0 \land x_1 = x_2 \). 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 Theorems 17 and 18 are disabled (\( \text{Ev-Inc}, \text{FP} \) in Table 4). Then, the only remaining non-termination technique is \( \text{Non-Termination via Monotonic Increase} \). 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 Sect. 9).

\[ 1567523105272726.\text{koat.smt2} \]

\[ 7 \]

| Table 4 Comparison of LoAT versions |
|------------------------------------|
| LoAT                              |
| AProVE:                           |
| AProVE NT:                        |
| iRankFinder:                      |
| iRankFinder NT:                   |
| RevTerm:                          |
| Ultimate:                         |
| VeryMax:                          |
| \( \text{Ev-Inc} \):              |
| \( \text{Ev-Inc}, \text{FP} \):   |
| \( \text{Ev-Inc}, \text{FP} \):   |

Regard
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 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 (Sect. 3) and presented a calculus to combine acceleration techniques modularly (Sect. 4). Then, we showed how to combine existing (Sect. 5) and two novel (Sect. 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 [8,22,27,29,44] in the future. An empirical evaluation (Sect. 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 Sect. 4 can be adapted for proving non-termination in Sect. 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 Sect. 8) would improve the power of our approach for loop acceleration, as Flata exactly accelerated 49 loops where LoAT failed to do so (see Sect. 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).

Acknowledgements 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).

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Albert, E., Genaim, S., Martin-Martín, 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). https://arxiv.org/abs/cs/0512056 [cs.MS]
3. Brockschmidt, M., 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. Brockschmidt, M., 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. Brockschmidt, M., Ströder, T., Otto, C., Giesl, J.: Automated 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/Boigelot99.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., Girlea, 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(3:2014)
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.: TermCmp 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., Ishitaq, S., Khlaaf, H., Piterman, N.: T2: temporal property verification. In: TACAS ’16. LNCS 9636 (2016), pp. 387–393. https://doi.org/10.1007/978-3-662-49674-9_22
14. Chatterjee, K., Goharshady, E.K., Novotný, P., Zikelić, 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., Lengal, O., Li, Y., Tsai, M., Turini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In: PLDI ’18. pp. 135–150 (2018). https://doi.org/10.1145/3192366.3194208
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: FM/CAD ’14. pp. 67–74 (2014). https://doi.org/10.10110/FM/CAD2014.6987597
19. Dutertre, B.: Yices 2.2. In: CAV ’14. LNCS 8559 (2014), pp. 737–744. https://doi.org/10.1007/978-3-319-08867-9_49
20. 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
21. Farzan, A., Kincaid, Z.: Compositional recurrence analysis. In: FM/CAD ’15. pp. 57–64 (2015). https://doi.org/10.10110/FM/CAD.2015.7542253
22. Frohn, F., Giesl, J.: Proving non-termination via loop acceleration. In: FM/CAD ’19. pp. 221–230 (2019). https://doi.org/10.23919/FMACAD.2019.8894271
43. Konečný, F.: PTIME computation of transitive closures of octagonal relations. In: TACAS ’16. LNCS 9636 (2016), pp. 645–661. https://doi.org/10.1007/978-3-662-49674-9_42
44. 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
45. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using max-SMT. In: CAV ’14. LNCS 8559 (2014), pp. 779–796. https://doi.org/10.1007/978-3-319-08867-9_52
46. 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
47. 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
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. LNCS 11562 (2019), pp. 97–115. 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. Strejček, 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. LNCS 9636 (2016), pp. 54–70. 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

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.