RWTH Aachen

Department of Computer Science
Technical Report

Lower Runtime Bounds for Integer Programs

Florian Frohn, Matthias Naaf, Jera Hensel,
Marc Brockschmidt, Jürgen Giesl

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2016-3

RWTH Aachen · Department of Computer Science · April 2016

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Lower Runtime Bounds for Integer Programs*

F. Frohn[1], M. Naaf[1], J. Hensel[1], M. Brockschmidt[2], and J. Giesl[1]

[1] LuFG Informatik 2, RWTH Aachen University, Germany
[2] Microsoft Research, Cambridge, UK

Abstract. We present a technique to infer *lower* bounds on the worst-case runtime complexity of integer programs. To this end, we construct symbolic representations of program executions using a framework for iterative, under-approximating program simplification. The core of this simplification is a method for (under-approximating) program acceleration based on recurrence solving and a variation of ranking functions. Afterwards, we deduce *asymptotic* lower bounds from the resulting simplified programs. We implemented our technique in our tool LoAT and show that it infers non-trivial lower bounds for a large number of examples.

1 Introduction

Recent advances in program analysis yield efficient methods to find *upper* bounds on the complexity of sequential integer programs. Here, one usually considers "worst-case complexity", i.e., for any variable valuation, one analyzes the length of the longest execution starting from that valuation. But in many cases, in addition to upper bounds, it is also important to find *lower* bounds for this notion of complexity. Together with an analysis for upper bounds, this can be used to infer *tight* complexity bounds. Lower bounds also have important applications in security analysis, e.g., to detect possible denial-of-service or side-channel attacks, as programs whose runtime depends on a secret parameter "leak" information about that parameter. In general, *concrete* lower bounds that hold for arbitrary variable valuations can hardly be expressed concisely. In contrast, *asymptotic* bounds are easily understood by humans and witness possible attacks in a convenient way.

We first introduce our program model in Sect. 2. In Sect. 3, we show how to use a variation of classical ranking functions which we call *metering functions* to under-estimate the number of iterations of a simple loop (i.e., a single transition t looping on a location ℓ). Then, we present a framework for repeated program simplifications in Sect. 4. It simplifies full programs (with branching and sequences of possibly nested loops) to programs with only simple loops. Moreover, it eliminates simple loops by (under-)approximating their effect using a combination of metering functions and recurrence solving. In this way, programs are transformed to *simplified programs* without loops. In Sect. 5, we then show how to extract asymptotic lower bounds and variables that influence the runtime from simplified programs. Finally, we conclude with an experimental evaluation of our implementation LoAT in Sect. 6. All proofs can be found in the appendix.

* Supported by the DFG grant GI 274/6-1 and the Air Force Research Laboratory (AFRL).

Related Work While there are many techniques to infer *upper bounds* on the worst-case complexity of integer programs (e.g., [1–4, 8, 9, 14, 19, 26]), there is little work on *lower bounds*. In [3], it is briefly mentioned that their technique could also be adapted to infer lower instead of upper bounds for *abstract cost rules*, i.e., integer procedures with (possibly multiple) outputs. However, this only considers *best-case* lower bounds instead of worst-case lower bounds as in our technique. Upper and lower bounds for *cost relations* are inferred in [1]. Cost relations extend recurrence equations such that, e.g., non-determinism can be modeled. However, this technique also considers best-case lower bounds only.

A method for best-case lower bounds for logic programs is presented in [11]. Moreover, we recently introduced a technique to infer worst-case lower bounds for term rewrite systems (TRSs) [15]. However, TRSs differ fundamentally from the programs considered here, since they do not allow integers and have no notion of a "program start". Thus, the technique of [15], based on synthesizing families of reductions by automatic induction proofs, is very different to the present paper.

To simplify programs, we use a variant of *loop acceleration* to summarize the effect of applying a loop repeatedly. Acceleration is mostly used in over-approximating settings (e.g., [13, 17, 21, 24]), where handling non-determinism is challenging, as loop summaries have to cover *all* possible non-deterministic choices. However, our technique is under-approximating, i.e., we can instantiate non-deterministic values arbitrarily. In contrast to the under-approximating acceleration technique in [22], instead of quantifier elimination we use an adaptation of ranking functions to under-estimate the number of loop iterations symbolically.

2 Preliminaries

We consider sequential non-recursive imperative integer programs, allowing non-linear arithmetic and non-determinism, whereas heap usage and concurrency are not supported. While most existing abstractions that transform heap programs to integer programs are "over-approximations", we would need an under-approximating abstraction to ensure that the inference of worst-case lower bounds is sound. As in most related work, we treat numbers as mathematical integers $\mathbb{Z}$. However, the transformation from [12] can be used to handle machine integers correctly by inserting explicit normalization steps at possible overflows.

$\mathcal{A}(\mathcal{V})$ is the set of arithmetic terms[3] over the variables $\mathcal{V}$ and $\mathcal{F}(\mathcal{V})$ is the set of conjunctions[4] of (in)equations over $\mathcal{A}(\mathcal{V})$. So for $x, y \in \mathcal{V}$, $\mathcal{A}(\mathcal{V})$ contains terms like $x \cdot y + 2^y$ and $\mathcal{F}(\mathcal{V})$ contains formulas such as $x \cdot y \le 2^y \wedge y > 0$.

[3] Our implementation only supports addition, subtraction, multiplication, division, and exponentiation. Since we consider integer programs, we only allow programs where all variable values are integers (so in contrast to $x = \frac{1}{2}x$, the assignment $x = \frac{1}{2}x + \frac{1}{2}x^2$ is permitted). While our program simplification technique preserves this property, we do not allow division or exponentiation in the *initial* program to ensure its validity.

[4] Note that negations can be expressed by negating (in)equations directly, and disjunctions in programs can be expressed using multiple transitions.

```
ℓ0: y = 0
ℓ1: while x > 0 do
      y = y + x
      x = x − 1
    done
    z = y
ℓ2: while z > 0 do
      u = z − 1
ℓ3:   while u > 0 do
        u = u − random(0, ω)
      done
      z = z − 1
    done
```

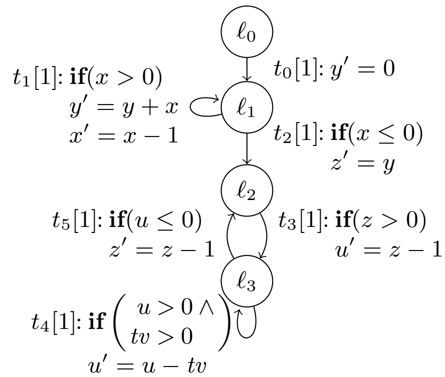

Fig. 1: Example integer program

We fix a finite set of *program variables* $\mathcal{PV}$ and represent integer programs as directed graphs. Nodes are program *locations* $\mathcal{L}$ and edges are program *transitions* $\mathcal{T}$ where $\mathcal{L}$ contains a *canonical start location* ℓ_0. W.l.o.g., no transition leads back to ℓ_0 and all transitions $\mathcal{T}$ are reachable from ℓ_0. To model non-deterministic program data, we introduce pairwise disjoint finite sets of *temporary variables* $\mathcal{TV}_\ell$ with $\mathcal{PV} \cap \mathcal{TV}_\ell = \varnothing$ and define $\mathcal{V}_\ell = \mathcal{PV} \cup \mathcal{TV}_\ell$ for all locations $\ell \in \mathcal{L}$.

Definition 1 (Programs). Configurations $(\ell, \boldsymbol{v})$ *consist of a location* $\ell \in \mathcal{L}$ *and a* valuation $\boldsymbol{v} : \mathcal{V}_\ell \to \mathbb{Z}$. $\mathcal{V}al_\ell = \mathcal{V}_\ell \to \mathbb{Z}$ *is the set of all valuations for* $\ell \in \mathcal{L}$ *and valuations are lifted to terms* $\mathcal{A}(\mathcal{V}_\ell)$ *and formulas* $\mathcal{F}(\mathcal{V}_\ell)$ *as usual. A* transition $t = (\ell, \gamma, \eta, c, \ell')$ *can evaluate a configuration* $(\ell, \boldsymbol{v})$ *if the* guard $\gamma \in \mathcal{F}(\mathcal{V}_\ell)$ *is satisfied (i.e.,* $\boldsymbol{v}(\gamma)$ *holds) to a new configuration* $(\ell', \boldsymbol{v}')$*. The* update $\eta : \mathcal{PV} \to \mathcal{A}(\mathcal{V}_\ell)$ *maps any* $x \in \mathcal{PV}$ *to a term* $\eta(x)$ *where* $\boldsymbol{v}(\eta(x)) \in \mathbb{Z}$ *for all* $\boldsymbol{v} \in \mathcal{V}al_\ell$*. It determines* $\boldsymbol{v}'$ *by setting* $\boldsymbol{v}'(x) = \boldsymbol{v}(\eta(x))$ *for* $x \in \mathcal{PV}$*, while* $\boldsymbol{v}'(x)$ *for* $x \in \mathcal{TV}_{\ell'}$ *is arbitrary. Such an* evaluation step *has* cost $k = \boldsymbol{v}(c)$ *for* $c \in \mathcal{A}(\mathcal{V}_\ell)$ *and is written* $(\ell, \boldsymbol{v}) \to_{t,k} (\ell', \boldsymbol{v}')$*. We use* $\mathsf{src}(t) = \ell$, $\mathsf{guard}(t) = \gamma$, $\mathsf{cost}(t) = c$, *and* $\mathsf{dest}(t) = \ell'$*. We sometimes drop the indices* t, k *and write* $(\ell, \boldsymbol{v}) \to_k^* (\ell', \boldsymbol{v}')$ *if* $(\ell, \boldsymbol{v}) \to_{k_1} \cdots \to_{k_m} (\ell', \boldsymbol{v}')$ *and* $\sum_{1 \leq i \leq m} k_i = k$*. A* program *is a set of transitions* $\mathcal{T}$*.*

Fig. 1 shows an example, where the pseudo-code on the left corresponds to the program on the right. Here, $\texttt{random}(x, y)$ returns a random integer m with $x < m < y$ and we fix $-\omega < m < \omega$ for all numbers m. The loop at location ℓ_1 sets y to a value that is quadratic in x. Thus, the loop at ℓ_2 is executed quadratically often where in each iteration, the inner loop at ℓ_3 may also be repeated quadratically often. Thus, the length of the program's worst-case execution is a polynomial of degree 4 in x. Our technique can infer such lower bounds automatically.

In the graph of Fig. 1, we write the costs of a transition in [ ] next to its name and represent the updates by imperative commands. We use x to refer to the value of the variable x before the update and x' to refer to x's value after the update. Here, $\mathcal{PV} = \{x, y, z, u\}$, $\mathcal{TV}_{\ell_3} = \{tv\}$, and $\mathcal{TV}_\ell = \varnothing$ for all locations $\ell \neq \ell_3$. We have $(\ell_3, \boldsymbol{v}) \to_{t_4} (\ell_3, \boldsymbol{v}')$ for all valuations $\boldsymbol{v}$ where $\boldsymbol{v}(u) > 0$, $\boldsymbol{v}(tv) > 0$, $\boldsymbol{v}'(u) = \boldsymbol{v}(u) - \boldsymbol{v}(tv)$, and $\boldsymbol{v}'(v) = \boldsymbol{v}(v)$ for all $v \in \{x, y, z\}$.

Our goal is to find a lower bound on the worst-case runtime of a program $\mathcal{T}$. To this end, we define its *derivation height* [18] by a function $\mathsf{dh}_{\mathcal{T}}$ that operates on valuations $\boldsymbol{v}$ of the program variables (i.e., $\boldsymbol{v}$ is not defined for temporary variables). The function $\mathsf{dh}_{\mathcal{T}}$ maps $\boldsymbol{v}$ to the maximum of the costs of all evaluation sequences starting in configurations $(\ell_0, \boldsymbol{v}_{\ell_0})$ where $\boldsymbol{v}_{\ell_0}$ is an extension of $\boldsymbol{v}$ to $\mathcal{V}_{\ell_0}$. So in our example we have $\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}) = 2$ for all valuations $\boldsymbol{v}$ where $\boldsymbol{v}(x) = 0$, since then we can only apply the transitions t_0 and t_2 once. For all valuations $\boldsymbol{v}$ with $\boldsymbol{v}(x) > 1$, our method will detect that the worst-case runtime of our program is at least $\frac{1}{8}\boldsymbol{v}(x)^4 + \frac{1}{4}\boldsymbol{v}(x)^3 + \frac{7}{8}\boldsymbol{v}(x)^2 + \frac{7}{4}\boldsymbol{v}(x)$. From this concrete lower bound, our approach will infer that the asymptotic runtime of the program is in $\Omega(x^4)$. In particular, the runtime of the program depends on x. Hence, if x is "secret", then the program is vulnerable to side-channel attacks.

Definition 2 (Derivation Height). *Let* $\mathcal{V}al = \mathcal{PV} \to \mathbb{Z}$. *The* derivation height $\mathsf{dh}_{\mathcal{T}} : \mathcal{V}al \to \mathbb{R}_{\geq 0} \cup \{\omega\}$ *of a program* $\mathcal{T}$ *is defined as* $\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}) = \sup\{k \in \mathbb{R} \mid \exists \boldsymbol{v}_{\ell_0} \in \mathcal{V}al_{\ell_0}, \ell \in \mathcal{L}, \boldsymbol{v}_\ell \in \mathcal{V}al_\ell .\ \boldsymbol{v}_{\ell_0}|_{\mathcal{PV}} = \boldsymbol{v} \ \wedge\ (\ell_0, \boldsymbol{v}_{\ell_0}) \to_k^* (\ell, \boldsymbol{v}_\ell)\}$.

Since $\to_k^*$ also permits evaluations with 0 steps, we always have $\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}) \geq 0$. Obviously, $\mathsf{dh}_{\mathcal{T}}$ is not computable in general, and thus our goal is to compute a lower bound that is as precise as possible (i.e., a lower bound which is, e.g., unbounded,[5] exponential, or a polynomial of a degree as high as possible).

3 Estimating the Number of Iterations of Simple Loops

We now show how to under-estimate the number of possible iterations of a *simple loop* $t = (\ell, \gamma, \eta, c, \ell)$. More precisely, we infer a term $b \in \mathcal{A}(\mathcal{V}_\ell)$ such that for all $\boldsymbol{v} \in \mathcal{V}al_\ell$ with $\boldsymbol{v} \models \gamma$, there is a $\boldsymbol{v}' \in \mathcal{V}al_\ell$ with $(\ell, \boldsymbol{v}) \to_t^{\lceil \boldsymbol{v}(b) \rceil} (\ell, \boldsymbol{v}')$. Here, $\lceil k \rceil = \min\{m \in \mathbb{N} \mid m \geq k\}$ for all $k \in \mathbb{R}$. Moreover, $(\ell, \boldsymbol{v}) \to_t^m (\ell, \boldsymbol{v}')$ means that $(\ell, \boldsymbol{v}) = (\ell, \boldsymbol{v}_0) \to_{t,k_1} (\ell, \boldsymbol{v}_1) \to_{t,k_2} \cdots \to_{t,k_m} (\ell, \boldsymbol{v}_m) = (\ell, \boldsymbol{v}')$ for some costs $k_1, \ldots, k_m$. We say that $(\ell, \boldsymbol{v}) \to_t^m (\ell, \boldsymbol{v}')$ *preserves* $\mathcal{TV}_\ell$ iff $\boldsymbol{v}(tv) = \boldsymbol{v}_i(tv) = \boldsymbol{v}'(tv)$ for all $tv \in \mathcal{TV}_\ell$ and all $0 \leq i \leq m$. Accordingly, we lift the update η to arbitrary arithmetic terms by leaving temporary variables unchanged (i.e., if $\mathcal{PV} = \{x_1, \ldots, x_n\}$ and $b \in \mathcal{A}(\mathcal{V}_\ell)$, then $\eta(b) = b[x_1/\eta(x_1), \ldots, x_n/\eta(x_n)]$, where $[x/a]$ denotes the substitution that replaces all occurrences of the variable x by a).

To find such estimations, we use an adaptation of ranking functions [2, 6, 25] which we call *metering functions*. We say that a term $b \in \mathcal{A}(\mathcal{V}_\ell)$ is a *ranking function*[6] for $t = (\ell, \gamma, \eta, c, \ell)$ iff the following conditions hold.

$$\gamma \implies b > 0 \qquad (1) \qquad\qquad \gamma \implies \eta(b) \leq b - 1 \qquad (2)$$

So e.g., x is a ranking function for t_1 in Fig. 1. If $\mathcal{TV}_\ell = \varnothing$, then for any valuation $\boldsymbol{v} \in \mathcal{V}al$, $\boldsymbol{v}(b)$ *over-estimates* the number of repetitions of the loop t: (2) ensures that $\boldsymbol{v}(b)$ decreases at least by 1 in each loop iteration, and (1) requires that $\boldsymbol{v}(b)$ is positive whenever the loop can be executed. In contrast, metering functions are *under-estimations* for the maximal number of repetitions of a simple loop.

[5] Programs with $\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}) = \omega$ result from non-termination or non-determinism. As an example, consider the program $x = \texttt{random}(0, \omega)$; **while** $x > 0$ **do** $x = x - 1$ **done**.

[6] In the following, we often use arithmetic terms $\mathcal{A}(\mathcal{V}_\ell)$ to denote functions $\mathcal{V}_\ell \to \mathbb{R}$.

Definition 3 (Metering Function). *Let $t = (\ell, \gamma, \eta, c, \ell)$ be a transition. We call $b \in \mathcal{A}(\mathcal{V}_\ell)$ a* metering function *for t iff the following conditions hold:*

$$\neg\gamma \implies b \le 0 \qquad (3) \qquad\qquad \gamma \implies \eta(b) \ge b - 1 \qquad (4)$$

Here, (4) ensures that $\boldsymbol{v}(b)$ decreases at most by 1 in each loop iteration, and (3) requires that $\boldsymbol{v}(b)$ is non-positive if the loop cannot be executed. Thus, the loop can be executed *at least* $\boldsymbol{v}(b)$ times (i.e., $\boldsymbol{v}(b)$ is an under-estimation).

For the transition t_1 in the example of Fig. 1, x is also a valid metering function. Condition (3) requires $\neg x > 0 \implies x \le 0$ and (4) requires $x > 0 \implies x - 1 \ge x - 1$. While x is a metering *and* a ranking function, $\frac{x}{2}$ is a metering, but not a ranking function for t_1. Similarly, x^2 is a ranking, but not a metering function for t_1. Thm. 4 states that a simple loop t with a metering function b can be executed at least $\lceil \boldsymbol{v}(b) \rceil$ times when starting with the valuation $\boldsymbol{v}$.

Theorem 4 (Metering Functions are Under-Estimations). *Let b be a metering function for $t = (\ell, \gamma, \eta, c, \ell)$. Then b* under-estimates *t, i.e., for all $\boldsymbol{v} \in \mathcal{V}al_\ell$ with $\boldsymbol{v} \models \gamma$ there is an evaluation $(\ell, \boldsymbol{v}) \to_t^{\lceil \boldsymbol{v}(b) \rceil} (\ell, \boldsymbol{v}')$ that preserves $\mathcal{TV}_\ell$.*

Our implementation builds upon a well-known transformation based on Farkas' Lemma [6, 25] to find *linear* metering functions. The basic idea is to search for coefficients of a linear template polynomial b such that (3) and (4) hold for all possible instantiations of the variables $\mathcal{V}_\ell$. In addition to (3) and (4), we also require (1) to avoid trivial solutions like $b = 0$. Here, the coefficients of b are existentially quantified, while the variables from $\mathcal{V}_\ell$ are universally quantified. As in [6, 25], eliminating the universal quantifiers using Farkas' Lemma allows us to use standard SMT solvers to search for b's coefficients efficiently.

When searching for a metering function for $t = (\ell, \gamma, \eta, c, \ell)$, one can omit constraints from γ that are irrelevant for t's termination. So if γ is $\varphi \wedge \psi$, $\psi \in \mathcal{F}(\mathcal{PV})$, and $\gamma \implies \eta(\psi)$, then it suffices to find a metering function b for $t' = (\ell, \varphi, \eta, c, \ell)$. The reason is that if $\boldsymbol{v} \models \gamma$ and $(\ell, \boldsymbol{v}) \to_{t'} (\ell, \boldsymbol{v}')$, then $\boldsymbol{v}' \models \psi$ (since $\boldsymbol{v} \models \gamma$ entails $\boldsymbol{v} \models \eta(\psi)$). Hence, if $\boldsymbol{v} \models \gamma$ then $(\ell, \boldsymbol{v}) \to_{t'}^{\lceil \boldsymbol{v}(b) \rceil} (\ell, \boldsymbol{v}')$ implies $(\ell, \boldsymbol{v}) \to_t^{\lceil \boldsymbol{v}(b) \rceil} (\ell, \boldsymbol{v}')$, i.e., b under-estimates t. So if $t = (\ell,\, x < y \wedge 0 < y,\, x' = x{+}1, c, \ell)$, we can consider $t' = (\ell,\, x < y,\, x' = x{+}1, c, \ell)$ instead. While t only has complex metering functions like $\min(y - x, y)$, t' has the metering function $y - x$.

Example 5 (Unbounded Loops). Loops $t = (\ell, \gamma, \eta, c, \ell)$ where the *whole* guard can be omitted (since $\gamma \implies \eta(\gamma)$) do not terminate. Here, we also allow ω as under-estimation. So for $\mathcal{T} = \{(\ell_0, \mathit{true}, \mathsf{id}, 1, \ell),\, t\}$ with $t = (\ell,\, 0 < x,\, x' = x{+}1,\, y,\, \ell)\}$, we can omit $0 < x$ since $0 < x \implies 0 < x + 1$. Hence, ω under-estimates the resulting loop $(\ell,\, \mathit{true},\, x' = x + 1,\, y,\, \ell)$ and thus, ω also under-estimates t.

4 Simplifying Programs to Compute Lower Bounds

We now define *processors* mapping programs to simpler programs. Processors are applied repeatedly to transform the program until extraction of a (concrete) lower bound is straightforward. For this, processors should be sound, i.e., any lower bound for the derivation height of $\mathsf{proc}(\mathcal{T})$ should also be a lower bound for $\mathcal{T}$.

Definition 6 (Sound Processor). *A mapping* proc *from programs to programs is* sound *iff* $\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}) \geq \mathsf{dh}_{\mathsf{proc}(\mathcal{T})}(\boldsymbol{v})$ *holds for all programs* $\mathcal{T}$ *and all* $\boldsymbol{v} \in \mathcal{V}al$.

In Sect. 4.1, we show how to *accelerate* a simple loop t to a transition which is equivalent to applying t multiple times (according to a metering function for t). The resulting program can be simplified by *chaining* subsequent transitions which may result in new simple loops, cf. Sect. 4.2. We describe a simplification strategy which alternates these steps repeatedly. In this way, we eventually obtain a *simplified* program without loops which directly gives rise to a concrete lower bound.

4.1 Accelerating Simple Loops

Consider a simple loop $t = (\ell, \gamma, \eta, c, \ell)$. For $m \in \mathbb{N}$, let η^m denote m applications of η. To accelerate t, we compute its *iterated* update and costs, i.e., a closed form η_{it} of η^{tv} and an under-approximation $c_{\mathrm{it}} \in \mathcal{A}(\mathcal{V}_\ell)$ of $\sum_{0 \leq i < tv} \eta^i(c)$ for a fresh temporary variable tv. If b under-estimates t, then we add the transition $(\ell,\, \gamma \wedge 0 < tv < b+1,\, \eta_{\mathrm{it}},\, c_{\mathrm{it}},\, \ell)$ to the program. It summarizes tv iterations of t, where tv is bounded by $\lceil b \rceil$. Here, η_{it} and c_{it} may also contain exponentiation (i.e., we can also infer exponential bounds).

For $\mathcal{PV} = \{x_1, \ldots, x_n\}$, the iterated update is computed by solving the recurrence equations $x^{(1)} = \eta(x)$ and $x^{(tv+1)} = \eta(x)[x_1/x_1^{(tv)}, \ldots, x_n/x_n^{(tv)}]$ for all $x \in \mathcal{PV}$ and $tv \geq 1$. So for the transition t_1 from Fig. 1 we get the recurrence equations $x^{(1)} = x-1$, $x^{(tv_1+1)} = x^{(tv_1)}-1$, $y^{(1)} = y+x$, and $y^{(tv_1+1)} = y^{(tv_1)}+x^{(tv_1)}$. Usually, they can easily be solved using state-of-the-art recurrence solvers [4]. In our example, we obtain the closed forms $\eta_{\mathrm{it}}(x) = x^{(tv_1)} = x - tv_1$ and $\eta_{\mathrm{it}}(y) = y^{(tv_1)} = y + tv_1 \cdot x - \frac{1}{2} tv_1^2 + \frac{1}{2} tv_1$. While $\eta_{\mathrm{it}}(y)$ contains rational coefficients, our approach ensures that η_{it} always maps integers to integers. Thus, we again obtain an integer program. We proceed similarly for the iterated cost of a transition, where we may under-approximate the solution of the recurrence equations $c^{(1)} = c$ and $c^{(tv+1)} = c^{(tv)} + c[x_1/x_1^{(tv)}, \ldots, x_n/x_n^{(tv)}]$. For t_1 in Fig. 1, we get $c^{(1)} = 1$ and $c^{(tv_1+1)} = c^{(tv_1)} + 1$ which leads to the closed form $c_{\mathrm{it}} = c^{(tv_1)} = tv_1$.

Theorem 7 (Loop Acceleration). *Let* $t = (\ell, \gamma, \eta, c, \ell) \in \mathcal{T}$ *and let* tv *be a fresh temporary variable. Moreover, let* $\eta_{\mathrm{it}}(x) = \eta^{tv}(x)$ *for all* $x \in \mathcal{PV}$ *and let* $c_{\mathrm{it}} \leq \sum_{0 \leq i < tv} \eta^i(c)$. *If* b *under-estimates* t, *then the processor mapping* $\mathcal{T}$ *to* $\mathcal{T} \cup \{(\ell,\, \gamma \wedge 0 < tv < b+1,\, \eta_{\mathrm{it}},\, c_{\mathrm{it}},\, \ell)\}$ *is sound.*

We say that the resulting new simple loop is *accelerated* and we refer to all simple loops which were not introduced by Thm. 7 as *non-accelerated*.

Example 8 (Non-Integer Metering Functions). Thm. 7 also allows metering functions that do not map to the integers. Let $\mathcal{T} = \{(\ell_0, \mathit{true}, \mathsf{id}, 1, \ell),\, t\}$ with $t = (\ell,\, 0 < x,\, x' = x-2,\, 1,\, \ell)$. Accelerating t with the metering function $\frac{x}{2}$ yields $(\ell,\, 0 < tv < \frac{x}{2}+1,\, x' = x - 2\,tv,\, tv,\, \ell)$. Note that $0 < tv < \frac{x}{2}+1$ implies $0 < x$ as tv and x range over $\mathbb{Z}$. Hence, $0 < x$ can be omitted in the resulting guard.

Example 9 (Unbounded Loops Cont.). In Ex. 5, ω under-estimates $t = (\ell,\, 0 < x,\, x' = x+1,\, y,\, \ell)$. The accelerated transition is $\overline{t} = (\ell,\, 0 < x \wedge \gamma',\, x' = x+tv,\, tv \cdot y,\, \ell)$, where γ' corresponds to $0 < tv < \omega + 1 = \omega$, i.e., tv has no upper bound.

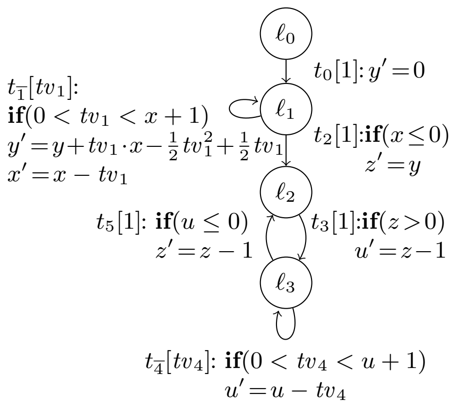

Fig. 2: Accelerating t_1 and t_4

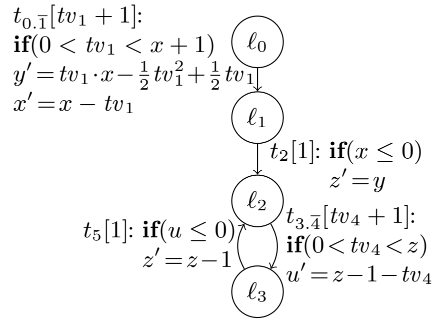

Fig. 3: Eliminating $t_{\overline{1}}$ and $t_{\overline{4}}$

If we cannot find a metering function or fail to obtain the closed form η_{it} or c_{it} for a simple loop t, then we can simplify t by eliminating temporary variables. To do so, we fix their values by adding suitable constraints to $\mathsf{guard}(t)$. As we are interested in witnesses for maximal computations, we use a heuristic that adds constraints $tv = a$ for temporary variables tv, where $a \in \mathcal{A}(\mathcal{V}_\ell)$ is a suitable upper or lower bound on tv's values, i.e., $\mathsf{guard}(t)$ implies $tv \leq a$ or $tv \geq a$. This is repeated until we find constraints which allow us to apply loop acceleration. Note that adding additional constraints to $\mathsf{guard}(t)$ is *always* sound in our setting.

Theorem 10 (Strengthening). *Let $t = (\ell, \gamma, \eta, c, \ell') \in \mathcal{T}$ and $\varphi \in \mathcal{F}(\mathcal{V}_\ell)$. Then the processor mapping $\mathcal{T}$ to $\mathcal{T} \setminus \{t\} \cup \{(\ell,\, \gamma \wedge \varphi,\, \eta,\, c,\, \ell')\}$ is sound.*

In t_4 from Fig. 1, γ contains $tv > 0$. So γ implies the bound $tv \geq 1$ since tv must be instantiated by integers. Hence, we add the constraint $tv = 1$. Now the update $u' = u - tv$ of the transition t_4 becomes $u' = u - 1$, and thus, u is a metering function. So after fixing $tv = 1$, t_4 can be accelerated similarly to t_1.

To simplify the program, we delete a simple loop t after trying to accelerate it. So we just keep the accelerated loop (or none, if acceleration of t still fails after eliminating all temporary variables by strengthening t's guard). For our example, we obtain the program in Fig. 2 with the accelerated transitions $t_{\overline{1}}$, $t_{\overline{4}}$.

Theorem 11 (Deletion). *For $t \in \mathcal{T}$, the processor mapping $\mathcal{T}$ to $\mathcal{T} \setminus \{t\}$ is sound.*

4.2 Chaining Transitions

After trying to accelerate all simple loops of a program, we can *chain* subsequent transitions t_1, t_2 by adding a new transition $t_{1.2}$ that simulates their combination. Afterwards, the transitions t_1 and t_2 can (but need not) be deleted with Thm. 11.

Theorem 12 (Chaining). *Let $t_1 = (\ell_1, \gamma_1, \eta_1, c_1, \ell_2)$ and $t_2 = (\ell_2, \gamma_2, \eta_2, c_2, \ell_3)$ with $t_1, t_2 \in \mathcal{T}$. Let* ren *be an injective function renaming the variables in $\mathcal{TV}_{\ell_2}$ to fresh ones and let*[7] $t_{1.2} = (\ell_1,\, \gamma_1 \wedge \mathrm{ren}(\eta_1(\gamma_2)),\, \mathrm{ren} \circ \eta_1 \circ \eta_2,\, c_1 + \mathrm{ren}(\eta_1(c_2)),$

[7] For all $x \in \mathcal{PV}$, $\mathrm{ren} \circ \eta_1 \circ \eta_2(x) = \mathrm{ren}(\eta_1(\eta_2(x))) = \eta_2(x)[x_1/\eta_1(x_1), \ldots, x_n/\eta_1(x_n), tv_1/\mathrm{ren}(tv_1), \ldots, tv_m/\mathrm{ren}(tv_m)]$ if $\mathcal{PV} = \{x_1, \ldots, x_n\}$ and $\mathcal{TV}_{\ell_2} = \{tv_1, \ldots, tv_m\}$.

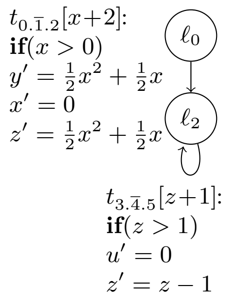

Fig. 4: Eliminating ℓ_1 and ℓ_3

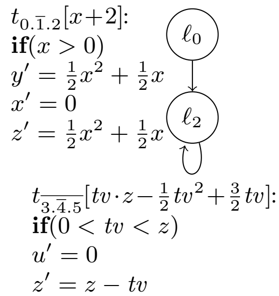

Fig. 5: Accelerating $t_{3.\overline{4}.5}$

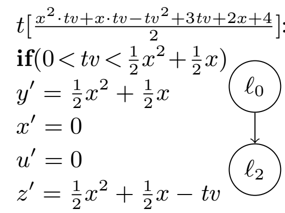

Fig. 6: Eliminating $t_{\overline{3.\overline{4}.5}}$

ℓ_3). Then the processor mapping $\mathcal{T}$ to $\mathcal{T} \cup \{t_{1.2}\}$ is sound. In the new program $\mathcal{T} \cup \{t_{1.2}\}$, the temporary variables of ℓ_1 are defined to be $\mathcal{TV}_{\ell_1} \cup \mathrm{ren}(\mathcal{TV}_{\ell_2})$.

One goal of chaining is *loop elimination* of all accelerated simple loops. To this end, we chain all subsequent transitions t', t where t is a simple loop and t' is no simple loop. Afterwards, we delete t. Moreover, once t' has been chained with all subsequent simple loops, then we also remove t', since its effect is now covered by the newly introduced (chained) transitions. So in our example from Fig. 2, we chain t_0 with $\overline{t}_1$ and t_3 with $\overline{t}_4$. The resulting program is depicted in Fig. 3, where we always simplify arithmetic terms and formulas to ease readability.

Chaining also allows *location elimination* by chaining all pairs of incoming and outgoing transitions for a location ℓ and removing them afterwards. It is advantageous to eliminate locations with just a single incoming transition first. This heuristic takes into account which locations were the entry points of loops. So for the example in Fig. 3, it would avoid chaining t_5 and $t_{3.\overline{4}}$ in order to eliminate ℓ_2. In this way, we avoid constructing chained transitions that correspond to a run from the "middle" of a loop to the "middle" of the next loop iteration.

So instead of eliminating ℓ_2, we chain $t_{0.\overline{1}}$ and t_2 as well as $t_{3.\overline{4}}$ and t_5 to eliminate the locations ℓ_1 and ℓ_3, leading to the program in Fig. 4. Here, the temporary variables tv_1 and tv_4 vanish since, before applying arithmetic simplifications, the guards of $t_{0.\overline{1}.2}$ resp. $t_{3.\overline{4}.5}$ imply $tv_1{=}x$ resp. $tv_4{=}z-1$.

Our overall approach for program simplification is shown in Alg. 1. Of course, this algorithm is a heuristic and other strategies for the application of the processors would also be possible. The set S in Steps 3 – 5 is needed to handle locations ℓ with multiple simple loops. The reason is that each transition t' with $\mathsf{dest}(t') = \ell$ should be chained with *each* of ℓ's simple loops before removing t'.

Alg. 1 terminates: In the loop 2.1 – 2.2, each iteration decreases the number of temporary variables in t. The loop 2 terminates since each iteration reduces the number of non-accelerated simple loops. In loop 4, the number of simple loops is decreasing and for loop 6, the number of reachable locations decreases. The overall loop terminates as it reduces the number of reachable locations. The reason is that the program does not have simple loops anymore when the algorithm reaches Step 6. Thus, at this point there is either a location ℓ which can be eliminated or the program does not have a path of length 2.

According to Alg. 1, in our example we go back to Step 1 and 2 and apply *Loop*

Algorithm 1 Program Simplification

While there is a path of length 2:
1. Apply *Deletion* to transitions whose guard is proved unsatisfiable.
2. While there is a non-accelerated simple loop t:
   - 2.1. Try to apply *Loop Acceleration* to t.
   - 2.2. If 2.1. failed and t uses temporary variables: Apply *Strengthening* to t to eliminate a temporary variable and go to 2.1.
   - 2.3. Apply *Deletion* to t.
3. Let $S = \varnothing$.
4. While there is a simple loop t:
   - 4.1. Apply *Chaining* to each pair t', t where $\mathsf{src}(t') \neq \mathsf{dest}(t') = \mathsf{src}(t)$.
   - 4.2. Add all these transitions t' to S and apply *Deletion* to t.
5. Apply *Deletion* to each transition in S.
6. While there is a location ℓ without simple loops but with incoming and outgoing transitions (starting with locations ℓ with just one incoming transition):
   - 6.1. Apply *Chaining* to each pair t', t where $\mathsf{dest}(t') = \mathsf{src}(t) = \ell$.
   - 6.2. Apply *Deletion* to each t where $\mathsf{src}(t) = \ell$ or $\mathsf{dest}(t) = \ell$.

Acceleration to transition $t_{3.\overline{4}.5}$. This transition has the metering function $z - 1$ and its iterated update sets u to 0 and z to $z - tv$ for a fresh temporary variable tv. To compute $t_{3.\overline{4}.5}$'s iterated costs, we have to find an under-approximation for the solution of the recurrence equations $c^{(1)} = z + 1$ and $c^{(tv+1)} = c^{(tv)} + z^{(tv)} + 1$. After computing the closed form $z - tv$ of $z^{(tv)}$, the second equation simplifies to $c^{(tv+1)} = c^{(tv)} + z - tv + 1$, which results in the closed form $c_{\text{it}} = c^{(tv)} = tv \cdot z - \frac{1}{2} tv^2 + \frac{3}{2} tv$. In this way, we obtain the program in Fig. 5. A final chaining step and deletion of the only simple loop yields the program in Fig. 6.

5 Asymptotic Lower Bounds for Simplified Programs

After Alg. 1, all program paths have length 1. We call such programs *simplified* and let $\mathcal{T}$ be a simplified program throughout this section. Now for any $\boldsymbol{v} \in \mathcal{V}al_{\ell_0}$,

$$\max\{\boldsymbol{v}(\mathsf{cost}(t)) \mid t \in \mathcal{T}, \boldsymbol{v} \models \mathsf{guard}(t)\}, \tag{5}$$

is a lower bound on $\mathcal{T}$'s derivation height $\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}|_{\mathcal{PV}})$, i.e., (5) is the maximal cost of those transitions whose guard is satisfied by $\boldsymbol{v}$. So for the program in Fig. 6, we obtain the bound $\frac{x^2 \cdot tv + x \cdot tv - tv^2 + 3tv + 2x + 4}{2}$ for all valuations with $\boldsymbol{v} \models 0 < tv < \frac{1}{2}x^2 + \frac{1}{2}x$. However, such bounds do not provide an intuitive understanding of the program's complexity and are also not suitable to detect possible attacks. Hence, we now show how to derive *asymptotic* lower bounds for simplified programs. These asymptotic bounds can easily be understood (i.e., a high lower bound can help programmers to improve their program to make it more efficient) and they identify potential attacks. After introducing our notion of asymptotic bounds in Sect. 5.1, we present a technique to derive them automatically in Sect. 5.2.

5.1 Asymptotic Bounds and Limit Problems

While $\mathsf{dh}_{\mathcal{T}}$ is defined on valuations, asymptotic bounds are usually defined for functions on $\mathbb{N}$. To bridge this gap, we use the common definition of complexity as a function of the size of the input. So the *runtime complexity* $\mathsf{rc}_{\mathcal{T}}(n)$ is the maximal cost of any evaluation that starts with a configuration where the sum of the absolute values of all program variables is at most n.

Definition 13 (Runtime Complexity). *Let* $|\boldsymbol{v}| = \sum_{x \in \mathcal{PV}} |\boldsymbol{v}(x)|$ *for all valuations* $\boldsymbol{v}$. *The* runtime complexity $\mathsf{rc}_{\mathcal{T}} : \mathbb{N} \to \mathbb{R}_{\geq 0} \cup \{\omega\}$ *is defined as* $\mathsf{rc}_{\mathcal{T}}(n) = \sup\{\mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}al, |\boldsymbol{v}| \leq n\}$.

Our goal is to derive an asymptotic lower bound for $\mathsf{rc}_{\mathcal{T}}$ from a simplified program $\mathcal{T}$. So for the program $\mathcal{T}$ in Fig. 6, we would like to derive $\mathsf{rc}_{\mathcal{T}}(n) \in \Omega(n^4)$. As usual, $f(n) \in \Omega(g(n))$ means that there is an $m > 0$ and an $n_0 \in \mathbb{N}$ such that $f(n) \geq m \cdot g(n)$ holds for all $n \geq n_0$. However, in general, the costs of a transition do not directly give rise to the desired asymptotic lower bound. For instance, in Fig. 6, the costs of the only transition are cubic, but the complexity of the program is a polynomial of degree 4 (since tv may be quadratic in x).

To infer an asymptotic lower bound from a transition $t \in \mathcal{T}$, we try to find an infinite family of valuations $\boldsymbol{v}_n \in \mathcal{V}al_{\ell_0}$ (parameterized by $n \in \mathbb{N}$) where there is an $n_0 \in \mathbb{N}$ such that $\boldsymbol{v}_n \models \mathsf{guard}(t)$ holds for all $n \geq n_0$. This implies $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(\boldsymbol{v}_n(\mathsf{cost}(t)))$, since for all $n \geq n_0$ we have:

$$\begin{aligned} \mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) &\geq \mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}_n|_{\mathcal{PV}}) && \text{as } |\,\boldsymbol{v}_n|_{\mathcal{PV}}\,| = |\boldsymbol{v}_n| \\ &\geq \boldsymbol{v}_n(\mathsf{cost}(t)) && \text{by (5)} \end{aligned}$$

We first normalize all constraints in $\mathsf{guard}(t)$ such that they have the form $a > 0$. Now our goal is to find infinitely many models $\boldsymbol{v}_n$ for a formula of the form $\bigwedge_{1 \leq i \leq k} (a_i > 0)$. Obviously, such a formula is satisfied if all terms a_i are positive constants or increase infinitely towards ω. Thus, we introduce a technique which tries to find out whether fixing the valuations of some variables and increasing or decreasing the valuations of others results in positive resp. increasing valuations of $a_1, \ldots, a_k$. Our technique operates on so-called *limit problems* $\{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}\}$ where $a_i \in \mathcal{A}(\mathcal{V}_{\ell_0})$ and $\bullet_i \in \{+, -, +_!, -_!\}$. Here, a^+ (resp. a^-) means that a grows towards ω (resp. $-\omega$) and $a^{+_!}$ (resp. $a^{-_!}$) means that a has to be a positive (resp. negative) constant. So we represent $\mathsf{guard}(t)$ by an *initial limit problem* $\{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}\}$ where $\bullet_i \in \{+, +_!\}$ for all $1 \leq i \leq k$. We say that a family of valuations $\boldsymbol{v}_n$ is a *solution* to a limit problem S iff $\boldsymbol{v}_n$ "satisfies" S for large n.

To define this notion formally, for any function $f : \mathbb{N} \to \mathbb{R}$ we say that $\lim_{n \mapsto \omega} f(n) = \omega$ (resp. $\lim_{n \mapsto \omega} f(n) = -\omega$) iff for every $m \in \mathbb{Z}$ there is an $n_0 \in \mathbb{N}$ such that $f(n) \geq m$ (resp. $f(n) \leq m$) holds for all $n \geq n_0$. Similarly, $\lim_{n \mapsto \omega} f(n) = m$ iff there is an n_0 such that $f(n) = m$ holds for all $n \geq n_0$.

Definition 14 (Solutions of Limit Problems). *For any function* $f : \mathbb{N} \to \mathbb{R}$ *and any* $\bullet \in \{+, -, +_!, -_!\}$, *we say that* f satisfies $\bullet$ *iff*

$$\begin{array}{ll} \lim_{n \mapsto \omega} f(n) = \omega, \quad \text{if } \bullet = + & \exists m \in \mathbb{Z}.\ \lim_{n \mapsto \omega} f(n) = m > 0, \text{ if } \bullet = +_! \\ \lim_{n \mapsto \omega} f(n) = -\omega, \text{ if } \bullet = - & \exists m \in \mathbb{Z}.\ \lim_{n \mapsto \omega} f(n) = m < 0, \text{ if } \bullet = -_! \end{array}$$

A family $\boldsymbol{v}_n$ *of valuations is a* solution *of a limit problem* S *iff for every* $a^\bullet \in S$, *the function* $\lambda n.\ \boldsymbol{v}_n(a)$ *satisfies* $\bullet$. *Here, "*$\lambda n.\ \boldsymbol{v}_n(a)$*" is the function from* $\mathbb{N} \to \mathbb{R}$ *that maps any* $n \in \mathbb{N}$ *to* $\boldsymbol{v}_n(a)$.

Example 15 (Bound for Fig. 6). In Fig. 6 where $\mathsf{guard}(t)$ is $0 < tv < \frac{1}{2}x^2 + \frac{1}{2}x$, the family $\boldsymbol{v}_n$ with $\boldsymbol{v}_n(tv) = \frac{1}{2}n^2 + \frac{1}{2}n - 1, \boldsymbol{v}_n(x) = n$, and $\boldsymbol{v}_n(y) = \boldsymbol{v}_n(z) = \boldsymbol{v}_n(u) = 0$ is a solution of the initial limit problem $\{tv^+, (\frac{1}{2}x^2 + \frac{1}{2}x - tv)^{+_!}\}$. The reason is that the function $\lambda n.\ \boldsymbol{v}_n(tv)$ that maps any $n \in \mathbb{N}$ to $\boldsymbol{v}_n(tv) = \frac{1}{2}n^2 + \frac{1}{2}n - 1$ satisfies $+$, i.e., $\lim_{n \mapsto \omega}(\frac{1}{2}n^2 + \frac{1}{2}n - 1) = \omega$. Similarly, the function $\lambda n.\ \boldsymbol{v}_n(\frac{1}{2}x^2 + \frac{1}{2}x - tv) = \lambda n.\ 1$ satisfies $+_!$. Sect. 5.2 will show how to infer such solutions of limit problems automatically. Thus, there is an n_0 such that $\boldsymbol{v}_n \models \mathsf{guard}(t)$ holds for all $n \geq n_0$. Hence, we get the asymptotic lower bound $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(\boldsymbol{v}_n(\mathsf{cost}(t))) = \Omega(\frac{1}{8}n^4 + \frac{1}{4}n^3 + \frac{7}{8}n^2 + \frac{7}{4}n) = \Omega(n^4)$.

Theorem 16 (Asymptotic Bounds for Simplified Programs). *Given a transition t of a simplified program $\mathcal{T}$ with $\mathsf{guard}(t) = a_1 > 0 \wedge \cdots \wedge a_k > 0$, let the family $\boldsymbol{v}_n$ be a solution of an initial limit problem $\{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}\}$ with $\bullet_i \in \{+, +_!\}$ for all $1 \leq i \leq k$. Then $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(\boldsymbol{v}_n(\mathsf{cost}(t)))$.*

Of course, if $\mathcal{T}$ has several transitions, then we try to take the one which results in the highest lower bound. Moreover, one should extend the initial limit problem $\{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}\}$ by $\mathsf{cost}(t)^+$. In this way, one searches for valuations $\boldsymbol{v}_n$ where $\boldsymbol{v}_n(\mathsf{cost}(t))$ depends on n, i.e., where the costs are not constant.

The costs are *unbounded* (i.e., they only depend on temporary variables) iff the initial limit problem $\{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}, \mathsf{cost}(t)^+\}$ has a solution $\boldsymbol{v}_n$ where $\boldsymbol{v}_n(x)$ is constant for all $x \in \mathcal{PV}$. Then we can even infer $\mathsf{rc}_{\mathcal{T}}(n) \in \Omega(\omega)$. For instance, after chaining the transition $\bar{t}$ of Ex. 9 with the transition from the start location (see Ex. 5), the resulting initial limit problem $\{x^{+_!}, tv^+, (tv \cdot y + 1)^+\}$ has the solution $\boldsymbol{v}_n$ with $\boldsymbol{v}_n(x) = \boldsymbol{v}_n(y) = 1$ and $\boldsymbol{v}_n(tv) = n$, which implies $\mathsf{rc}_{\mathcal{T}}(n) \in \Omega(\omega)$.

If the costs are not unbounded, we say that they *depend* on $x \in \mathcal{PV}$ iff the initial limit problem $\{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}, \mathsf{cost}(t)^+\}$ has a solution $\boldsymbol{v}_n$ where $\boldsymbol{v}_n(y)$ is constant for all $y \in \mathcal{PV} \setminus \{x\}$. If x corresponds to a "secret", then the program can be subject to side-channel attacks. For example, in Ex. 15 we have $\boldsymbol{v}_n(\mathsf{cost}(t)) = \frac{1}{8}n^4 + \frac{1}{4}n^3 + \frac{7}{8}n^2 + \frac{7}{4}n$. Since $\boldsymbol{v}_n$ maps all program variables except x to constants, the costs of our program depend on the program variable x. So if x is "secret", then the program is not safe from side-channel attacks.

Thm. 16 results in bounds of the form "$\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(\boldsymbol{v}_n(c))$", which depend on the *sizes* $|\boldsymbol{v}_n|$. Let $f(n) = \mathsf{rc}_{\mathcal{T}}(n)$, $g(n) = |\boldsymbol{v}_n|$, and let $\Omega(\boldsymbol{v}_n(c))$ have the form $\Omega(n^k)$ or $\Omega(k^n)$ for a $k \in \mathbb{N}$. Moreover for all $x \in \mathcal{PV}$, let $\boldsymbol{v}_n(x)$ be a polynomial of at most degree m, i.e., let $g(n) \in \mathcal{O}(n^m)$. Then the following observation from [15] allows us to infer a bound for $\mathsf{rc}_{\mathcal{T}}(n)$ instead of $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|)$.

Lemma 17 (Bounds for Function Composition). *Let $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $g : \mathbb{N} \to \mathbb{N}$ where $g(n) \in \mathcal{O}(n^m)$ for some $m \in \mathbb{N} \setminus \{0\}$. Moreover, let $f(n)$ be weakly and let $g(n)$ be strictly monotonically increasing for large enough n.*

- *If $f(g(n)) \in \Omega(n^k)$ with $k \in \mathbb{N}$, then $f(n) \in \Omega(n^{\frac{k}{m}})$.*
- *If $f(g(n)) \in \Omega(k^n)$ with $k \in \mathbb{N}$, then $f(n) \in \Omega(k^{\sqrt[m]{n}})$.*

Example 18 (Bound for Fig. 6 Continued). In Ex. 15, we inferred $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(n^4)$ where $\boldsymbol{v}_n(x) = n$ and $\boldsymbol{v}_n(y) = \boldsymbol{v}_n(z) = \boldsymbol{v}_n(u) = 0$. Hence, we have $|\boldsymbol{v}_n| = n \in \mathcal{O}(n^1)$. By Lemma 17, we obtain $\mathsf{rc}_{\mathcal{T}}(n) \in \Omega(n^{\frac{4}{1}}) = \Omega(n^4)$.

Example 19 (Non-Polynomial Bounds). Let $\mathcal{T}=\{(\ell_0, x=y^2, \mathsf{id}, y, \ell)\}$. By Def. 14, the family $\boldsymbol{v}_n$ with $\boldsymbol{v}_n(x) = n^2$ and $\boldsymbol{v}_n(y) = n$ is a solution of the initial limit problem $\{(x-y^2+1)^{+_!}, (y^2-x+1)^{+_!}, y^+\}$. Due to Thm. 16, this proves $\mathsf{rc}_\mathcal{T}(|\boldsymbol{v}_n|) \in \Omega(n)$. As $|\boldsymbol{v}_n| = n^2 + n \in \mathcal{O}(n^2)$, Lemma 17 results in $\mathsf{rc}_\mathcal{T}(n) \in \Omega(n^{\frac{1}{2}})$.

5.2 Transformation of Limit Problems

A limit problem S is *trivial* iff all terms in S are variables and there is no variable x with $x^{\bullet_1}, x^{\bullet_2} \in S$ and $\bullet_1 \neq \bullet_2$. For trivial limit problems S we can immediately obtain a particular solution $\boldsymbol{v}_n^S$ which instantiates variables "according to S".

Lemma 20 (Solving Trivial Limit Problems). *Let S be a trivial limit problem. Then $\boldsymbol{v}_n^S$ is a solution of S where for all $n \in \mathbb{N}$, $\boldsymbol{v}_n^S$ is defined as follows:*

$$\begin{array}{lll} \boldsymbol{v}_n^S(x) = n, \text{ if } x^+ \in S & \boldsymbol{v}_n^S(x) = 1, \text{ if } x^{+_!} \in S & \boldsymbol{v}_n^S(x) = 0, \text{ otherwise} \\ \boldsymbol{v}_n^S(x) = -n, \text{ if } x^- \in S & \boldsymbol{v}_n^S(x) = -1, \text{ if } x^{-_!} \in S & \end{array}$$

For instance, if $\mathcal{V}_{\ell_0} = \{x, y, tv\}$ and $S = \{x^+, y^{-_!}\}$, then S is a trivial limit problem and $\boldsymbol{v}_n^S$ with $\boldsymbol{v}_n^S(x) = n, \boldsymbol{v}_n^S(y) = -1$, and $\boldsymbol{v}_n^S(tv) = 0$ is a solution for S.

However, in general the initial limit problem $S = \{a_1^{\bullet_1}, \dots, a_k^{\bullet_k}, \mathsf{cost}(t)^+\}$ is not trivial. Therefore, we now define a transformation $\rightsquigarrow$ to simplify limit problems until one reaches a trivial problem. With our transformation, $S \rightsquigarrow S'$ ensures that each solution of S' also gives rise to a solution of S.

If S contains $f(a_1, a_2)^\bullet$ for some standard arithmetic operation f like addition, subtraction, multiplication, division, and exponentiation, we use a so-called *limit vector* $(\bullet_1, \bullet_2)$ with $\bullet_i \in \{+, -, +_!, -_!\}$ to characterize for which kinds of arguments the operation f is increasing (if $\bullet = +$) resp. decreasing (if $\bullet = -$) resp. a positive or negative constant (if $\bullet = +_!$ or $\bullet = -_!$).[8] Then S can be transformed into the new limit problem $S \setminus \{f(a_1, a_2)^\bullet\} \cup \{a_1^{\bullet_1}, a_2^{\bullet_2}\}$.

For example, $(+, +_!)$ is an increasing limit vector for subtraction. The reason is that $a_1 - a_2$ is increasing if a_1 is increasing and a_2 is a positive constant. Hence, our transformation $\rightsquigarrow$ allows us to replace $(a_1 - a_2)^+$ by a_1^+ and $a_2^{+_!}$.

To define limit vectors formally, we say that $(\bullet_1, \bullet_2)$ is an *increasing* (resp. *decreasing*) *limit vector* for f iff the function $\lambda n.\ f(g(n), h(n))$ satisfies $+$ (resp. $-$) for any functions g and h that satisfy $\bullet_1$ and $\bullet_2$, respectively. Here, "$\lambda n.\ f(g(n), h(n))$" is the function from $\mathbb{N} \to \mathbb{R}$ that maps any $n \in \mathbb{N}$ to $f(g(n), h(n))$. Similarly, $(\bullet_1, \bullet_2)$ is a *positive* (resp. *negative*) *limit vector* for f iff $\lambda n.\ f(g(n), h(n))$ satisfies $+_!$ (resp. $-_!$) for any functions g and h that satisfy $\bullet_1$ and $\bullet_2$, respectively.

With this definition, $(+, +_!)$ is indeed an increasing limit vector for subtraction, since $\lim_{n\mapsto\omega} g(n) = \omega$ and $\lim_{n\mapsto\omega} h(n) = m$ with $m > 0$ implies $\lim_{n\mapsto\omega}(g(n) - h(n)) = \omega$. In other words, if $g(n)$ satisfies $+$ and $h(n)$ satisfies $+_!$, then $g(n) - h(n)$ satisfies $+$ as well. In contrast, $(+, +)$ is not an increasing limit vector for subtraction. To see this, consider the functions $g(n) = h(n) = n$. Both $g(n)$ and $h(n)$ satisfy $+$, whereas $g(n) - h(n) = 0$ does not satisfy $+$. Similarly, $(+_!, +_!)$ is

[8] To ease the presentation, we restrict ourselves to binary operations f. For operations of arity m, one would need limit vectors of the form $(\bullet_1, \dots, \bullet_m)$.

not a positive limit vector for subtraction, since for $g(n) = 1$ and $h(n) = 2$, both $g(n)$ and $h(n)$ satisfy $+_!$, but $g(n) - h(n) = -1$ does not satisfy $+_!$.

Limit vectors can be used to simplify limit problems, cf. (A) in the following definition. Moreover, for numbers $m \in \mathbb{Z}$, one can easily simplify constraints of the form $m^{+_!}$ and $m^{-_!}$ (e.g., $2^{+_!}$ is obviously satisfied since $2 > 0$), cf. (B).

Definition 21 ($\rightsquigarrow$). *Let S be a limit problem. We have:*

(A) $S \cup \{f(a_1, a_2)^\bullet\} \rightsquigarrow S \cup \{a_1^{\bullet_1}, a_2^{\bullet_2}\}$ if $\bullet$ is $+$ (resp. $-, +_!, -_!$) and $(\bullet_1, \bullet_2)$ is an increasing (resp. decreasing, positive, negative) limit vector for f

(B) $S \cup \{m^{+_!}\} \rightsquigarrow S$ if $m \in \mathbb{Z}$ with $m > 0$, $S \cup \{m^{-_!}\} \rightsquigarrow S$ if $m < 0$

Example 22 (Bound for Fig. 6 Continued). For the initial limit problem from Ex. 15, we have $\{tv^+, (\frac{1}{2}x^2 + \frac{1}{2}x - tv)^{+_!}\} \rightsquigarrow \{tv^+, (\frac{1}{2}x^2 + \frac{1}{2}x)^{+_!}, tv^{-_!}\} \rightsquigarrow \{tv^+, (\frac{1}{2}x^2)^{+_!}, (\frac{1}{2}x)^{+_!}, tv^{-_!}\} \rightsquigarrow^* \{tv^+, x^{+_!}, tv^{-_!}\}$ using the positive limit vector $(+_!, -_!)$ for subtraction and the positive limit vector $(+_!, +_!)$ for addition.

The resulting problem in Ex. 22 is not trivial as it contains tv^+ and $tv^{-_!}$, i.e., we failed to compute an asymptotic lower bound. However, if we substitute tv with its upper bound $\frac{1}{2}x^2 + \frac{1}{2}x - 1$, then we could reduce the initial limit problem to a trivial one. Hence, we now extend $\rightsquigarrow$ by allowing to apply substitutions.

Definition 23 ($\rightsquigarrow$ Continued). *Let S be a limit problem and let $\sigma : \mathcal{V}_{\ell_0} \to \mathcal{A}(\mathcal{V}_{\ell_0})$ be a substitution such that x does not occur in $x\sigma$ and $\boldsymbol{v}(x\sigma) \in \mathbb{Z}$ for all valuations $\boldsymbol{v} \in \mathcal{V}al_{\ell_0}$ and all $x \in \mathcal{V}_{\ell_0}$. Then we have*[9]

(C) $S \rightsquigarrow^\sigma S\sigma$

Example 24 (Bound for Fig. 6 Continued). For the initial limit problem from Ex. 15, we now have[10] $\{tv^+, (\frac{1}{2}x^2 + \frac{1}{2}x - tv)^{+_!}\} \rightsquigarrow^{[tv/\frac{1}{2}x^2+\frac{1}{2}x-1]} \{(\frac{1}{2}x^2 + \frac{1}{2}x - 1)^+, 1^{+_!}\} \rightsquigarrow \{(\frac{1}{2}x^2 + \frac{1}{2}x - 1)^+\} \rightsquigarrow \{(\frac{1}{2}x^2 + \frac{1}{2}x)^+, 1^{+_!}\} \rightsquigarrow^* \{x^+\}$, which is trivial.

Although Def. 23 requires that variables may only be instantiated by integer terms, it is also useful to handle limit problems that contain non-integer terms.

Example 25 (Non-Integer Metering Functions Continued). After chaining the accelerated transition of Ex. 8 with the transition from the start location, for the resulting initial limit problem we get $\{tv^+, (\frac{1}{2}x - tv + 1)^{+_!}, (tv + 1)^+\} \rightsquigarrow^2 \{tv^+, (\frac{1}{2}x - tv + 1)^{+_!}\} \rightsquigarrow^{[x/2tv-1]} \{tv^+, \frac{1}{2}^{+_!}\} \rightsquigarrow \{tv^+, 1^{+_!}, 2^{+_!}\} \rightsquigarrow^2 \{tv^+\}$, using the positive limit vector $(+_!, +_!)$ for division. This allows us to infer $\mathsf{rc}_{\mathcal{T}}(n) \in \Omega(n)$.

However, up to now we cannot prove that, e.g., a transition t with $\mathsf{guard}(t) = x^2 - x > 0$ and $\mathsf{cost}(t) = x$ has a linear lower bound, since $(+, +)$ is not an increasing limit vector for subtraction. To handle such cases, the following rules allow us to neglect polynomial sub-expressions if they are "dominated" by other polynomials of higher degree or by exponential sub-expressions.

Definition 26 ($\rightsquigarrow$ Continued). *Let S be a limit problem, let $\pm \in \{+, -\}$, and let $a, b, e \in \mathcal{A}(\{x\})$ be (univariate) polynomials. Then we have:*

(D) $S \cup \{(a \pm b)^\bullet\} \rightsquigarrow S \cup \{a^\bullet\}$, if $\bullet \in \{+, -\}$, and a has a higher degree than b

[9] The other rules for $\rightsquigarrow$ are implicitly labeled with the identical substitution id.

[10] $\sigma = [tv/\frac{1}{2}x^2 + \frac{1}{2}x - 1]$ satisfies the condition $\boldsymbol{v}(y\sigma) \in \mathbb{Z}$ for all $\boldsymbol{v} \in \mathcal{V}al_{\ell_0}$ and $y \in \mathcal{V}_{\ell_0}$.

(E) $S \cup \{(a^e \pm b)^+\} \rightsquigarrow S \cup \{(a-1)^\bullet, e^+\}$*, if* $\bullet \in \{+, +_!\}$*.*

Thus, $\{(x^2 - x)^+\} \rightsquigarrow \{(x^2)^+\} = \{(x \cdot x)^+\} \rightsquigarrow \{x^+\}$ by the increasing limit vector $(+,+)$ for multiplication. Similarly, $\{(2^x - x^3)^+\} \rightsquigarrow \{(2-1)^{+_!}, x^+\} \rightsquigarrow \{x^+\}$. Rule (E) can also be used to handle problems like $(a^e)^+$ (by choosing $b = 0$).

Thm. 27 states that $\rightsquigarrow$ is indeed correct. When constructing the valuation from the resulting trivial limit problem, one has to take the substitutions into account which were used in the derivation. Here, $(\boldsymbol{v}_n \circ \sigma)(x)$ stands for $\boldsymbol{v}_n(\sigma(x))$.

Theorem 27 (Correctness of $\rightsquigarrow$). *If $S \rightsquigarrow^\sigma S'$ and the family $\boldsymbol{v}_n$ is a solution of S', then $\boldsymbol{v}_n \circ \sigma$ is a solution of S.*

Example 28 (Bound for Fig. 6 Continued). Ex. 24 leads to the solution $\boldsymbol{v}'_n \circ \sigma$ of the initial limit problem for the program from Fig. 6 where $\sigma = [tv/\frac{1}{2}x^2 + \frac{1}{2}x - 1]$, $\boldsymbol{v}'_n(x) = n$, and $\boldsymbol{v}'_n(tv) = \boldsymbol{v}'_n(y) = \boldsymbol{v}'_n(z) = \boldsymbol{v}'_n(u) = 0$. Hence, $\boldsymbol{v}'_n \circ \sigma = \boldsymbol{v}_n$ where $\boldsymbol{v}_n$ is as in Ex. 15. As explained in Ex. 18, this proves $\mathsf{rc}_\mathcal{T}(n) \in \Omega(n^4)$.

So we start with an initial limit problem $S = \{a_1^{\bullet_1}, \ldots, a_k^{\bullet_k}, \mathsf{cost}(t)^+\}$ that represents $\mathsf{guard}(t)$ and requires non-constant costs, and transform S with $\rightsquigarrow$ into a trivial S', i.e., $S \rightsquigarrow^{\sigma_1} \ldots \rightsquigarrow^{\sigma_m} S'$. For automation, one should leave the $\bullet_i$ in the initial problem S open, and only instantiate them by a value from $\{+, +_!\}$ when this is needed to apply a particular rule for the transformation $\rightsquigarrow$. Then the resulting family $\boldsymbol{v}_n^{S'}$ of valuations gives rise to a solution $\boldsymbol{v}_n^{S'} \circ \sigma_m \circ \ldots \circ \sigma_1$ of S. Thus, we have $\mathsf{rc}_\mathcal{T}(|\boldsymbol{v}_n^{S'} \circ \sigma|) \in \Omega(\boldsymbol{v}_n^{S'}(\sigma(\mathsf{cost}(t))))$, where $\sigma = \sigma_m \circ \ldots \circ \sigma_1$, which leads to a lower bound for $\mathsf{rc}_\mathcal{T}(n)$ with Lemma 17.

Our implementation uses the following strategy to apply the rules from Def. 21, 23, 26 for $\rightsquigarrow$. Initially, we reduce the number of variables by propagating bounds implied by the guard, i.e., if $\gamma \implies x \geq a$ or $\gamma \implies x \leq a$ for some $a \in \mathcal{A}(\mathcal{V}_{\ell_0} \setminus \{x\})$, then we apply the substitution $[x/a]$ to the initial limit problem by rule (C). For example, we simplify the limit problem from Ex. 19 by instantiating x with y^2, as the guard of the corresponding transition implies $x = y^2$. So here, we get $\{(x - y^2 + 1)^{+_!}, (y^2 - x + 1)^{+_!}, y^+\} \rightsquigarrow^{[x/y^2]} \{1^{+_!}, y^+\} \rightsquigarrow \{y^+\}$. Afterwards, we use (B) and (D) with highest and (E) with second highest priority. The third priority is trying to apply (A) to univariate terms (since processing univariate terms helps to guide the search). As fourth priority, we apply (C) with a substitution $[x/m]$ if $x^{+_!}$ or $x^{-_!}$ in S, where we use SMT solving to find a suitable $m \in \mathbb{Z}$. Otherwise, we apply (A) to multivariate terms. Since $\rightsquigarrow$ is well founded and, except for (C), finitely branching, one may also backtrack and explore alternative applications of $\rightsquigarrow$. In particular, we backtrack if we obtain a contradictory limit problem. Moreover, if we obtain a trivial S' where $\boldsymbol{v}_n^{S'}(\sigma(\mathsf{cost}(t)))$ is a polynomial, but $\mathsf{cost}(t)$ is a polynomial of higher degree or an exponential function, then we backtrack to search for other solutions which might lead to a higher lower bound. However, our implementation can of course fail, since solvability of limit problems is undecidable (due to Hilbert's Tenth Problem).

6 Experiments and Conclusion

We presented the first technique to infer lower bounds on the worst-case runtime complexity of integer programs, based on a modular program simplification framework. The main simplification technique is *loop acceleration*, which relies on *recurrence solving* and *metering functions*, an adaptation of classical ranking functions. By eliminating loops and locations via *chaining*, we eventually obtain *simplified programs*. We presented a technique to infer *asymptotic lower bounds* from simplified programs, which can also be used to find vulnerabilities.

Our implementation LoAT ("Lower Bounds Analysis Tool") is freely available at [23]. It was inspired by KoAT [8], which alternates runtime- and size-analysis to infer *upper* bounds in a modular way. Similarly, LoAT alternates runtime-analysis and recurrence solving to transform loops to loop-free transitions independently. LoAT uses the recurrence solver PURRS [4] and the SMT solver Z3 [10].

We evaluated LoAT on the benchmarks [5] from the evaluation of [8]. We omitted 50 recursive programs, since our approach cannot yet handle recursion. As we know of no other tool to compute worst-case lower bounds for integer programs, we compared our results with the asymptotically smallest results of leading tools for upper bounds: KoAT, CoFloCo [14], Loopus [26], RanK [2]. We did not compare with PUBS [1], since the cost relations analyzed by PUBS significantly differ from the integer programs handled by LoAT. Moreover, as PUBS computes *best-case* lower bounds, such a comparison would be meaningless since the worst-case lower bounds computed by LoAT are no valid best-case lower bounds. We used a timeout of 60 seconds. In the following, we disregard 132 examples where $\mathsf{rc}_{\mathcal{T}}(n) \in \mathcal{O}(1)$ was proved since there is no non-trivial lower bound in these cases.

$\mathsf{rc}_{\mathcal{T}}(n)$	$\Omega(1)$	$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^3)$	$\Omega(n^4)$	EXP	$\Omega(\omega)$
$\mathcal{O}(1)$	(132)	–	–	–	–	–	–
$\mathcal{O}(n)$	45	125	–	–	–	–	–
$\mathcal{O}(n^2)$	9	18	33	–	–	–	–
$\mathcal{O}(n^3)$	2	–	–	3	–	–	–
$\mathcal{O}(n^4)$	1	–	–	–	2	–	–
EXP	–	–	–	–	–	5	–
$\mathcal{O}(\omega)$	57	31	3	–	–	–	173

LoAT infers non-trivial lower bounds for 393 (78%) of the remaining 507 examples. *Tight* bounds (i.e., the lower and the upper bound coincide) are proved in 341 cases (67%). Whenever an exponential upper bound is proved, LoAT also proves an exponential lower bound (i.e., $\mathsf{rc}_{\mathcal{T}}(n) \in \Omega(k^n)$ for some $k > 1$). In 173 cases, LoAT infers unbounded runtime complexity. In some cases, this is due to non-termination, but for this particular goal, specialized tools are more powerful (e.g., whenever LoAT proves unbounded runtime complexity due to non-termination, the termination analyzer T2 [7] shows non-termination as well). The average runtime of LoAT was 2.4 seconds per example. These results could be improved further by supplementing LoAT with invariant inference as implemented in tools like APRON [20]. For a detailed experimental evaluation of our implementation as well as the sources and a pre-compiled binary of LoAT we refer to [16].

Acknowledgments We thank S. Genaim and J. Böker for discussions and comments.

References

1. Albert, E., Genaim, S., Masud, A.N.: On the inference of resource usage upper and lower bounds. ACM Transactions on Computational Logic 14(3) (2013)
2. Alias, C., Darte, A., Feautrier, P., Gonnord, L.: Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In: Proc. SAS '10. pp. 117–133. LNCS 6337 (2010)

3. Alonso-Blas, D.E., Genaim, S.: On the limits of the classical approach to cost analysis. In: Proc. SAS '12. pp. 405–421. LNCS 7460 (2012)
4. Bagnara, R., Pescetti, A., Zaccagnini, A., Zaffanella, E.: PURRS: Towards computer algebra support for fully automatic worst-case complexity analysis. CoRR abs/cs/0512056 (2005)
5. Benchmark examples, `https://github.com/s-falke/kittel-koat/tree/master/koat-evaluation/examples`
6. Bradley, A.R., Manna, Z., Sipma, H.B.: Linear ranking with reachability. In: Proc. CAV '05. pp. 491–504. LNCS 3576 (2005)
7. Brockschmidt, M., Cook, B., Fuhs, C.: Better termination proving through cooperation. In: Proc. CAV '13. pp. 413–429. LNCS 8044 (2013)
8. Brockschmidt, M., Emmes, F., Falke, S., Fuhs, C., Giesl, J.: Alternating runtime and size complexity analysis of integer programs. In: Proc. TACAS '14. pp. 140–155. LNCS 8413 (2014), full version in ACM Trans. on Prog. Languages and Systems
9. Carbonneaux, Q., Hoffmann, J., Shao, Z.: Compositional certified resource bounds. In: Proc. PLDI '15. pp. 467–478 (2015)
10. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. TACAS '08. pp. 337–340. LNCS 4963 (2008)
11. Debray, S., López-García, P., Hermenegildo, M.V., Lin, N.: Lower bound cost estimation for logic programs. In: Proc. ILPS '97. pp. 291–305 (1997)
12. Falke, S., Kapur, D., Sinz, C.: Termination analysis of imperative programs using bitvector arithmetic. In: Proc. VSTTE '12. pp. 261–277. LNCS 7152 (2012)
13. Farzan, A., Kincaid, Z.: Compositional recurrence analysis. In: Proc. FMCAD '15. pp. 57–64 (2015)
14. Flores-Montoya, A., Hähnle, R.: Resource analysis of complex programs with cost equations. In: Proc. APLAS '14. pp. 275–295. LNCS 8858 (2014)
15. Frohn, F., Giesl, J., Emmes, F., Ströder, T., Aschermann, C., Hensel, J.: Inferring lower bounds for runtime complexity. In: Proc. RTA '15. pp. 334–349. LIPIcs 36 (2015)
16. Frohn, F., Naaf, M., Hensel, J., Brockschmidt, M., Giesl, J.: Proofs and empirical evaluation of "Lower Runtime Bounds for Integer Programs" (2016), available at `http://aprove.informatik.rwth-aachen.de/eval/integerLower/`
17. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: Proc. SAS '06. pp. 144–160. LNCS 4134 (2006)
18. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Proc. RTA '89. pp. 167–177. LNCS 355 (1989)
19. Hoffmann, J., Aehlig, K., Hofmann, M.: Multivariate amortized resource analysis. ACM Transactions on Programming Languages and Systems 34(3) (2012)
20. Jeannet, B., Miné, A.: APRON: A library of numerical abstract domains for static analysis. In: Proc. CAV '09. pp. 661–667. LNCS 5643 (2009)
21. Jeannet, B., Schrammel, P., Sankaranarayanan, S.: Abstract acceleration of general linear loops. ACM SIGPLAN Notices 49(1), 529–540 (2014)
22. Kroening, D., Lewis, M., Weissenbacher, G.: Under-approximating loops in C programs for fast counterexample detection. Form. Meth. Sys. Des. 47(1), 75–92 (2015)
23. LoAT, `https://github.com/aprove-developers/LoAT`
24. Madhukar, K., Wachter, B., Kroening, D., Lewis, M., Srivas, M.K.: Accelerating invariant generation. In: Proc. FMCAD '15. pp. 105–111 (2015)
25. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Proc. VMCAI '04. pp. 239–251. LNCS 2937 (2004)
26. Sinn, M., Zuleger, F., Veith, H.: A simple and scalable static analysis for bound analysis and amortized complexity analysis. In: CAV '14. pp. 745–761. LNCS 8559 (2014)

A Proofs

Theorem 4 (Metering Functions are Under-Estimations). *Let b be a metering function for $t = (\ell, \gamma, \eta, c, \ell)$. Then b* under-estimates *t, i.e., for all $\boldsymbol{v} \in \mathcal{V}al_\ell$ with $\boldsymbol{v} \models \gamma$ there is an evaluation $(\ell, \boldsymbol{v}) \to_t^{\lceil \boldsymbol{v}(b) \rceil} (\ell, \boldsymbol{v}')$ that preserves $\mathcal{TV}_\ell$.*

Proof. For any $\boldsymbol{v} \in \mathcal{V}al_\ell$, let $m_{\boldsymbol{v}} \in \mathbb{N} \cup \{\omega\}$ be the maximal number such that $(\ell, \boldsymbol{v}) \to_t^{m_{\boldsymbol{v}}} (\ell, \boldsymbol{v}')$ is an evaluation that preserves $\mathcal{TV}_\ell$. So the loop t can be executed $m_{\boldsymbol{v}}$ times without changing the temporary variables when starting in the configuration $(\ell, \boldsymbol{v})$. If $m_{\boldsymbol{v}} = \omega$, then the theorem obviously holds for $\boldsymbol{v}$. For all other valuations $\boldsymbol{v}$, we prove it by induction on $m_{\boldsymbol{v}}$.

Clearly, $\boldsymbol{v} \models \gamma$ implies $(\ell, \boldsymbol{v}) \to_t (\ell, \overline{\boldsymbol{v}})$ for the valuation with $\overline{\boldsymbol{v}}(x) = \boldsymbol{v}(\eta(x))$ for $x \in \mathcal{PV}$ and $\overline{\boldsymbol{v}}(x) = \boldsymbol{v}(x)$ for $x \in \mathcal{TV}_\ell$. So $\overline{\boldsymbol{v}}(b) = \boldsymbol{v}(\eta(b)) \geq \boldsymbol{v}(b) - 1$ by (4).

Note that $m_{\boldsymbol{v}} > m_{\overline{\boldsymbol{v}}}$. So if $\overline{\boldsymbol{v}} \models \gamma$, then by the induction hypothesis there exists a valuation $\boldsymbol{v}'$ with $(\ell, \boldsymbol{v}) \to_t (\ell, \overline{\boldsymbol{v}}) \to_t^{\lceil \overline{\boldsymbol{v}}(b) \rceil} (\ell, \boldsymbol{v}')$. This evaluation sequence has length $1 + \lceil \overline{\boldsymbol{v}}(b) \rceil$ and preserves $\mathcal{TV}_\ell$. This proves the theorem, since $1 + \overline{\boldsymbol{v}}(b) \geq \boldsymbol{v}(b)$ implies $\lceil 1 + \overline{\boldsymbol{v}}(b) \rceil = 1 + \lceil \overline{\boldsymbol{v}}(b) \rceil \geq \lceil \boldsymbol{v}(b) \rceil$.

Otherwise, if $\overline{\boldsymbol{v}} \not\models \gamma$, then (3) implies $\overline{\boldsymbol{v}}(b) \leq 0$, and thus $\boldsymbol{v}(b) \leq 1$ and $\lceil \boldsymbol{v}(b) \rceil \leq 1$. Hence, the evaluation sequence $(\ell, \boldsymbol{v}) \to_t (\ell, \overline{\boldsymbol{v}})$ has at least length $\lceil \boldsymbol{v}(b) \rceil$ and preserves $\mathcal{TV}_\ell$, as desired. □

Theorem 7 (Loop Acceleration). *Let $t = (\ell, \gamma, \eta, c, \ell) \in \mathcal{T}$ and let tv be a fresh temporary variable. Moreover, let $\eta_{\mathrm{it}}(x) = \eta^{tv}(x)$ for all $x \in \mathcal{PV}$ and let $c_{\mathrm{it}} \leq \sum_{0 \leq i < tv} \eta^i(c)$. If b under-estimates t, then the processor mapping $\mathcal{T}$ to $\mathcal{T} \cup \{(\ell,\, \gamma \wedge 0 < tv < b+1,\, \eta_{\mathrm{it}},\, c_{\mathrm{it}},\, \ell)\}$ is sound.*

Proof. We show that every evaluation step with $t_{\mathrm{it}} = (\ell,\, \gamma \wedge 0 < tv < b+1,\, \eta_{\mathrm{it}},\, c_{\mathrm{it}},\, \ell)$ can be simulated using a sequence of evaluation steps with t. Thus, let $(\ell, \boldsymbol{v}) \to_{t_{\mathrm{it}},k} (\ell, \boldsymbol{v}')$.

Note that $\boldsymbol{v} \models 0 < tv < b+1$ implies that $0 < \boldsymbol{v}(tv) \leq \lceil \boldsymbol{v}(b) \rceil$. Moreover, we have $\boldsymbol{v} \models \gamma$, and thus by Thm. 4, there is an evaluation

$$(\ell, \boldsymbol{v}_0) \to_{t,k_1} \ldots \to_{t,k_{\boldsymbol{v}(tv)}} (\ell, \boldsymbol{v}_{\boldsymbol{v}(tv)})$$

with $\boldsymbol{v} = \boldsymbol{v}_0$ which preserves $\mathcal{TV}_\ell$. It remains to show that $\boldsymbol{v}_{\boldsymbol{v}(tv)}|_{\mathcal{PV}} = \boldsymbol{v}'|_{\mathcal{PV}}$ and $k \leq \sum_{1 \leq i \leq \boldsymbol{v}(tv)} k_i$. Since η does not modify temporary variables and since $\boldsymbol{v}_i$ is the same as $\boldsymbol{v}$ for all temporary variables, for all $x \in \mathcal{PV}$ and all $0 \leq i \leq \boldsymbol{v}(tv)$ we have $\boldsymbol{v}_i(x) = \boldsymbol{v}(\eta^i(x))$. Thus, $\boldsymbol{v}_{\boldsymbol{v}(tv)}(x) = \boldsymbol{v}(\eta^{\boldsymbol{v}(tv)}(x)) = \boldsymbol{v}(\eta^{tv}(x)) = \boldsymbol{v}(\eta_{\mathrm{it}}(x)) = \boldsymbol{v}'(x)$. Moreover, $k_i = \boldsymbol{v}_{i-1}(c) = \boldsymbol{v}(\eta^{i-1}(c))$, since η does not modify temporary variables and since $\boldsymbol{v}_i$ is the same as $\boldsymbol{v}$ for all temporary variables. Thus, by construction

$$k = \boldsymbol{v}(c_{\mathrm{it}}) \leq \boldsymbol{v}\Big(\sum\nolimits_{0 \leq i < tv} \eta^i(c)\Big) = \sum\nolimits_{1 \leq i \leq \boldsymbol{v}(tv)} \boldsymbol{v}(\eta^{i-1}(c))) = \sum\nolimits_{1 \leq i \leq \boldsymbol{v}(tv)} k_i.$$

□

Theorem 10 (Strengthening). *Let $t = (\ell, \gamma, \eta, c, \ell') \in \mathcal{T}$ and $\varphi \in \mathcal{F}(\mathcal{V}_\ell)$. Then the processor mapping $\mathcal{T}$ to $\mathcal{T} \setminus \{t\} \cup \{(\ell,\, \gamma \wedge \varphi,\, \eta,\, c,\, \ell')\}$ is sound.*

Proof. The processor is sound since $(\ell, \boldsymbol{v}) \to_{(\ell,\gamma\wedge\varphi,\eta,c,\ell'),k} (\ell', \boldsymbol{v}')$ implies $(\ell, \boldsymbol{v}) \to_{t,k} (\ell', \boldsymbol{v}')$ due to $\boldsymbol{v} \models \gamma$. ◻

Theorem 11 (Deletion). *For $t \in \mathcal{T}$, the processor mapping $\mathcal{T}$ to $\mathcal{T} \setminus \{t\}$ is sound.*

Proof. Every evaluation sequence in $\mathcal{T} \setminus \{t\}$ is also an evaluation sequence in $\mathcal{T}$, and thus the processor is trivially sound. ◻

Theorem 12 (Chaining). *Let $t_1 = (\ell_1, \gamma_1, \eta_1, c_1, \ell_2)$ and $t_2 = (\ell_2, \gamma_2, \eta_2, c_2, \ell_3)$ with $t_1, t_2 \in \mathcal{T}$. Let* ren *be an injective function renaming the variables in $\mathcal{TV}_{\ell_2}$ to fresh ones and let $t_{1.2} = (\ell_1,\ \gamma_1 \wedge \mathrm{ren}(\eta_1(\gamma_2)),\ \mathrm{ren} \circ \eta_1 \circ \eta_2,\ c_1 + \mathrm{ren}(\eta_1(c_2)),\ \ell_3)$. Then the processor mapping $\mathcal{T}$ to $\mathcal{T} \cup \{t_{1.2}\}$ is sound. In the new program $\mathcal{T} \cup \{t_{1.2}\}$, the temporary variables of ℓ_1 are defined to be $\mathcal{TV}_{\ell_1} \cup \mathrm{ren}(\mathcal{TV}_{\ell_2})$.*

Proof. We show that every evaluation step with $t_{1.2}$ can be simulated by two evaluation steps with the transitions t_1, t_2 of the same cost. Thus, let $(\ell_1, \boldsymbol{v}_1) \to_{t_{1.2},k} (\ell_3, \boldsymbol{v}_3)$. This implies $\boldsymbol{v}_1 \models \gamma_1 \wedge \mathrm{ren}(\eta_1(\gamma_2))$, $\boldsymbol{v}_3(x) = \boldsymbol{v}_1(\mathrm{ren}(\eta_1(\eta_2(x))))$ for all $x \in \mathcal{PV}$, and $k = \boldsymbol{v}_1(c_1) + \boldsymbol{v}_1(\mathrm{ren}(\eta_1(c_2)))$.

Since $\boldsymbol{v}_1 \models \gamma_1$, we have $(\ell_1, \boldsymbol{v}_1) \to_{t_1,k_1} (\ell_2, \boldsymbol{v}_2)$ for the valuation $\boldsymbol{v}_2$ where $\boldsymbol{v}_2(x) = \boldsymbol{v}_1(\eta_1(x))$ for all $x \in \mathcal{PV}$, $\boldsymbol{v}_2(tv) = \boldsymbol{v}_1(\mathrm{ren}(tv))$ for all $tv \in \mathcal{TV}_{\ell_2}$, and $k_1 = \boldsymbol{v}_1(c_1)$.

Suppose that $\mathcal{PV} = \{x_1, \dots, x_n\}$ and $\mathcal{TV}_{\ell_2} = \{tv_1, \dots, tv_m\}$. Then $\boldsymbol{v}_1 \models \mathrm{ren}(\eta_1(\gamma_2))$ implies that $\gamma_2[x_1/\boldsymbol{v}_1(\eta_1(x_1)), \dots, x_n/\boldsymbol{v}_1(\eta_1(x_n)), tv_1/\boldsymbol{v}_1(\mathrm{ren}(tv_1)), \dots, tv_m/\boldsymbol{v}_1(\mathrm{ren}(tv_m))]$ holds, i.e., $\boldsymbol{v}_2 \models \gamma_2$. To show that $(\ell_2, \boldsymbol{v}_2) \to_{t_2,k_2} (\ell_3, \boldsymbol{v}_3)$ holds for some costs k_2, we need to prove that $\boldsymbol{v}_3(x) = \boldsymbol{v}_2(\eta_2(x))$ for all $x \in \mathcal{PV}$. Indeed, we have

$$\begin{aligned}
&\boldsymbol{v}_2(\eta_2(x))\\
&= (\eta_2(x))\,[\,x_1/\boldsymbol{v}_1(\eta_1(x_1)), \dots, x_n/\boldsymbol{v}_1(\eta_1(x_n)),\\
&\qquad\qquad tv_1/\boldsymbol{v}_1(\mathrm{ren}(tv_1)), \dots, tv_m/\boldsymbol{v}_1(\mathrm{ren}(tv_m))\,]\\
&= \boldsymbol{v}_1(\mathrm{ren}(\eta_1(\eta_2(x))))\\
&= \boldsymbol{v}_3(x)
\end{aligned}$$

Moreover, we have $k_2 = \boldsymbol{v}_2(c_2) = \boldsymbol{v}_1(\mathrm{ren}(\eta_1(c_2)))$, which implies $k = k_1 + k_2$. ◻

Theorem 16 (Asymptotic Bounds for Simplified Programs). *Given a transition t of a simplified program $\mathcal{T}$ with $\mathsf{guard}(t) = a_1 > 0 \wedge \cdots \wedge a_k > 0$, let the family $\boldsymbol{v}_n$ be a solution of an initial limit problem $\{a_1^{\bullet_1}, \dots, a_k^{\bullet_k}\}$ with $\bullet_i \in \{+, +_!\}$ for all $1 \le i \le k$. Then $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(\boldsymbol{v}_n(\mathsf{cost}(t)))$.*

Proof. Since $\boldsymbol{v}_n$ is a solution of $\{a_1^{\bullet_1}, \dots, a_k^{\bullet_k}\}$, there is an $n_0 \in \mathbb{N}$ such that for all $n \ge n_0$, we have $\boldsymbol{v}_n \models a_1 > 0 \wedge \cdots \wedge a_k > 0$, i.e., $\boldsymbol{v}_n \models \mathsf{guard}(t)$. Hence, for all $n \ge n_0$, we obtain

$$\begin{aligned}
\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) &\ge \mathsf{dh}_{\mathcal{T}}(\boldsymbol{v}_n|_{\mathcal{PV}}) && \text{as } |\,\boldsymbol{v}_n|_{\mathcal{PV}}\,| = |\boldsymbol{v}_n(x)|\\
&\ge \boldsymbol{v}_n(\mathsf{cost}(t)) && \text{as } \boldsymbol{v}_n \models \mathsf{guard}(t)
\end{aligned}$$

This implies $\mathsf{rc}_{\mathcal{T}}(|\boldsymbol{v}_n|) \in \Omega(\boldsymbol{v}_n(\mathsf{cost}(t)))$. ◻

Lemma 17 (Bounds for Function Composition). *Let $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $g : \mathbb{N} \to \mathbb{N}$ where $g(n) \in \mathcal{O}(n^m)$ for some $m \in \mathbb{N} \setminus \{0\}$. Moreover, let $f(n)$ be weakly and let $g(n)$ be strictly monotonically increasing for large enough n.*

- *If $f(g(n)) \in \Omega(n^k)$ with $k \in \mathbb{N}$, then $f(n) \in \Omega(n^{\frac{k}{m}})$.*
- *If $f(g(n)) \in \Omega(k^n)$ with $k \in \mathbb{N}$, then $f(n) \in \Omega(k^{\sqrt[m]{n}})$.*

Proof. We adapt the corresponding proof of [15, Lemma 18]. For any $n_0 \in \mathbb{N}$, let $\mathbb{N}_{\geq n_0} = \{n \in \mathbb{N} \mid n \geq n_0\}$. Moreover, for any (total) function $h : M \to \mathbb{N}_{\geq n_0}$ with $M \subseteq \mathbb{N}$ where M is infinite, we define $\lfloor h \rfloor(n) : \mathbb{N}_{\geq \min(M)} \to \mathbb{N}_{\geq n_0}$ and $\lceil h \rceil(n) : \mathbb{N}_{\geq \min(M)} \to \mathbb{N}_{\geq n_0}$ by:

$$\lfloor h \rfloor(n) = h(\,\max\{n' \mid n' \in M, n' \leq n\}\,) \qquad \lceil h \rceil(n) = h(\,\min\{n' \mid n' \in M, n' \geq n\}\,)$$

Note that infinity of h's domain M ensures that there is always an $n' \in M$ with $n' \geq n$.

To prove the lemma, we first show that if $h : M \to \mathbb{N}_{\geq n_0}$ is strictly monotonically increasing and surjective then

$$\lfloor h \rfloor(n) \in \{\lceil h \rceil(n), \lceil h \rceil(n) - 1\} \quad \text{for all } n \in \mathbb{N}_{\geq \min(M)} \tag{6}$$

To prove (6), let $n \in \mathbb{N}_{\geq \min(M)}$. If $n \in M$, then clearly $\lfloor h \rfloor(n) = \lceil h \rceil(n)$. If $n \notin M$, then let $n' = \max\{n' \mid n' \in M, n' < n\}$ and $n'' = \min\{n'' \mid n'' \in M, n'' > n\}$. Thus, $n' < n < n''$. Strict monotonicity of h implies $h(n') < h(n'')$. Assume that $h(n'') - h(n') > 1$. Then by surjectivity of h, there is an $\hat{n} \in M$ with $h(\hat{n}) = h(n') + 1$ and thus $h(n') < h(\hat{n}) < h(n'')$. By strict monotonicity of h, we obtain $n' < \hat{n} < n''$. Since $n \notin M$ and $\hat{n} \in M$ implies $n \neq \hat{n}$, we either have $\hat{n} < n$ which contradicts $n' = \max\{n' \mid n' \in M, n' < n\}$ or $\hat{n} > n$ which contradicts $n'' = \min\{n'' \mid n'' \in M, n'' > n\}$. Hence, $\lfloor h \rfloor(n) = h(n') = h(n'') - 1 = \lceil h \rceil(n) - 1$, which proves (6).

Now we prove Lemma 17. Here, $g(n) \in \mathcal{O}(n^m)$ and $f(g(n)) \in \Omega(n^k)$ (the case $f(g(n)) \in \Omega(k^n)$ is analogous) imply

$$\exists n_0, c, c' > 0.\ \forall n \in \mathbb{N}_{\geq n_0}. \quad g(n) \leq c \cdot n^m \ \wedge\ c' \cdot n^k \leq f(g(n)).$$

We can choose n_0 large enough such that $f(n)$ is weakly and $g(n)$ is strictly monotonically increasing for all $n \geq n_0$. Let $M = \{g(n) \mid n \geq n_0\}$ and let $g^{-1} : M \to \mathbb{N}_{\geq n_0}$ be the function such that $g(g^{-1}(n)) = n$. By instantiating n with $g^{-1}(n)$, we obtain

$$\exists n_0, c, c' > 0.\ \forall n \in M.\ g(g^{-1}(n)) \leq c \cdot (g^{-1}(n))^m \ \wedge\ c' \cdot (g^{-1}(n))^k \leq f(g(g^{-1}(n)))$$

which simplifies to

$$\exists n_0, c, c' > 0.\ \forall n \in M. \quad n \leq c \cdot (g^{-1}(n))^m \ \wedge\ c' \cdot (g^{-1}(n))^k \leq f(n).$$

When dividing by c and building the m-th root on both sides of the first inequality, we get

$$\exists n_0, c, c' > 0.\ \forall n \in M. \quad \sqrt[m]{\frac{n}{c}} \leq g^{-1}(n) \ \wedge\ c' \cdot (g^{-1}(n))^k \leq f(n).$$

By monotonicity of $\sqrt[m]{\frac{n}{c}}$ and $f(n)$, this implies

$$\exists n_0, c, c' > 0.\ \forall n \in \mathbb{N}_{\geq g(n_0)}. \quad \sqrt[m]{\frac{n}{c}} \leq \lceil g^{-1} \rceil(n) \ \wedge\ c' \cdot (\lfloor g^{-1} \rfloor(n))^k \leq f(n).$$

Note that g is total and hence, $g^{-1} : M \to \mathbb{N}_{\geq n_0}$ is surjective. Moreover, by strict monotonicity of g, M is infinite and g^{-1} is also strictly monotonically increasing. Hence, by (6) we get $\lceil g^{-1} \rceil(n) \leq \lfloor g^{-1} \rfloor(n) + 1$ for all $n \in \mathbb{N}_{\geq g(n_0)}$. Hence,

$$\exists n_0, c, c' > 0.\ \forall n \in \mathbb{N}_{\geq g(n_0)}. \quad \sqrt[m]{\frac{n}{c}} - 1 \leq \lfloor g^{-1} \rfloor(n) \ \wedge\ c' \cdot (\lfloor g^{-1} \rfloor(n))^k \leq f(n)$$

which implies

$$\exists n_0, c, c' > 0.\ \forall n \in \mathbb{N}_{\geq g(n_0)}. \quad c' \cdot (\sqrt[m]{\frac{n}{c}} - 1)^k \leq f(n).$$

Therefore, $\exists c > 0.\ f(n) \in \Omega\left((\sqrt[m]{\frac{n}{c}} - 1)^k\right)$ and thus, $f(n) \in \Omega(n^{\frac{k}{m}})$. □

Lemma 20 (Solving Trivial Limit Problems). *Let S be a trivial limit problem. Then $\boldsymbol{v}_n^S$ is a solution of S where for all $n \in \mathbb{N}$, $\boldsymbol{v}_n^S$ is defined as follows:*

$$\begin{array}{lll} \boldsymbol{v}_n^S(x) = n, \text{ if } x^+ \in S & \boldsymbol{v}_n^S(x) = 1, \text{ if } x^{+_!} \in S & \boldsymbol{v}_n^S(x) = 0, \text{ otherwise} \\ \boldsymbol{v}_n^S(x) = -n, \text{ if } x^- \in S & \boldsymbol{v}_n^S(x) = -1, \text{ if } x^{-_!} \in S & \end{array}$$

Proof. If $x^+ \in S$ (resp. $x^- \in S$), then $\boldsymbol{v}_n^S(x) = n$ (resp. $\boldsymbol{v}_n^S(x) = -n$) and thus, $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = \lim_{n \mapsto \omega} n = \omega$ (resp. $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = \lim_{n \mapsto \omega} -n = -\omega$), i.e., $\lambda n.\ \boldsymbol{v}_n(x)$ satisfies $+$ (resp. $-$). If $x^{+_!} \in S$ (resp. $x^{-_!} \in S$), then $\boldsymbol{v}_n^S(x) = 1$ (resp. $\boldsymbol{v}_n^S(x) = -1$). Thus, $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = 1$ (resp. $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = -1$), i.e., $\lambda n.\ \boldsymbol{v}_n(x)$ satisfies $+_!$ (resp. $-_!$). Hence, $\boldsymbol{v}_n^S$ is a solution of S. □

Theorem 27 (Correctness of $\leadsto$). *If $S \leadsto^\sigma S'$ and the family $\boldsymbol{v}_n$ is a solution of S', then $\boldsymbol{v}_n \circ \sigma$ is a solution of S.*

Proof. We start with the case where the step from S to S' was done by Def. 21 (A). Since $\boldsymbol{v}_n$ is a solution for S', it is a solution for $a_1^{\bullet_1}$ and $a_2^{\bullet_2}$, where $(\bullet_1, \bullet_2)$ is an increasing (resp. decreasing, positive, or negative) limit vector for f. As $\boldsymbol{v}_n$ is a solution for both $a_i^{\bullet_i}$, the function $\lambda n.\ \boldsymbol{v}_n(a_i)$ satisfies $\bullet_i$. By the definition of limit vectors, this implies that $\lambda n.\ f(\boldsymbol{v}_n(a_1), \boldsymbol{v}_n(a_2)) = \lambda n.\ \boldsymbol{v}_n(f(a_1, a_2))$ satisfies $\bullet$. Thus, $\boldsymbol{v}_n$ is a solution for $f(a_1, a_2)^\bullet$.

If the step from S to S' was done by Def. 21 (B), then every solution $\boldsymbol{v}_n$ for S' is also a solution for S, since $\boldsymbol{v}_n(m) = m$ holds for any $m \in \mathbb{Z}$.

If the step from S to S' was done by Def. 23 (C), then let $\boldsymbol{v}_n$ be a solution for $S' = S\sigma$. Then for all terms a, we have that $\boldsymbol{v}_n(a\sigma) = (\boldsymbol{v}_n \circ \sigma)(a)$. So if $\boldsymbol{v}_n$ is a solution for $(a\sigma)^\bullet$, then $\boldsymbol{v}_n \circ \sigma$ is a solution for $a^\bullet$.

If the step from S to S' was done by Def. 26 (D), then let $\boldsymbol{v}_n$ be a solution for $a^\bullet$. Since the only variable of the polynomial a is x, we must have $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = \omega$ or $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = -\omega$. W.l.o.g, let $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = \omega$ and $\bullet = +$ (the other cases work analogously). Then $\lim_{n \mapsto \omega} \boldsymbol{v}_n(a) = \omega$ implies $\lim_{x \mapsto \omega} a = \omega$. Since the degree of a is greater than than the degree of b, this means $\lim_{x \mapsto \omega} a \pm b = \omega$ and hence $\lim_{n \mapsto \omega} \boldsymbol{v}_n(a \pm b) = \omega$.

For Def. 26 (E), the proof is analogous. Here for large enough n, $\boldsymbol{v}_n(a^e)$ is an exponential function with a base > 1. Since $\boldsymbol{v}_n$ is a solution for e^+, we again have $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = \omega$ or $\lim_{n \mapsto \omega} \boldsymbol{v}_n(x) = -\omega$. Thus $\boldsymbol{v}_n(a^e)$ is an exponential function which grows faster than $\boldsymbol{v}_n(b)$ for $n \mapsto \omega$. Hence, we obtain $\lim_{n \mapsto \omega} \boldsymbol{v}_n(a^e \pm b) = \omega$. $\square$

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

http://aib.informatik.rwth-aachen.de/

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: biblio@informatik.rwth-aachen.de

2013-01 * Fachgruppe Informatik: Annual Report 2013
2013-02 Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssysteme in Klein- und mittelständischen Unternehmen
2013-03 Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM
2013-04 Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries
2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013
2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation
2013-07 André Stollenwerk: Ein modellbasiertes Sicherheitskonzept für die extrakorporale Lungenunterstützung
2013-08 Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika Ábrahám: On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers
2013-10 Joost-Pieter Katoen, Thomas Noll, Thomas Santen, Dirk Seifert, and Hao Wu: Performance Analysis of Computing Servers using Stochastic Petri Nets and Markov Automata
2013-12 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl: Alternating Runtime and Size Complexity Analysis of Integer Programs
2013-13 Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle: SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators
2013-14 Jörg Brauer: Automatic Abstraction for Bit-Vectors using Decision Procedures
2013-16 Carsten Otto: Java Program Analysis by Symbolic Execution
2013-19 Florian Schmidt, David Orlea, and Klaus Wehrle: Support for error tolerance in the Real-Time Transport Protocol

2013-20 Jacob Palczynski: Time-Continuous Behaviour Comparison Based on Abstract Models
2014-01 * Fachgruppe Informatik: Annual Report 2014
2014-02 Daniel Merschen: Integration und Analyse von Artefakten in der modellbasierten Entwicklung eingebetteter Software
2014-03 Uwe Naumann, Klaus Leppkes, and Johannes Lotz: dco/c++ User Guide
2014-04 Namit Chaturvedi: Languages of Infinite Traces and Deterministic Asynchronous Automata
2014-05 Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp: Automated Termination Analysis for Programs with Pointer Arithmetic
2014-06 Esther Horbert, Germán Martín García, Simone Frintrop, and Bastian Leibe: Sequence Level Salient Object Proposals for Generic Object Detection in Video
2014-07 Niloofar Safiran, Johannes Lotz, and Uwe Naumann: Algorithmic Differentiation of Numerical Methods: Second-Order Tangent and Adjoint Solvers for Systems of Parametrized Nonlinear Equations
2014-08 Christina Jansen, Florian Göbe, and Thomas Noll: Generating Inductive Predicates for Symbolic Execution of Pointer-Manipulating Programs
2014-09 Thomas Ströder and Terrance Swift (Editors): Proceedings of the International Joint Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments 2014
2014-14 Florian Schmidt, Matteo Ceriotti, Niklas Hauser, and Klaus Wehrle: HotBox: Testing Temperature Effects in Sensor Networks
2014-15 Dominique Gückel: Synthesis of State Space Generators for Model Checking Microcontroller Code
2014-16 Hongfei Fu: Verifying Probabilistic Systems: New Algorithms and Complexity Results
2015-01 * Fachgruppe Informatik: Annual Report 2015
2015-02 Dominik Franke: Testing Life Cycle-related Properties of Mobile Applications
2015-05 Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder: Inferring Lower Bounds for Runtime Complexity
2015-06 Thomas Ströder and Wolfgang Thomas (Editors): Proceedings of the Young Researchers' Conference "Frontiers of Formal Methods"
2015-07 Hilal Diab: Experimental Validation and Mathematical Analysis of Cooperative Vehicles in a Platoon
2015-08 Mathias Pelka, Jó Agila Bitsch, Horst Hellbrück, and Klaus Wehrle (Editors): Proceedings of the 1st KuVS Expert Talk on Localization
2015-09 Xin Chen: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models
2015-11 Stefan Wüller, Marián Kühnel, and Ulrike Meyer: Information Hiding in the Public RSA Modulus
2015-12 Christoph Matheja, Christina Jansen, and Thomas Noll: Tree-like Grammars and Separation Logic

2015-13 Andreas Polzer: Ansatz zur variantenreichen und modellbasierten Entwicklung von eingebetteten Systemen unter Berücksichtigung regelungs- und softwaretechnischer Anforderungen

2015-14 Niloofar Safiran and Uwe Naumann: Symbolic vs. Algorithmic Differentiation of GSL Integration Routines

2016-01 * Fachgruppe Informatik: Annual Report 2016

2016-02 Ibtissem Ben Makhlouf: Comparative Evaluation and Improvement of Computational Approaches to Reachability Analysis of Linear Hybrid Systems

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.