Proving Non-Termination by Acceleration Driven Clause Learning (Short Paper)[⋆]

Florian Frohn(✉) and Jürgen Giesl(✉)

LuFG Informatik 2, RWTH Aachen University, Aachen, Germany

Abstract. We recently proposed *Acceleration Driven Clause Learning* (ADCL), a novel calculus to analyze satisfiability of *Constrained Horn Clauses* (CHCs). Here, we adapt ADCL to transition systems and introduce ADCL-NT, a variant for disproving termination. We implemented ADCL-NT in our tool LoAT and evaluate it against the state of the art.

1 Introduction

Termination is one of the most important properties of programs, and thus termination analysis is a very active field of research. Here, we are concerned with *dis*proving termination of *transition systems* (TSs), a popular intermediate representation for verification of programs written in more expressive languages.

Example 1. Consider the following TS $\mathcal{T}$ with entry-point init and two further *locations* ℓ_1, ℓ_2 over the variables x, y, z, where x', y', z' represent the values of x, y, z *after* applying a transition, and $\overset{=}{x}$, x++, and x-- abbreviate $x' = x$, $x' = x+1$, and $x' = x - 1$. The first two transitions are a variant[1] of `chc-LIA-Lin_052` from the *CHC Competition '22* [7] and the last two are a variant[2] of `flip2_rec.jar-obl-8` from the *Termination and Complexity Competition (TermComp)* [21].

$$\mathsf{init} \to \ell_1 \, [\![x' \leq 0 \land z' \geq 5000 \land y' \leq z']\!] \qquad (\tau_\mathsf{i})$$

$$\ell_1 \to \ell_1 \, [\![y \leq 2 \cdot z \land x\text{++} \land ((x < z \land \overset{=}{y}) \lor (x \geq z \land y\text{++})) \land \overset{=}{z}]\!] \qquad (\tau_{\ell_1})$$

$$\ell_1 \to \ell_2 \, [\![x = y \land x > 2 \cdot z \land \overset{=}{x} \land \overset{=}{y}]\!] \qquad (\tau_{\ell_1 \to \ell_2})$$

$$\ell_2 \to \ell_2 \, [\![x = y \land x > 0 \land \overset{=}{x} \land y\text{--}]\!] \qquad (\tau_{\ell_2}^{=})$$

$$\ell_2 \to \ell_2 \, [\![x > 0 \land y > 0 \land x' = y \land ((x > y \land y' = x) \lor (x < y \land \overset{=}{y}))]\!] \qquad (\tau_{\ell_2}^{\neq})$$

At ℓ_1, $\mathcal{T}$ operates in two "phases": First, just x is incremented until x reaches z (1st disjunct of τ_{ℓ_1}). Then, x and y are incremented until y reaches $2 \cdot z + 1$ (2nd disjunct of τ_{ℓ_1}). If $x = y = c$ holds for some $c > 1$ at that point (which is the case if $x \leq y = z$ holds initially), then the execution can continue at ℓ_2 as follows:

⋆ funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2)

1 We generalized the example to make it more interesting, and we added the condition $y \leq 2 \cdot z$ to enforce termination of τ_{ℓ_1}.

2 We combined the transitions for the cases $x > y$ and $x < y$ into the equivalent transition $\tau_{\ell_2}^{\neq}$ to demonstrate how our approach can deal with disjunctions in conditions.

$$\ell_2(c, c, c_z) \longrightarrow_{\tau^{=}_{\ell_2}} \ell_2(c, c-1, c_z) \longrightarrow_{\tau^{\neq}_{\ell_2}} \ell_2(c-1, c, c_z) \longrightarrow_{\tau^{\neq}_{\ell_2}} \ell_2(c, c, c_z) \longrightarrow_{\tau^{=}_{\ell_2}} \ldots$$

Here, $\ell_2(c, c, c_z)$ means that the current location is ℓ_2 and the values of x, y, z are c, c, c_z. The 1st and 2nd step with $\tau^{\neq}_{\ell_2}$ satisfy the 1st $(x > y \wedge \ldots)$ and 2nd $(x < y \wedge \ldots)$ disjunct of $\tau^{\neq}_{\ell_2}$'s condition, respectively. Thus, $\mathcal{T}$ does not terminate.

Ex. 1 is challenging for state-of-the-art tools for several reasons. First, more than 5000 steps are required to reach ℓ_2, so reachability of ℓ_2 is difficult to prove for approaches that unroll the transition relation or use other variants of iterative deepening. Thus, `chc-LIA-Lin_052` is beyond the capabilities of most other state-of-the-art tools for proving reachability.

Second, the pattern "$\tau^{=}_{\ell_2}$, 1st disjunct of $\tau^{\neq}_{\ell_2}$, 2nd disjunct of $\tau^{\neq}_{\ell_2}$" must be found to prove non-termination. Therefore, `flip2_rec.jar-obl-8` (which does not use disjunctions) cannot be solved by other state-of-the-art termination tools.

Third, Ex. 1 contains disjunctions, which are not supported by many termination tools. Presumably, the reason is that most techniques for (dis)proving termination of loops are restricted to conjunctions (e.g., due to the use of templates and Farkas' Lemma). While disjunctions can be avoided by splitting disjunctive transitions according to the DNF of their conditions, this leads to an exponential blow-up in the number of transitions.

We present an approach that can prove non-termination of systems like Ex. 1 automatically. To this end, we tightly integrate non-termination techniques into our recent *Acceleration Driven Clause Learning (ADCL)* calculus [16], which has originally been designed for CHCs, but it can also be used to analyze TSs.

Due to the use of acceleration techniques that compute the transitive closure of recursive transitions, ADCL finds long witnesses of reachability automatically. If acceleration techniques cannot be applied, it unrolls the transition relation, so it can easily detect complex patterns of transitions that admit non-terminating runs. Finally, ADCL reduces reasoning about disjunctions to reasoning about conjunctions by considering conjunctive variants of disjunctive transitions. Thus, combining ADCL with non-termination techniques for conjunctive transitions allows for disproving termination of TSs with complex Boolean structure.

After introducing preliminaries in Sect. 2, Sect. 3 presents a straightforward adaption of ADCL to TSs. Sect. 4 introduces our main contribution: ADCL-NT, a variant of ADCL for proving non-termination. Finally, in Sect. 5, we discuss related work and demonstrate the power of our approach by comparing it with other state-of-the-art tools. All proofs can be found in [19].

2 Preliminaries

We assume familiarity with basics from many-sorted first-order logic. $\mathcal{V}$ is a countably infinite set of variables and $\mathcal{A}$ is a first-order theory over a k-sorted signature $\Sigma_{\mathcal{A}}$ with carrier $\mathcal{C}_{\mathcal{A}} = (\mathcal{C}_{\mathcal{A},1}, \ldots, \mathcal{C}_{\mathcal{A},k})$. $\mathsf{QF}(\Sigma_{\mathcal{A}})$ is the set of all quantifier-free first-order formulas over $\Sigma_{\mathcal{A}}$, which are w.l.o.g. assumed to be in negation normal form, and $\mathsf{QF}_{\wedge}(\Sigma_{\mathcal{A}})$ only contains conjunctions of $\Sigma_{\mathcal{A}}$-literals. Given a first-order formula η over $\Sigma_{\mathcal{A}}$, σ is a *model* of η (written $\sigma \models_{\mathcal{A}} \eta$) if it is a model of $\mathcal{A}$ with carrier $\mathcal{C}_{\mathcal{A}}$, extended with interpretations for $\mathcal{V}$ such that η is satisfied. As usual, $\models_{\mathcal{A}} \eta$ means that η is valid, and $\eta \equiv_{\mathcal{A}} \eta'$ means $\models_{\mathcal{A}} \eta \iff \eta'$.

We write $\vec{x}$ for sequences and x_i is the i^{th} element of $\vec{x}$. We use "::" for concatenation of sequences, where we identify sequences of length 1 with their elements, so we may write, e.g., $x :: xs$ instead of $[x] :: xs$.

Transition Systems Let $d \in \mathbb{N}$ be fixed, and let $\vec{x}, \vec{x}' \in \mathcal{V}^d$ be disjoint vectors of pairwise different variables. Each $\psi \in \mathsf{QF}(\Sigma_{\mathcal{A}})$ induces a relation $\longrightarrow_{\psi}$ on $\mathcal{C}_{\mathcal{A}}^d$ where $\vec{s} \longrightarrow_{\psi} \vec{t}$ iff $\psi[\vec{x}/\vec{s}, \vec{x}'/\vec{t}]$ is satisfiable. So for the condition $\psi := (x = y \wedge x > 0 \wedge \overset{=}{x} \wedge y{-}{-})$ of $\tau_{\overset{=}{\ell_2}}$, we have $(4,4,4) \longrightarrow_{\psi} (4,3,7)$. $\mathcal{L} \supseteq \{\mathsf{init}, \mathsf{err}\}$ is a finite set of *locations*. A *configuration* is a pair $(\ell, \vec{s}) \in \mathcal{L} \times \mathcal{C}_{\mathcal{A}}^d$, written $\ell(\vec{s})$. A *transition* is a triple $\tau = (\ell, \psi, \ell') \in \mathcal{L} \times \mathsf{QF}(\Sigma_{\mathcal{A}}) \times \mathcal{L}$, written $\ell \to \ell' \llbracket \psi \rrbracket$, and its *condition* is $\mathsf{cond}(\tau) := \psi$. W.l.o.g., we assume $\ell \neq \mathsf{err}$ and $\ell' \neq \mathsf{init}$. Then τ induces a relation $\longrightarrow_{\tau}$ on configurations where $\mathfrak{s} \longrightarrow_{\tau} \mathfrak{t}$ iff $\mathfrak{s} = \ell(\vec{s}), \mathfrak{t} = \ell'(\vec{t})$, and $\vec{s} \longrightarrow_{\psi} \vec{t}$. So, e.g., $\ell_2(4,4,4) \longrightarrow_{\tau_{\overset{=}{\ell_2}}} \ell_2(4,3,7)$. We call τ *recursive* if $\ell = \ell'$, *conjunctive* if $\psi \in \mathsf{QF}_{\wedge}(\Sigma_{\mathcal{A}})$, *initial* if $\ell = \mathsf{init}$, and *safe* if $\ell' \neq \mathsf{err}$. Moreover, we define $(\ell \to \ell' \llbracket \psi \rrbracket)|_{\psi'} := \ell \to \ell' \llbracket \psi' \rrbracket$. A *transition system* (TS) $\mathcal{T}$ is a finite set of transitions, and it induces the relation $\longrightarrow_{\mathcal{T}} := \bigcup_{\tau \in \mathcal{T}} \longrightarrow_{\tau}$.

Chaining $\tau = \ell_s \to \ell_t \llbracket \psi \rrbracket$ and $\tau' = \ell_s' \to \ell_t' \llbracket \psi' \rrbracket$ yields $\mathsf{chain}(\tau, \tau') := (\ell_s \to \ell_t' \llbracket \psi_c \rrbracket)$ where $\psi_c := \psi[\vec{x}'/\vec{x}''] \wedge \psi'[\vec{x}/\vec{x}'']$ for fresh $\vec{x}'' \in \mathcal{V}^d$ if $\ell_t = \ell_s'$, and $\psi_c := \bot$ (meaning *false*) if $\ell_t \neq \ell_s'$. So $\longrightarrow_{\mathsf{chain}(\tau,\tau')} = \longrightarrow_{\tau} \circ \longrightarrow_{\tau'}$, and $\mathsf{chain}(\tau_{\ell_1 \to \ell_2}, \tau_{\overset{=}{\ell_2}}) = \ell_1 \to \ell_2 \llbracket \psi \rrbracket$ where $\psi \equiv_{\mathcal{A}} (x = y \wedge x > 2 \cdot z \wedge x > 0 \wedge \overset{=}{x} \wedge y{-}{-})$. For non-empty, finite sequences of transitions we define $\mathsf{chain}([\tau]) := \tau$ and $\mathsf{chain}([\tau_1, \tau_2] :: \vec{\tau}) := \mathsf{chain}(\mathsf{chain}(\tau_1, \tau_2) :: \vec{\tau})$. We lift notations for transitions to finite sequences via chaining. So $\mathsf{cond}(\vec{\tau}) := \mathsf{cond}(\mathsf{chain}(\vec{\tau}))$, $\vec{\tau}$ is *recursive* if $\mathsf{chain}(\vec{\tau})$ is recursive, $\longrightarrow_{\vec{\tau}} = \longrightarrow_{\mathsf{chain}(\vec{\tau})}$, etc. If τ is initial and $\mathsf{cond}(\tau :: \vec{\tau}) \not\equiv_{\mathcal{A}} \bot$, then $(\tau :: \vec{\tau}) \in \mathcal{T}^{+}$ is a *finite run*. $\mathcal{T}$ is safe if every finite run is safe. If every finite prefix of $\vec{\tau} \in \mathcal{T}^{\omega}$ is a finite run, then $\vec{\tau}$ is an *infinite run*. If no infinite run exists, then $\mathcal{T}$ is *terminating*.

Acceleration *Acceleration techniques* compute the transitive closure of relations. In the following definition, we only consider relations defined by conjunctive formulas, since many existing acceleration techniques do not support disjunctions [4], or have to resort to approximations in the presence of disjunctions [13].

Definition 2 (Acceleration). *An* acceleration technique *is a function* $\mathsf{accel} : \mathsf{QF}_{\wedge}(\Sigma_{\mathcal{A}}) \mapsto \mathsf{QF}_{\wedge}(\Sigma_{\mathcal{A}'})$ *such that* $\longrightarrow_{\psi}^{+} = \longrightarrow_{\mathsf{accel}(\psi)}$*, where* $\mathcal{A}'$ *is a first-order theory. For recursive conjunctive transitions* τ*, we define* $\mathsf{accel}(\tau) := \tau|_{\mathsf{accel}(\mathsf{cond}(\tau))}$.

So we clearly have $\longrightarrow_{\tau}^{+} = \longrightarrow_{\mathsf{accel}(\tau)}$. Note that most theories are not "closed under acceleration". E.g., accelerating the Presburger formula $x_1' = x_1 + x_2 \wedge \overset{=}{x_2}$ yields the non-linear formula $n > 0 \wedge x_1' = x_1 + n \cdot x_2 \wedge \overset{=}{x_2}$. If neither $\mathbb{N}$ nor $\mathbb{Z}$ are contained in $\mathcal{C}_{\mathcal{A}}$, then an additional sort for the range of n is required in the formula that results from applying accel. Hence, Def. 2 allows $\mathcal{A}' \neq \mathcal{A}$.

3 ADCL for Transition Systems

We originally proposed the ADCL calculus to analyze satisfiability of linear *Constrained Horn Clauses* (CHCs) [16]. Here, we rephrase it for TSs, and in Sect. 4, we modify it for proving non-termination. The adaption to TSs is straightforward as TSs can be transformed into equivalent linear CHCs and vice versa (see, e.g., [10]).

To bridge the gap between transitions τ where $\mathsf{cond}(\tau) \in \mathsf{QF}(\Sigma_{\mathcal{A}})$ and acceleration techniques for formulas from $\mathsf{QF}_{\land}(\Sigma_{\mathcal{A}})$, ADCL uses *syntactic implicants*.

Definition 3 (Syntactic Implicants [16, Def. 6]). *If $\psi \in \mathsf{QF}(\Sigma_{\mathcal{A}})$, then:*

$$\begin{aligned}
\mathsf{sip}(\psi,\sigma) &:= \bigwedge\{\pi \textit{ is a literal of } \psi \mid \sigma \models_{\mathcal{A}} \pi\} && \textit{if } \sigma \models_{\mathcal{A}} \psi\\
\mathsf{sip}(\psi) &:= \{\mathsf{sip}(\psi,\sigma) \mid \sigma \models_{\mathcal{A}} \psi\}\\
\mathsf{sip}(\tau) &:= \{\tau|_{\psi} \mid \psi \in \mathsf{sip}(\mathsf{cond}(\tau))\} && \textit{for transitions } \tau\\
\mathsf{sip}(\mathcal{T}) &:= \bigcup_{\tau\in\mathcal{T}} \mathsf{sip}(\tau) && \textit{for TSs } \mathcal{T}
\end{aligned}$$

Here, sip *abbreviates* syntactic implicant projection.

As $\mathsf{sip}(\psi,\sigma)$ is restricted to literals from ψ, $\mathsf{sip}(\psi)$ is finite. Syntactic implicants ignore the semantics of literals. So we have, e.g., $(X > 1) \notin \mathsf{sip}(X > 0 \land X > 1) = \{X > 0 \land X > 1\}$. It is easy to show $\psi \equiv_{\mathcal{A}} \bigvee \mathsf{sip}(\psi)$, and thus $\longrightarrow_{\mathcal{T}} = \longrightarrow_{\mathsf{sip}(\mathcal{T})}$.

Since $\mathsf{sip}(\tau)$ is worst-case exponential in the size of $\mathsf{cond}(\tau)$, we do not compute it explicitly. Instead, ADCL constructs a run $\vec{\tau}$ step by step, and to perform a step with τ, it searches for a model σ of $\mathsf{cond}(\vec{\tau} :: \tau)$. If such a model exists, it appends $\tau|_{\mathsf{sip}(\mathsf{cond}(\tau),\sigma)}$ to $\vec{\tau}$. This corresponds to a step with a conjunctive variant of τ whose condition is satisfied by σ. In other words, our calculus constructs $\mathsf{sip}(\mathsf{cond}(\tau),\sigma)$ "on the fly" when performing a step with τ, where $\sigma \models_{\mathcal{A}} \mathsf{cond}(\vec{\tau} :: \tau)$

The core idea of ADCL is to learn new, *non-redundant* transitions via acceleration. Essentially, a transition is redundant if its transition relation is a subset of another transition's relation. Thus, redundant transitions are not useful for (dis-)proving safety.

Definition 4 (Redundancy, [16, Def. 8]). *A transition τ is* (strictly) redundant *w.r.t. τ', denoted $\tau \sqsubseteq \tau'$ ($\tau \sqsubset \tau'$) if $\longrightarrow_{\tau} \subseteq \longrightarrow_{\tau'}$ ($\longrightarrow_{\tau} \subset \longrightarrow_{\tau'}$). For a TS $\mathcal{T}$, we have $\tau \sqsubseteq \mathcal{T}$ ($\tau \sqsubset \mathcal{T}$) if $\tau \sqsubseteq \tau'$ ($\tau \sqsubset \tau'$) for some $\tau' \in \mathcal{T}$.*

In the sequel, we assume oracles for redundancy, satisfiability of $\mathsf{QF}(\Sigma_{\mathcal{A}})$-formulas, and acceleration. In practice, we use incomplete techniques instead (see Sect. 5).

From now on, let $\mathcal{T}$ be the TS that is being analyzed with ADCL. A *state* of ADCL consists of a TS $\mathcal{S}$ that augments $\mathcal{T}$ with *learned transitions*, a run $\vec{\tau}$ of $\mathcal{S}$ called the *trace*, and a sequence of sets of *blocking transitions* $[B_i]_{i=0}^{k}$, where transitions that are redundant w.r.t. B_k must not be appended to the trace.

The following definition introduces the ADCL calculus. It extends the trace step by step (using the rule Step, which performs an evaluation step with a transition) and learns new transitions via acceleration (Accelerate) whenever a suffix of the trace is recursive. To avoid non-terminating ADCL-derivations, our notion of *redundancy* from Def. 4 is used to backtrack whenever a suffix of the trace corresponds to a special case of another (learned) transition (Covered). Moreover, Backtrack is used whenever a run cannot be continued. A more detailed explanation of ADCL is provided after Def. 5.

Definition 5 (ADCL [16, Def. 9, 10]). *A* state *is a triple* $(\mathcal{S}, [\tau_i]_{i=1}^k, [B_i]_{i=0}^k)$ *where* $\mathcal{S} \supseteq \mathcal{T}$ *is a TS,* $\bigcup_{i=0}^k B_i \subseteq \mathsf{sip}(\mathcal{S})$*, and* $[\tau_i]_{i=1}^k \in \mathsf{sip}(\mathcal{S})^*$*. The transitions in* $\mathsf{sip}(\mathcal{T})$ *are called* original *and the transitions in* $\mathsf{sip}(\mathcal{S}) \setminus \mathsf{sip}(\mathcal{T})$ *are* learned*. A transition* $\tau_{k+1} \sqsubseteq B_k$ *is* blocked*, and* $\tau_{k+1} \not\sqsubseteq B_k$ *is* active *if* $\mathsf{chain}([\tau_i]_{i=1}^{k+1})$ *is an initial transition with satisfiable condition (i.e.,* $[\tau_i]_{i=1}^{k+1}$ *is a run). Let*

$$\mathsf{bt}(\mathcal{S}, [\tau_i]_{i=1}^k, [B_0, \ldots, B_k]) := (\mathcal{S}, [\tau_i]_{i=1}^{k-1}, [B_0, \ldots, B_{k-1} \cup \{\tau_k\}])$$

where bt *abbreviates "backtrack". Our calculus is defined by the following rules.*

$$\frac{}{\mathcal{T} \rightsquigarrow (\mathcal{T}, [], [\varnothing])} \quad (\textsc{Init}) \qquad \frac{\tau \in \mathsf{sip}(\mathcal{S}) \text{ is active}}{(\mathcal{S}, \vec{\tau}, \vec{B}) \rightsquigarrow (\mathcal{S}, \vec{\tau} :: \tau, \vec{B} :: \varnothing)} \quad (\textsc{Step})$$

$$\frac{\vec{\tau}^{\circlearrowleft} \text{ is recursive} \quad |\vec{\tau}^{\circlearrowleft}| = |\vec{B}^{\circlearrowleft}| \quad \mathsf{accel}(\vec{\tau}^{\circlearrowleft}) = \tau \not\sqsubseteq \mathsf{sip}(\mathcal{S})}{(\mathcal{S}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B} :: \vec{B}^{\circlearrowleft}) \rightsquigarrow (\mathcal{S} \cup \{\tau\}, \vec{\tau} :: \tau, \vec{B} :: \{\tau\})} \quad (\textsc{Accelerate})$$

$$\frac{\vec{\tau}' \sqsubset \mathsf{sip}(\mathcal{S}) \quad \text{or} \quad \vec{\tau}' \sqsubseteq \mathsf{sip}(\mathcal{S}) \wedge |\vec{\tau}'| > 1}{s = (\mathcal{S}, \vec{\tau} :: \vec{\tau}', \vec{B}) \rightsquigarrow \mathsf{bt}(s)} \quad (\textsc{Covered})$$

$$\frac{\text{all transitions from } \mathsf{sip}(\mathcal{S}) \text{ are inactive} \qquad \tau \text{ is safe}}{s = (\mathcal{S}, \vec{\tau} :: \tau, \vec{B}) \rightsquigarrow \mathsf{bt}(s)} \quad (\textsc{Backtrack})$$

$$\frac{\vec{\tau} \text{ is unsafe}}{(\mathcal{S}, \vec{\tau}, \vec{B}) \rightsquigarrow \mathsf{unsafe}} \quad (\textsc{Refute}) \qquad \frac{\text{all transitions from } \mathsf{sip}(\mathcal{S}) \text{ are inactive}}{(\mathcal{S}, [], [B]) \rightsquigarrow \mathsf{safe}} \quad (\textsc{Prove})$$

We write $\overset{\text{I}}{\rightsquigarrow}$, $\overset{\text{S}}{\rightsquigarrow}$, ... to indicate that the rule Init, Step, ... was used. Step adds a transition to the trace. When the trace has a recursive suffix, Accelerate allows for learning a new transition which then replaces the recursive suffix on the trace, or we may backtrack via Covered if the recursive suffix is redundant. Note that Covered does not apply if $\vec{\tau}' \sqsubseteq \mathsf{sip}(\mathcal{S})$ and $|\vec{\tau}'| = 1$, as it could immediately undo every Step, otherwise. If no further Step is possible, Backtrack applies. Note that Backtrack and Covered block the last transition from the trace so that we do not perform the same Step again. If $\vec{\tau}$ is an unsafe run, Refute yields unsafe, and if the entire search space has been exhausted without finding an unsafe run (i.e., if all initial transitions are blocked), Prove yields safe.

The definition of ADCL in [16] is more liberal than ours: In our setting, Accelerate may only be applied if the learned transition is non-redundant, and our definition of "active transitions" enforces that the first transition on the trace is always an initial transition. In [16], these requirements are not enforced by the definition of ADCL, but by the definition of *reasonable strategies* [16, Def. 14]. For simplicity, we integrated these requirements into Def. 5. Additionally, Covered should be preferred over Accelerate, and Accelerate should be preferred over Step.

Example 6. We apply ADCL to a version of Ex. 1 with the additional transition

$$\ell_1 \to \mathsf{err} \, [\![x = y \wedge x > 2 \cdot z \wedge \overline{\overline{x}} \wedge \overline{\overline{y}} \wedge \overline{\overline{z}}]\!]. \qquad (\tau_{\mathsf{err}})$$

$$\begin{aligned}
\mathcal{T} &\overset{\mathrm{I}}{\leadsto} (\mathcal{T}, [], [\varnothing]) \overset{\mathrm{S}}{\leadsto}{}^{2} (\mathcal{T}, [\tau_{\mathsf{i}}, \tau_{\ell_1}|_{\psi_{x<z}}], [\varnothing, \varnothing, \varnothing]) && (x \leq 1 \land z \geq 5k \land y \leq z)\\
&\overset{\mathrm{A}}{\leadsto} (\mathcal{S}_1, [\tau_{\mathsf{i}}, \tau^+_{x<z}], [\varnothing, \varnothing, \{\tau^+_{x<z}\}]) && (x \leq z \land z \geq 5k \land y \leq z)\\
&\overset{\mathrm{S}}{\leadsto} (\mathcal{S}_1, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau_{\ell_1}|_{\psi_{x\geq z}}], [\varnothing, \varnothing, \{\tau^+_{x<z}\}, \varnothing]) && (x = z+1 \land z \geq 5k \land y \leq z+1)\\
&\overset{\mathrm{A}}{\leadsto} (\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau^+_{x\geq z}], [\varnothing, \varnothing, \{\tau^+_{x<z}\}, \{\tau^+_{x\geq z}\}]) && (x \geq y \land x > z \geq 5k \land y \leq 2\cdot z+1)\\
&\overset{\mathrm{S}}{\leadsto} (\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau^+_{x\geq z}, \tau_{\mathsf{err}}], [\varnothing, \varnothing, \{\tau^+_{x<z}\}, \{\tau^+_{x\geq z}\}, \varnothing]) && (x = 2\cdot z+1 = y \land z \geq 5k)\\
&\overset{\mathrm{R}}{\leadsto} \mathsf{unsafe}
\end{aligned}$$

Here, $5k$ abbreviates 5000 and:

$$\begin{aligned}
\psi_{x<z} &:= y \leq 2\cdot z \land x{+}{+} \land x < z \land \overline{\overline{y}} \land \overline{\overline{z}} \qquad \psi_{x\geq z} := y \leq 2\cdot z \land x{+}{+} \land x \geq z \land y{+}{+} \land \overline{\overline{z}}\\
\tau^+_{x<z} &:= \ell_1 \to \ell_1 \, [\![y \leq 2\cdot z \land n > 0 \land x' = x+n \land x+n \leq z \land \overline{\overline{y}} \land \overline{\overline{z}}]\!]\\
\tau^+_{x\geq z} &:= \ell_1 \to \ell_1 \, [\![y+n-1 \leq 2\cdot z \land n > 0 \land x' = x+n \land x \geq z \land y' = y+n \land \overline{\overline{z}}]\!]\\
\mathcal{S}_1 &:= \mathcal{T} \cup \{\tau^+_{x<z}\} \qquad \mathcal{S}_2 := \mathcal{S}_1 \cup \{\tau^+_{x\geq z}\}
\end{aligned}$$

On the right, we show formulas describing the configurations that are reachable with the current trace. Every $\leadsto$-derivation starts with Init. The first two Steps add the initial transition τ_{i} and an element of $\mathsf{sip}(\tau_{\ell_1})$ to the trace. Since $x < z$ holds after applying τ_{i}, the only possible choice for the latter is $\tau_{\ell_1}|_{\psi_{x<z}}$.

As $\tau_{\ell_1}|_{\psi_{x<z}}$ is recursive, it is accelerated and replaced with $\mathsf{accel}(\tau_{\ell_1}|_{\psi_{x<z}}) = \tau^+_{x<z}$, which simulates n steps with $\tau_{\ell_1}|_{\psi_{x<z}}$. Moreover, $\tau^+_{x<z}$ is also added to the current set of blocking transitions, as we always have $\longrightarrow^2_\tau \subseteq \longrightarrow_\tau$ for learned transitions τ and thus adding them to the trace twice in a row is pointless.

Next, τ_{ℓ_1} is applicable again. As neither $x < z$ nor $x \geq z$ holds for all reachable configurations, we could continue with any element of $\mathsf{sip}(\tau_{\ell_1}) = \{\tau_{\ell_1}|_{\psi_{x<z}}, \tau_{\ell_1}|_{\psi_{x\geq z}}\}$. We choose $\tau_{\ell_1}|_{\psi_{x\geq z}}$, so that the recursive transition $\tau_{\ell_1}|_{\psi_{x\geq z}}$ can be accelerated to $\tau^+_{x\geq z}$. Then τ_{err} applies, and the proof is finished via Refute.

For our purposes, the most important property of ADCL is the following.

Theorem 7. *If $\mathcal{T} \leadsto^* (\mathcal{S}, \vec{\tau}, \vec{B})$ and $\vec{\tau}$ is non-empty, then $\mathsf{cond}(\vec{\tau}) \not\equiv_{\mathcal{A}} \bot$ and $\longrightarrow_{\vec{\tau}} \subseteq \longrightarrow^+_{\mathcal{T}}$. So if $\mathcal{T} \leadsto^* \mathsf{unsafe}$, then $\mathcal{T}$ is unsafe.*

The other properties of ADCL that were shown in [16] immediately carry over to our setting, too: if $\mathcal{T} \leadsto^* \mathsf{safe}$, then $\mathcal{T}$ is safe; if $\mathcal{T}$ is unsafe, then $\mathcal{T} \leadsto^* \mathsf{unsafe}$; in general, $\leadsto$ does not terminate. The proofs are analogous to [16].

4 Proving Non-Termination with ADCL-NT

From now on, we assume that the analyzed TS $\mathcal{T}$ does not contain unsafe transitions. To prove non-termination, we look for a corresponding *certificate*.

Definition 8 (Certificate of Non-Termination). *Let $\tau = \ell \to \ell\,[\![\ldots]\!]$. A satisfiable formula ψ* certifies non-termination of τ*, written $\psi \models^\infty_{\mathcal{A}} \tau$, if for any model σ of ψ, there is an infinite sequence $\ell(\sigma(\vec{x})) = \mathfrak{s}_1 \longrightarrow_\tau \mathfrak{s}_2 \longrightarrow_\tau \ldots$*

There exist many techniques for finding certificates of non-termination automatically, see Sect. 5. However, Def. 8 has several shortcomings. First, the problem of finding such certificates becomes very challenging if $\mathsf{cond}(\tau)$ contains disjunctions. Second, it is insufficient to consider a single transition when only non-singleton sequences $\vec{\tau}$ such that $\mathsf{chain}(\vec{\tau})$ is recursive admit non-terminating runs. Third, just finding a certificate ψ of non-termination for some $\vec{\tau} \in \mathcal{T}^*$ does not suffice for proving non-termination of $\mathcal{T}$. Additionally, a proof that the pre-image of $\longrightarrow_{\vec{\tau}|_\psi}$ is reachable from an initial configuration is required. All of these problems can be solved by integrating the search for certificates of non-termination into the ADCL calculus.

Definition 9 (ADCL-NT). *To prove non-termination, we extend ADCL with the rule* Nonterm *and modify* Covered *as shown below. We write* $\rightsquigarrow_{\mathsf{nt}}$ *for the relation defined by the (modified) rules from Def. 5 and* Nonterm*.*

$$\frac{\vec{\tau}^{\circlearrowleft} \textit{ is recursive} \qquad \vec{\tau}^{\circlearrowleft} \sqsubset \mathsf{sip}(\mathcal{S}) \textit{ or } \vec{\tau}^{\circlearrowleft} \sqsubseteq \mathsf{sip}(\mathcal{S}) \wedge |\vec{\tau}^{\circlearrowleft}| > 1}{s = (\mathcal{S}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B}) \rightsquigarrow_{\mathsf{nt}} \mathsf{bt}(s)} \quad (\textsc{Covered})$$

$$\frac{\mathsf{chain}(\vec{\tau}^{\circlearrowleft}) = \ell \to \ell \, [\![\ldots]\!] \qquad \psi \models^{\infty}_{\mathcal{A}} \vec{\tau}^{\circlearrowleft} \qquad \tau = \ell \to \mathsf{err} \, [\![\psi]\!] \not\sqsubseteq \mathsf{sip}(\mathcal{S})}{(\mathcal{S}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B}) \rightsquigarrow_{\mathsf{nt}} (\mathcal{S} \cup \{\tau\}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B})} \quad (\textsc{Nonterm})$$

So the idea of Nonterm is to apply a technique which searches for a certificate of non-termination to a recursive suffix of the trace. Apart from introducing Nonterm, we restricted Covered to recursive suffixes. The reason is that backtracking when the trace has a redundant, non-recursive suffix may prevent us from analyzing loops, resulting in a precision issue.

Example 10. Let $\mathcal{T} := \{\tau_{\mathsf{i}}, \tau_{\mathsf{i}}', \tau_\ell, \tau_{\ell'}\}$ where

$$\tau_{\mathsf{i}} := \mathsf{init} \to \ell \, [\![\top]\!] \qquad \tau_{\mathsf{i}}' := \mathsf{init} \to \ell' \, [\![\top]\!] \qquad \tau_\ell := \ell \to \ell' \, [\![\top]\!] \qquad \tau_{\ell'} := \ell' \to \ell \, [\![\top]\!]$$

and $\top$ means *true*. Due to the loop $\ell \longrightarrow_{\tau_\ell} \ell' \longrightarrow_{\tau_{\ell'}} \ell$, $\mathcal{T}$ is clearly non-terminating. Without requiring that $\vec{\tau}^{\circlearrowleft}$ is recursive in Covered, $\mathcal{T}$ can be analyzed as follows:

$$\begin{aligned} \mathcal{T} &\overset{\mathrm{I}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{T}, [], [\varnothing]) \overset{\mathrm{S}}{\rightsquigarrow}{}^{2}_{\mathsf{nt}} (\mathcal{T}, [\tau_{\mathsf{i}}, \tau_\ell], \varnothing^3) \overset{\mathrm{C}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{T}, [\tau_{\mathsf{i}}], [\varnothing, \{\tau_\ell\}]) \overset{\mathrm{B}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{T}, [], [\{\tau_{\mathsf{i}}\}]) \\ &\overset{\mathrm{S}}{\rightsquigarrow}{}^{2}_{\mathsf{nt}} (\mathcal{T}, [\tau_{\mathsf{i}}', \tau_{\ell'}], \{\tau_{\mathsf{i}}\} :: \varnothing^2) \overset{\mathrm{C}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{T}, [\tau_{\mathsf{i}}'], [\{\tau_{\mathsf{i}}\}, \{\tau_{\ell'}\}]) \overset{\mathrm{B}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{T}, [], [\{\tau_{\mathsf{i}}, \tau_{\mathsf{i}}'\}]) \overset{\mathrm{P}}{\rightsquigarrow}_{\mathsf{nt}} \mathsf{safe} \end{aligned}$$

The 1st application of Covered is possible as $[\tau_{\mathsf{i}}, \tau_\ell] \sqsubseteq \tau_{\mathsf{i}}'$ and the 2nd application of Covered is possible as $[\tau_{\mathsf{i}}', \tau_{\ell'}] \sqsubseteq \tau_{\mathsf{i}}$. Note that the trace never contains both τ_ℓ and $\tau_{\ell'}$, but both transitions are needed to prove non-termination.

Recall the shortcomings of Def. 8 mentioned above. First, due to the use of syntactic implicants, ADCL-NT reduces reasoning about arbitrary transitions to reasoning about conjunctive transitions. Second, as Nonterm considers a suffix $\vec{\tau}^{\circlearrowleft}$ of the trace, it can prove non-termination of sequences of transitions. Third, ADCL's capability to prove reachability directly carries over to our goal of proving non-termination. So in contrast to most other approaches (see Sect. 5), ADCL-NT does not have to resort to other tools or techniques for proving reachability.

We only search for a certificate of non-termination for $\vec{\tau}^{\circlearrowleft}$ if ADCL-NT established reachability of the pre-image of $\longrightarrow_{\vec{\tau}^{\circlearrowleft}}$ beforehand. Note, however, that this does not imply reachability of the pre-image of $\longrightarrow_{\ell \to \mathsf{err}\,[\![\psi]\!]}$, as ψ entails $\mathsf{cond}(\vec{\tau}^{\circlearrowleft})$, but not the other way around. Hence, we cannot directly derive non-termination of $\mathcal{T}$ when Nonterm applies. Regarding the strategy for $\rightsquigarrow_{\mathsf{nt}}$, one should try to use Nonterm once for each recursive suffix of the trace.

Example 11. Reconsider Ex. 1. Up to (excluding) the second-last step, the derivation from Ex. 6 remains unchanged. Then we get

$$\begin{array}{ll}
(\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau^+_{x\geq z}], [\ldots]) & (x \geq y \wedge x > 5k) \\
\overset{\mathrm{S}}{\rightsquigarrow}{}^4_{\mathsf{nt}} (\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau^+_{x\geq z}, \tau_{\ell_1\to\ell_2}, \tau^{=}_{\ell_2}, \tau^{\neq}_{\ell_2}|\psi_{x>y}, \tau^{\neq}_{\ell_2}|\psi_{x<y}], [\ldots]) & (1 \equiv_2 y = x > 10k) \\
\overset{\mathrm{N}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_3, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau^+_{x\geq z}, \tau_{\ell_1\to\ell_2}, \tau^{=}_{\ell_2}, \tau^{\neq}_{\ell_2}|\psi_{x>y}, \tau^{\neq}_{\ell_2}|\psi_{x<y}], [\ldots]) & (1 \equiv_2 y = x > 10k) \\
\overset{\mathrm{S}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_3, [\tau_{\mathsf{i}}, \tau^+_{x<z}, \tau^+_{x\geq z}, \tau_{\ell_1\to\ell_2}, \tau^{=}_{\ell_2}, \tau^{\neq}_{\ell_2}|\psi_{x>y}, \tau^{\neq}_{\ell_2}|\psi_{x<y}, \tau_{\mathsf{err}}], [\ldots]) \overset{\mathrm{R}}{\rightsquigarrow}_{\mathsf{nt}} \mathsf{unsafe} &
\end{array}$$

$$\begin{array}{lll}
\text{where} & \psi_{x>y} := x > 0 \wedge y > 0 \wedge x' = y \wedge x > y \wedge y' = x & \tau_{\mathsf{err}} := \ell_2 \to \mathsf{err}\,[\![x = y > 1]\!] \\
& \psi_{x<y} := x > 0 \wedge y > 0 \wedge x' = y \wedge x < y \wedge \overset{=}{y} & \mathcal{S}_3 := \mathcal{S}_2 \cup \{\tau_{\mathsf{err}}\}
\end{array}$$

The formulas on the right describe the values of x and y that are reachable with the current trace, where $1 \equiv_2 y$ means that y is odd. After the first Step with $\tau_{\ell_1\to\ell_2}$, just $\tau^{=}_{\ell_2}$ can be used, as $\mathsf{cond}(\tau_{\ell_1\to\ell_2})$ implies $x' = y'$. While $\tau^{=}_{\ell_2}$ is recursive, Accelerate cannot be applied next, as $\longrightarrow_{\tau^{=}_{\ell_2}} = \longrightarrow^+_{\tau^{=}_{\ell_2}}$, so the learned transition would be redundant. Thus, we continue with $\tau^{\neq}_{\ell_2}$, projected to $x > y$ (as $\mathsf{cond}(\tau^{=}_{\ell_2})$ implies $x' = y' + 1$). Again, all transitions that could be learned are redundant, so Accelerate does not apply. We next use $\tau^{\neq}_{\ell_2}$ projected to $x < y$, as the previous Step swapped x and y. As the suffix $[\tau^{=}_{\ell_2}, \tau^{\neq}_{\ell_2}|\psi_{x>y}, \tau^{\neq}_{\ell_2}|\psi_{x<y}]$ of the trace does not terminate (see Ex. 1), Nonterm applies. So we learn the transition τ_{err}, which is added to the trace to finish the proof, afterwards.

Theorem 12. *If $\mathcal{T} \rightsquigarrow^*_{\mathsf{nt}} \mathsf{unsafe}$, then $\mathcal{T}$ does not terminate.*

While Thm. 12 establishes the soundness of our approach, we now investigate completeness. In contrast to ADCL for safety (Sect. 3), ADCL-NT is not refutationally complete, but the proof is non-trivial. So in the following, we show that there are non-terminating TSs $\mathcal{T}$ where $\mathcal{T} \not\rightsquigarrow^*_{\mathsf{nt}} \mathsf{unsafe}$. To prove incompleteness, we adapt the construction from the proof that ADCL does not terminate [16, Thm. 18]. There, states $(\mathcal{S}, \vec{\tau}, \vec{B})$ were extended by a component $\mathcal{L}$ that maps every element of $\mathsf{sip}(\mathcal{S})$ to a regular language over $\mathsf{sip}(\mathcal{T})$. However, the proof of [16, Thm. 18] just required reasoning about finite (prefixes of infinite) runs, but we have to reason about infinite runs. So in our setting $\mathcal{L}$ maps each element τ of $\mathsf{sip}(\mathcal{S})$ to a regular or an ω-regular language over $\mathsf{sip}(\mathcal{T})$, i.e., $\mathcal{L}(\tau) \subseteq \mathsf{sip}(\mathcal{T})^*$ or $\mathcal{L}(\tau) \subseteq \mathsf{sip}(\mathcal{T})^\omega$. We lift $\mathcal{L}$ from $\mathsf{sip}(\mathcal{S})$ to sequences of transitions as follows.

$$\mathcal{L}(\varepsilon) := \varepsilon \qquad\qquad \mathcal{L}(\vec{\tau} :: \tau) := \mathcal{L}(\vec{\tau}) :: \mathcal{L}(\tau) \quad \text{if} \quad \mathcal{L}(\tau) \subseteq \mathsf{sip}(\tau)^*$$

Here, "::" denotes language concatenation (i.e., $\mathcal{L}_1 :: \mathcal{L}_2 = \{\tau_1 :: \tau_2 \mid \tau_1 \in \mathcal{L}_1, \tau_2 \in \mathcal{L}_2\}$) and we only consider sequences where $\mathcal{L}(\tau)$ is regular (not ω-regular) to ensure that $\mathcal{L}$ is well defined. So while we lift other notations to sequences of transitions via chaining, $\mathcal{L}(\vec{\tau})$ does *not* stand for $\mathcal{L}(\mathsf{chain}(\vec{\tau}))$.

Definition 13 (ADCL-NT with Regular Languages). *We extend states by a fourth component $\mathcal{L}$, and adapt* Init, Accelerate, *and* Nonterm *as follows:*

$$\frac{\mathcal{L}(\tau) = \{\tau\} \text{ for all } \tau \in \mathsf{sip}(\mathcal{T})}{\mathcal{T} \rightsquigarrow_{\mathsf{nt}} (\mathcal{T}, [], [\varnothing], \mathcal{L})} \quad (\textsc{Init})$$

$$\frac{\vec{\tau}^{\circlearrowleft} \text{ is recursive} \qquad |\vec{\tau}^{\circlearrowleft}| = |\vec{B}^{\circlearrowleft}| \qquad \mathsf{accel}(\vec{\tau}^{\circlearrowleft}) = \tau \not\sqsubseteq \mathsf{sip}(\mathcal{S})}{(\mathcal{S}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B} :: \vec{B}^{\circlearrowleft}, \mathcal{L}) \rightsquigarrow_{\mathsf{nt}} (\mathcal{S} \cup \{\tau\}, \vec{\tau} :: \tau, \vec{B} :: \{\tau\}, \mathcal{L} \uplus (\tau \mapsto \mathcal{L}(\vec{\tau}^{\circlearrowleft})^{+}))} \quad (\textsc{Accelerate})$$

$$\frac{\mathsf{chain}(\vec{\tau}^{\circlearrowleft}) = \ell \to \ell \, [\![\ldots]\!] \qquad \psi \models^{\infty}_{\mathcal{A}} \vec{\tau}^{\circlearrowleft} \qquad \tau = \ell \to \mathsf{err} \, [\![\psi]\!] \not\sqsubseteq \mathsf{sip}(\mathcal{S})}{(\mathcal{S}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B}, \mathcal{L}) \rightsquigarrow_{\mathsf{nt}} (\mathcal{S} \cup \{\tau\}, \vec{\tau} :: \vec{\tau}^{\circlearrowleft}, \vec{B}, \mathcal{L} \uplus (\tau \mapsto \mathcal{L}(\vec{\tau}^{\circlearrowleft})^{\omega}))} \quad (\textsc{Nonterm})$$

All other rules from Def. 5 leave the last component of the state unchanged.

Here, $\mathcal{L}(\pi)^{+} := \bigcup_{n \in \mathbb{N}_{\geq 1}} \mathcal{L}(\pi)^{n}$, and $\mathcal{L}(\pi)^{\omega}$ is the ω-regular language consisting of all words that result from concatenating infinitely many elements of $\mathcal{L}(\pi) \setminus \{\varepsilon\}$.

In Accelerate and Nonterm, $\mathsf{chain}(\vec{\tau}^{\circlearrowleft})$ is recursive. Thus, $\vec{\tau}^{\circlearrowleft}$ does not contain unsafe transitions. Hence, $\mathcal{L}(\vec{\tau}^{\circlearrowleft})$ and thus also $\mathcal{L}(\vec{\tau}^{\circlearrowleft})^{+}$ are well defined and regular, and $\mathcal{L}(\vec{\tau}^{\circlearrowleft})^{\omega}$ is ω-regular. Moreover, the use of "$\uplus$" is justified by the condition $\tau \not\sqsubseteq \mathsf{sip}(\mathcal{S})$. The next lemma states two crucial properties about $\mathcal{L}$.

Lemma 14. *Assume $\mathcal{T} \rightsquigarrow^{*}_{\mathsf{nt}} (\mathcal{S}, \vec{\tau}, \vec{B}, \mathcal{L})$ and let $\tau = (\ell \to \ell' \, [\![\psi]\!]) \in \mathsf{sip}(\mathcal{S})$.*

- *If $\mathcal{L}(\tau) \subseteq \mathsf{sip}(\mathcal{T})^{*}$, then $\longrightarrow_{\tau} = \bigcup_{\vec{\tau} \in \mathcal{L}(\tau)} \longrightarrow_{\vec{\tau}}$.*
- *If $\mathcal{L}(\tau) \subseteq \mathsf{sip}(\mathcal{T})^{\omega}$, then for every model σ of ψ, there is an infinite sequence $\ell(\sigma(\vec{x})) = \mathfrak{s}_1 \longrightarrow_{\tau_1} \mathfrak{s}_2 \longrightarrow_{\tau_2} \ldots$ where $[\tau_1, \tau_2, \ldots] \in \mathcal{L}(\tau)$.*

Based on this lemma, we can prove that our extension of $\rightsquigarrow_{\mathsf{nt}}$ from Def. 13 is not refutationally complete. Then refutational incompleteness of ADCL-NT as introduced in Def. 9 follows immediately. The reason is that $\mathcal{L}$ is only used in the premise of Init in Def. 13, but there the requirement "$\mathcal{L}(\tau) = \{\tau\}$ for all $\tau \in \mathsf{sip}(\mathcal{T})$" is trivially satisfiable by choosing $\mathcal{L}$ accordingly.

Theorem 15. *There is a non-terminating TS $\mathcal{T}$ such that $\mathcal{T} \not\rightsquigarrow^{*}_{\mathsf{nt}}$ unsafe.*

Proof (Sketch). As in the proof of [16, Thm. 18], for any (original or learned) transition τ such that $\mathcal{L}(\tau)$ is regular, $\mathcal{L}(\tau)$ contains at most one square-free word (i.e., a word without a non-empty infix $w :: w$). Thus, if $\mathcal{L}(\tau)$ is ω-regular, then $\mathcal{L}(\tau)$ does not contain an infinite square-free word. Moreover, as in the proof of [16, Thm. 18], one can construct a TS $\mathcal{T}$ that admits a single infinite run $\vec{\tau}$, and this infinite run is square-free. Thus, there is no transition τ such that $\mathcal{L}(\tau)$ contains a suffix of $\vec{\tau}$, i.e., no $\rightsquigarrow_{\mathsf{nt}}$-derivation starting with $\mathcal{T}$ corresponds to $\vec{\tau}$. Hence, by Lemma 14, assuming $\mathcal{T} \rightsquigarrow^{*}_{\mathsf{nt}}$ unsafe results in a contradiction. ☐

Since ADCL can prove unsafety as well as safety, it is natural to ask if there is a dual to ADCL-NT that can prove termination. The most obvious approach would be the following: Whenever the trace has a recursive suffix $\vec{\tau}^{\circlearrowleft}$, then termination of $\vec{\tau}^{\circlearrowleft}$ needs to be proven before the next $\rightsquigarrow$-step. The following example shows that this is not enough to ensure that $\mathcal{T} \rightsquigarrow^{+}_{\mathsf{nt}}$ safe implies termination of $\mathcal{T}$.

Example 16. Let $\mathcal{T} := \{\tau_{\mathsf{i}} = \mathsf{init} \to \ell \, [\![\psi_{\mathsf{i}}]\!]\} \cup \{\tau_m = \ell \to \ell \, [\![\psi_m]\!] \mid 0 \leq m \leq 2\}$ and

$$\psi_{\mathsf{i}} := x' = 0 \quad \psi_0 := x = 0 \land x' = 1 \quad \psi_1 := x = 1 \land x' = 2 \quad \psi_2 := x = 2 \land x' = 1.$$

As we have $\ell(1) \longrightarrow_{\tau_1} \ell(2) \longrightarrow_{\tau_2} \ell(1)$, $\mathcal{T}$ is clearly non-terminating. We get:

$$\begin{aligned}
\mathcal{T} &\overset{\mathrm{I}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{T}, [], [\varnothing]) \overset{\mathrm{S}}{\rightsquigarrow}_{\mathsf{nt}}^{3} (\mathcal{T}, [\tau_{\mathsf{i}}, \tau_0, \tau_1], \varnothing^4) \overset{\mathrm{A}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_1, [\tau_{\mathsf{i}}, \tau_{01}], \varnothing^2 :: \{\tau_{01}\}) \\
&\overset{\mathrm{S}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_1, [\tau_{\mathsf{i}}, \tau_{01}, \tau_2], \varnothing^2 :: \{\tau_{01}\} :: \varnothing) \overset{\mathrm{A}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau_{012}], \varnothing^2 :: \{\tau_{01}, \tau_{012}\}) \\
&\overset{\mathrm{S}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau_{012}, \tau_1], \varnothing^2 :: \{\tau_{01}, \tau_{012}\} :: \varnothing) \overset{\mathrm{C}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_2, [\tau_{\mathsf{i}}, \tau_{012}], \varnothing^2 :: \{\tau_{01}, \tau_{012}, \tau_1\}) \\
&\overset{\mathrm{B}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_2, [\tau_{\mathsf{i}}], \varnothing :: \{\tau_{012}\}) \rightsquigarrow_{\mathsf{nt}}^{*} (\mathcal{S}_2, [\tau_{\mathsf{i}}], \varnothing :: \{\tau_{012}, \tau_0, \tau_{01}\}) \overset{\mathrm{B}}{\rightsquigarrow}_{\mathsf{nt}} (\mathcal{S}_2, [], [\{\tau_{\mathsf{i}}\}]) \overset{\mathrm{P}}{\rightsquigarrow}_{\mathsf{nt}} \mathsf{safe}
\end{aligned}$$

After three Steps, we accelerate the recursive suffix $[\tau_0, \tau_1]$ of the trace, resulting in $\tau_{01} = \ell \to \ell\,[\![x = 0 \land x' = 2]\!]$ and $\mathcal{S}_1 = \mathcal{T} \cup \{\tau_{01}\}$. After one more step, $[\tau_{01}, \tau_2]$ is accelerated to $\tau_{012} = \ell \to \ell\,[\![x = 0 \land x' = 1]\!]$ and we get $\mathcal{S}_2 = \mathcal{S}_1 \cup \{\tau_{012}\}$. After the next step, $[\tau_{012}, \tau_1]$ is redundant w.r.t. τ_{01}, so Covered applies. Then we Backtrack, as no other transitions are active. The next Steps also yield states that allow for backtracking (as their traces have the redundant suffixes $[\tau_0, \tau_1]$ and $[\tau_{01}, \tau_2]$), so we can finally apply Backtrack again and finish with Prove.

Note that whenever the trace has a recursive suffix, then it leads from $\ell(i)$ to $\ell(j)$ where $i \neq j$, i.e., each such suffix is trivially terminating. In particular, the cycle $\ell(1) \longrightarrow_{\tau_1} \ell(2) \longrightarrow_{\tau_2} \ell(1)$ is not apparent in any of the states.

This example reveals a fundamental problem when adapting ADCL for proving termination: ADCL ensures that all reachable *configurations* are covered, which is crucial for proving safety, but there are no such guarantees for all *runs*. Therefore, we think that adapting ADCL for proving termination requires major changes.

5 Related Work and Experiments

We presented ADCL-NT, a variant of ADCL for proving non-termination. The key insight is that tightly integrating techniques to detect non-terminating transitions into ADCL allows for handling classes of TSs that are challenging for other techniques. In particular, ADCL-NT can find non-terminating executions involving disjunctive transitions or complex patterns of transitions. Moreover, it tightly couples the search for non-terminating configurations and the proof of their reachability, whereas other approaches usually separate these two steps.

Related Work There are many techniques to find certificates of non-termination [2,14,15,22,23,25]. We could use any of them (they are black boxes for ADCL-NT).

Most non-termination techniques for TSs first search for non-terminating configurations, and then prove their reachability [5, 6, 9, 22], or they extract and analyze *lassos* [23]. In contrast, ADCL-NT tightly integrates the search for non-terminating configurations and reachability analysis.

Earlier versions of our tool LoAT [12, 15] also interleaved both steps using a technique akin to the state elimination method to transform finite automata to regular expressions. This technique cannot handle disjunctions, and it is incomplete for reachability. Hence, LoAT is now solely based on ADCL-NT.

Implementation So far, our implementation in our tool LoAT is restricted to integer arithmetic. It uses the technique from [15] for acceleration and finding certificates of non-termination, the SMT solvers Z3 [26] and Yices [11], the recurrence solver PURRS [1], and libFAUDES [24] to implement the automata-based redundancy check from [16].

Experiments To evaluate our implementation in LoAT, we used the 1222 *Integer Transition Systems* (ITSs) and the 335 C *Integer Programs* from the *Termination Problems Database* [28] used in *TermComp* [21]. The C programs are small, hand-crafted examples that often require complex proofs. The ITSs are significantly larger, as they were obtained from automatic transformations of C or Java programs. Moreover, they contain a lot of "noise", e.g., branches where termination is trivial or variables that are irrelevant for (non-)termination. Thus, they are well suited to test the scalability and robustness of the tools.

We compared our implementation (LoAT ADCL) with other leading termination analyzers: iRankFinder [2,9], T2 [6], Ultimate [8], VeryMax [3,22], and the previous version of LoAT [15] (LoAT '22). For T2, VeryMax, and Ultimate, we took the versions of their last *TermComp* participations (2015, 2019, and 2022). For iRankFinder, we used the configuration from the evaluation of [15], which is tailored towards proving non-termination. We excluded AProVE [20], as it cannot prove non-termination of ITSs, and it uses LoAT and T2 as backends when analyzing C programs. Moreover, we excluded Ultimate from the evaluation on ITSs, as it cannot parse them. All experiments were run on StarExec [27] with 300s wallclock timeout, 1200s CPU timeout, and 128GB memory limit per example.

	No		Yes	Runtime overall			Runtime No	
	solved	unique	solved	average	median	timeouts	average	median
LoAT ADCL	521	9	0	48.6 s	0.1 s	183	2.9 s	0.1 s
LoAT '22	494	2	0	7.4 s	0.1 s	0	6.2 s	0.1 s
T2	442	3	615	17.2 s	0.6 s	45	7.4 s	0.6 s
VeryMax	421	6	631	28.3 s	0.5 s	30	30.5 s	14.5 s
iRankFinder	409	0	642	32.0 s	2.0 s	93	12.3 s	1.7 s

The table above shows the results for ITSs, where the column "unique" contains the number of examples that could be solved by the respective tool, but no others. It shows that LoAT ADCL is the most powerful tool for proving non-termination of ITSs. The main reasons for the improvement are that LoAT ADCL builds upon a complete technique for proving reachability (in contrast to, e.g., LoAT '22), and the close integration of non-termination techniques into a technique for proving reachability, whereas most competing tools separate these steps from each other.

If we only consider the examples where non-termination is proven, LoAT ADCL is also the fastest tool. If we consider all examples, then the *average* runtime of LoAT ADCL is significantly slower. This is not surprising, as ADCL-NT does not terminate in general. So while it is very fast in most cases (as witnessed by the very fast *median* runtime), it times out more often than the other tools.

For C integer programs, the best tools are very close (VeryMax: 103×No, LoAT ADCL: 102×No, Ultimate: 100×No). Regarding runtimes, the situation is analogous to ITSs. See [18] for detailed results, more information about our evaluation, and a pre-compiled binary. LoAT is open-source and available on GitHub [17].

References

1. 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), https://arxiv.org/abs/cs/0512056
2. Ben-Amram, A.M., Doménech, J.J., Genaim, S.: Multiphase-linear ranking functions and their relation to recurrent sets. In: SAS '19. pp. 459–480. LNCS 11822 (2019). https://doi.org/10.1007/978-3-030-32304-2_22
3. Borralleras, C., Brockschmidt, M., Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving termination through conditional termination. In: TACAS '17. pp. 99–117. LNCS 10205 (2017). https://doi.org/10.1007/978-3-662-54577-5_6
4. Bozga, M., Gîrlea, C., Iosif, R.: Iterating octagons. In: TACAS '09. pp. 337–351. LNCS 5505 (2009). https://doi.org/10.1007/978-3-642-00768-2_29
5. Brockschmidt, M., Ströder, T., Otto, C., Giesl, J.: Automated detection of non-termination and `NullPointerException`s for Java Bytecode. In: FoVeOOS '11. pp. 123–141. LNCS 7421 (2012). https://doi.org/10.1007/978-3-642-31762-0_9
6. Brockschmidt, M., Cook, B., Ishtiaq, S., Khlaaf, H., Piterman, N.: T2: Temporal property verification. In: TACAS '16. pp. 387–393. LNCS 9636 (2016). https://doi.org/10.1007/978-3-662-49674-9_22
7. CHC Competition, https://chc-comp.github.io
8. Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M., Turrini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In: PLDI '18. pp. 135–150 (2018). https://doi.org/10.1145/3192366.3192405
9. Doménech, J.J., Genaim, S.: iRankFinder. In: WST '18. p. 83 (2018), https://wst2018.webs.upv.es/wst2018proceedings.pdf
10. Doménech, J.J., Gallagher, J.P., Genaim, S.: Control-flow refinement by partial evaluation, and its application to termination and cost analysis. Theory Pract. Log. Program. **19**(5-6), 990–1005 (2019). https://doi.org/10.1017/S1471068419000310
11. Dutertre, B.: Yices 2.2. In: CAV '14. pp. 737–744. LNCS 8559 (2014). https://doi.org/10.1007/978-3-319-08867-9_49
12. Frohn, F., Giesl, J.: Proving non-termination via loop acceleration. In: FMCAD '19. pp. 221–230 (2019). https://doi.org/10.23919/FMCAD.2019.8894271
13. Frohn, F.: A calculus for modular loop acceleration. In: TACAS '20. pp. 58–76. LNCS 12078 (2020). https://doi.org/10.1007/978-3-030-45190-5_4
14. Frohn, F., Fuhs, C.: A calculus for modular loop acceleration and non-termination proofs. Int. J. Softw. Tools Technol. Transf. **24**(5), 691–715 (2022). https://doi.org/10.1007/s10009-022-00670-2
15. Frohn, F., Giesl, J.: Proving non-termination and lower runtime bounds with LoAT (system description). In: IJCAR '22. pp. 712–722. LNCS 13385 (2022). https://doi.org/10.1007/978-3-031-10769-6_41
16. Frohn, F., Giesl, J.: ADCL: Acceleration Driven Clause Learning for constrained Horn clauses. CoRR **abs/2303.01827** (2023), https://arxiv.org/abs/2303.01827
17. Frohn, F.: LoAT on GitHub (2023), https://github.com/LoAT-developers/LoAT
18. Frohn, F., Giesl, J.: Empirical evaluation of "Proving non-termination by Acceleration Driven Clause Learning" (2023), https://loat-developers.github.io/adcl-nonterm-eval
19. Frohn, F., Giesl, J.: Proving non-termination by Acceleration Driven Clause Learning. CoRR **abs/2304.10166** (2023), https://arxiv.org/abs/2304.10166
20. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. J. Autom. Reasoning **58**(1), 3–31 (2017). https://doi.org/10.1007/s10817-016-9388-y

21. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: TACAS '19. pp. 156–166. LNCS 11429 (2019). https://doi.org/10.1007/978-3-030-17502-3_10
22. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using Max-SMT. In: CAV '14. pp. 779–796. LNCS 8559 (2014). https://doi.org/10.1007/978-3-319-08867-9_52
23. 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
24. libFAUDES Library, https://fgdes.tf.fau.de/faudes/index.html
25. Nishida, N., Winkler, S.: Loop detection by logically constrained term rewriting. In: VSTTE '18. pp. 309–321. LNCS 11294 (2018). https://doi.org/10.1007/978-3-030-03592-1_18
26. 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
27. 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
28. Termination Problems Data Base (TPDB), https://termination-portal.org/wiki/TPDB