Accelerated Bounded Model Checking

Abstract. *Bounded Model Checking* (BMC) is a powerful technique for proving reachability of error states, i.e., unsafety. However, finding *deep counterexamples* that require a large bound is challenging for BMC. On the other hand, *acceleration techniques* compute "shortcuts" that "compress" many execution steps into a single one. In this paper, we tightly integrate acceleration techniques into SMT-based bounded model checking. By adding suitable "shortcuts" to the SMT-problem on the fly, our approach can quickly detect deep counterexamples, even when only using small bounds. Moreover, using so-called *blocking clauses*, our approach can prove safety of examples where BMC diverges. An empirical comparison with other state-of-the-art techniques shows that our approach is highly competitive for proving unsafety, and orthogonal to existing techniques for proving safety.

1 Introduction

Bounded Model Checking (BMC) is a powerful technique for disproving safety properties of software. However, as it uses breadth-first search to find counterexamples, the search space grows exponentially w.r.t. the *bound*, i.e., the limit on the length of potential counterexamples. Thus, finding *deep counterexamples* that require large bounds is challenging for BMC. On the other hand, *acceleration techniques* can compute a first-order formula that characterizes the transitive closure of the transition relation induced by a loop. Intuitively, such a formula corresponds to a "shortcut" that "compresses" many execution steps into a single one. In this paper, we consider relations defined by quantifier-free first-order formulas over some background theory like non-linear integer arithmetic and two disjoint vectors of variables $\vec{x}$ and $\vec{x}'$, called the *pre-* and *post-variables*. Such *transition formulas* can easily represent, e.g., *transition systems* (TSs), linear *Constrained Horn Clauses* (CHCs), *control-flow automata* (CFAs), ...,[1] i.e., they subsume many popular intermediate representations used for verification of programs written in more expressive languages.[2]

Example 1. Consider the transition formula $\tau := \tau_{x<100} \vee \tau_{x=100}$ where

[1] To this end, it suffices to introduce one additional variable that represents the control-flow location (for TSs and CFAs) or predicate (for CHCs).

[2] Essentially, the same formalism is, e.g., used in the category "Linear Real Arithmetic - Transition Systems" of the annual CHC-competition [8], where linear CHCs with just a single uninterpreted predicate are considered, so all information about the control flow is encoded in the constraints.

$\tau_{x<100} := x < 100 \wedge x' = x + 1 \wedge y' = y$ and
$\tau_{x=100} := x = 100 \wedge x' = 0 \wedge y' = y + 1.$

```
while (x <= 100) {
  while (x < 100) x++;
  x = 0, y++;
}
```

Listing 1: implementation of τ

It defines a relation $\to_\tau$ on $\mathbb{Z} \times \mathbb{Z}$ by relating the pre-variables x and y with the post-variables x' and y'. So for all $c_x, c_y, c'_x, c'_y \in \mathbb{Z}$ we have $(c_x, c_y) \to_\tau (c'_x, c'_y)$ iff $[x/c_x, y/c_y, x'/c'_x, y'/c'_y]$ is a model of τ. In other words, $(c_x, c_y) \to^*_\tau (c'_x, c'_y)$ if and only if a state with $x = c'_x \wedge y = c'_y$ is reachable from a state with $x = c_x \wedge y = c_y$ in Listing 1. Assume that we want to prove that an *error state* which satisfies $\psi_{\mathsf{err}} := y \geq 100$ is reachable from an *initial state* which satisfies $\psi_{\mathsf{init}} := x \leq 0 \wedge y \leq 0$. To do so, BMC has to unroll τ 10100 times.

Our new technique called *Accelerated* BMC (ABMC) accelerates τ, resulting in the "shortcut"

$$n > 0 \wedge x + n \leq 100 \wedge x' = x + n \wedge y' = y, \qquad (\tau^+_{\mathsf{i}})$$

meaning that we have $(c_x, x_y) \to^+_\tau (c'_x, c'_y)$ if $\tau^+_{\mathsf{i}}[x/c_x, y/c_y, x'/c'_x, y'/c'_y]$ is satisfiable. So τ^+_{i} can simulate arbitrarily many steps with τ in a single step, as long as x does not exceed 100. Here, acceleration was applied to $\tau_{x<100}$, i.e., the projection of τ to the case $x < 100$, which corresponds to the inner loop of Listing 1. We also call such projections *transitions*. Later, ABMC also accelerates the outer loop (consisting of $\tau_{x=100}$, $\tau_{x<100}$, and τ^+_{i}), resulting in

$$n > 0 \wedge x = 100 \wedge 1 < x' \leq 100 \wedge y' = y + n. \qquad (\tau^+_{\mathsf{o}})$$

For technical reasons, our algorithm accelerates $[\tau_{x=100}, \tau_{x<100}, \tau^+_{\mathsf{i}}]$ instead of just $[\tau_{x=100}, \tau^+_{\mathsf{i}}]$, so that τ^+_{o} requires $1 < x'$, i.e., it only covers cases where $\tau_{x<100}$ is applied at least twice after $\tau_{x=100}$. Details will be clarified in Sect. 3.2, see in particular Fig. 2. Using these shortcuts, ABMC can prove unsafety with bound 7.

While our main goal is to improve BMC's capability to find deep counterexamples, the following straightforward observations can be used to *block* certain parts of the transition relation in ABMC:

1. After accelerating a sequence of transitions, the resulting accelerated transition should be preferred over that sequence of transitions.
2. If an accelerated transition has been used, then the corresponding sequence of transitions should not be used immediately afterwards.

Both observations are justified by the fact that an accelerated transition describes the transitive closure of the transition relation induced by the corresponding sequence of transitions. Due to its ability to block parts of the transition relation, ABMC is able to prove safety in cases where BMC would unroll the transition relation indefinitely.

The idea of using acceleration to detect deep counterexamples is not new. However, earlier approaches either combined acceleration with off-the-shelf model checkers [19], or they used depth-first search [14]. In contrast, ABMC tightly integrates acceleration into the breadth-first model-checking algorithm BMC.

After introducing preliminaries in Sect. 2, we show how to use acceleration in order to improve the BMC algorithm to ABMC in Sect. 3. Sect. 4 refines ABMC by integrating blocking clauses. In Sect. 5, we discuss related work and evaluate our implementation of ABMC in the tool LoAT.

2 Preliminaries

We assume familiarity with basics from many-sorted first-order logic. Without loss of generality, we assume that all formulas are in negation normal form (NNF). $\mathcal{V}$ is a countably infinite set of variables and $\mathcal{A}$ is a first-order theory over a k-sorted signature Σ with carrier $\mathcal{C} = (\mathcal{C}_1, \dots, \mathcal{C}_k)$. For each entity e, $\mathcal{V}(e)$ is the set of variables that occur in e. $\mathsf{QF}(\Sigma)$ denotes the set of all quantifier-free first-order formulas over Σ, and $\mathsf{QF}_\wedge(\Sigma)$ only contains conjunctions of Σ-literals. We let $\top$ and $\bot$ stand for "true" and "false", respectively.

Given $\psi \in \mathsf{QF}(\Sigma)$ with $\mathcal{V}(\psi) = \vec{y}$, we say that ψ is *$\mathcal{A}$-valid* (written $\models_\mathcal{A} \psi$) if every model of $\mathcal{A}$ satisfies the universal closure $\forall \vec{y}.\, \psi$ of ψ. Moreover, $\sigma : \mathcal{V}(\psi) \to \mathcal{C}$ is an *$\mathcal{A}$-model* of ψ (written $\sigma \models_\mathcal{A} \psi$) if $\models_\mathcal{A} \sigma(\psi)$, where $\sigma(\psi)$ results from ψ by instantiating all variables according to σ. If ψ has an $\mathcal{A}$-model, then ψ is *$\mathcal{A}$-satisfiable*. We write $\psi \models_\mathcal{A} \psi'$ for $\models_\mathcal{A} (\psi \implies \psi')$, and $\psi \equiv_\mathcal{A} \psi'$ means $\models_\mathcal{A} (\psi \iff \psi')$. In the sequel, we omit the subscript $\mathcal{A}$, and we just say "valid", "model", and "satisfiable" instead of "$\mathcal{A}$-valid", "$\mathcal{A}$-model", and "$\mathcal{A}$-satisfiable". We assume that $\mathcal{A}$ is complete, i.e., we have either $\models \psi$ or $\models \neg\psi$ for every closed formula over Σ.

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 $[x] :: xs$.

Let $d \in \mathbb{N}$ be fixed, and let $\vec{x}, \vec{x}' \in \mathcal{V}^d$ be disjoint vectors of pairwise different variables, called the *pre-* and *post-variables*. Each $\tau \in \mathsf{QF}(\Sigma)$ induces a *transition relation* $\to_\tau$ on $\mathcal{C}^d$ where $\vec{s} \to_\tau \vec{t}$ iff $\tau[\vec{x}/\vec{s}, \vec{x}'/\vec{t}]$ is satisfiable. Here, $[\vec{x}/\vec{s}, \vec{x}'/\vec{t}]$ denotes the substitution θ with $\theta(x_i) = s_i$ and $\theta(x_i') = t_i$ for all $1 \leq i \leq d$. We refer to elements of $\mathsf{QF}(\Sigma)$ as *transition formulas* whenever we are interested in their induced transition relation. Moreover, we also refer to *conjunctive* transition formulas (i.e., elements of $\mathsf{QF}_\wedge(\Sigma)$) as *transitions*. A *safety problem* $\mathcal{T}$ is a triple $(\psi_\mathsf{init}, \tau, \psi_\mathsf{err}) \in \mathsf{QF}(\Sigma) \times \mathsf{QF}(\Sigma) \times \mathsf{QF}(\Sigma)$ where $\mathcal{V}(\psi_\mathsf{init}) \cup \mathcal{V}(\psi_\mathsf{err}) \subseteq \vec{x}$. It is *unsafe* if there are $\vec{s}, \vec{t} \in \mathcal{C}^d$ such that $[\vec{x}/\vec{s}] \models \psi_\mathsf{init}$, $\vec{s} \to_\tau^* \vec{t}$, and $[\vec{x}/\vec{t}] \models \psi_\mathsf{err}$.

The *composition* of τ and τ' is $\circledcirc(\tau, \tau') := \tau[\vec{x}'/\vec{x}''] \wedge \tau'[\vec{x}/\vec{x}'']$ where $\vec{x}'' \in \mathcal{V}^d$ is fresh. Here, we assume $\mathcal{V}(\tau) \cap \mathcal{V}(\tau') \subseteq \vec{x} \cup \vec{x}'$ (which can be ensured by renaming other variables correspondingly). So $\to_{\circledcirc(\tau,\tau')} = \to_\tau \circ \to_{\tau'}$ (where $\circ$ denotes relational composition). For finite sequences of transition formulas we define $\circledcirc([]) := (\vec{x} = \vec{x}')$ (i.e., $\to_{\circledcirc([])}$ is the identity relation) and $\circledcirc(\tau :: \vec{\tau}) := \circledcirc(\tau, \circledcirc(\vec{\tau}))$. We abbreviate $\to_{\circledcirc(\vec{\tau})}$ by $\to_{\vec{\tau}}$.

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 [6], or have to resort to approximations in the presence of disjunctions [11].

Algorithm 1: BMC

Input: a safety problem $\mathcal{T} = (\psi_{\mathsf{init}}, \tau, \psi_{\mathsf{err}})$

```
1  b ← 0
2  add(μ_b(ψ_init))
3  while ⊤ do
4      push()
5      add(μ_b(ψ_err))
6      if check_sat() = sat then return unsafe
7      pop()
8      add(μ_b(τ))
9      if check_sat() = unsat then return safe
10     b ← b + 1
```

Definition 2 (Acceleration). *An* acceleration technique *is a function* $\mathsf{accel} : \mathsf{QF}_\wedge(\Sigma) \to \mathsf{QF}_\wedge(\Sigma')$ *such that* $\to_{\mathsf{accel}(\tau)} = \to_\tau^+$*, where* Σ' *is the signature of a first-order theory* $\mathcal{A}'$*.*

We abbreviate $\mathsf{accel}(\circledcirc(\vec{\tau}))$ by $\mathsf{accel}(\vec{\tau})$. Note that most theories are not "closed under acceleration". For example, accelerating the following Presburger formula on the left may yield the non-linear formula on the right:

$$x' = x + y \wedge y' = y \qquad\qquad n > 0 \wedge x' = x + n \cdot y \wedge y' = y.$$

Hence, Def. 2 allows $\mathcal{A}' \neq \mathcal{A}$.

3 From BMC to ABMC

In this section, we introduce accelerated bounded model checking. To this end, we first recapitulate bounded model checking in Sect. 3.1. Then we present ABMC in Sect. 3.2. To implement ABMC efficiently, heuristics to decide when to perform acceleration are needed. Thus, we present such a heuristic in Sect. 3.3.

3.1 Bounded Model Checking

Alg. 1 shows how to implement BMC on top of an incremental SMT solver. First, the description of the initial states is added to the SMT problem (Line 2). Here and in the following, for all $i \in \mathbb{N}$ we define $\mu_i(x) := x^{(i)}$ if $x \in \mathcal{V} \setminus \vec{x}'$, and $\mu_i(x') = x^{(i+1)}$ if $x' \in \vec{x}'$. So in particular, we have $\mu_i(\vec{x}) = \vec{x}^{(i)}$ and $\mu_i(\vec{x}') = \vec{x}^{(i+1)}$, where we assume that $\vec{x}^{(0)}, \vec{x}^{(1)}, \ldots \in \mathcal{V}^d$ are disjoint vectors of pairwise different variables. In the loop, we first check if an error state can be reached with the current bound b. To this end, we set a backtracking point with the "$\mathsf{push}()$" command (Line 4), add a suitably variable renamed version of the description of the error states to the SMT problem (Line 5), and check for satisfiability (Line 6). If no error state is reachable yet, the description of the error states is removed from the SMT problem with the "$\mathsf{pop}()$" command that deletes all formulas from the SMT problem that have been added since the last backtracking point (Line 7). Then a variable renamed version of the transition

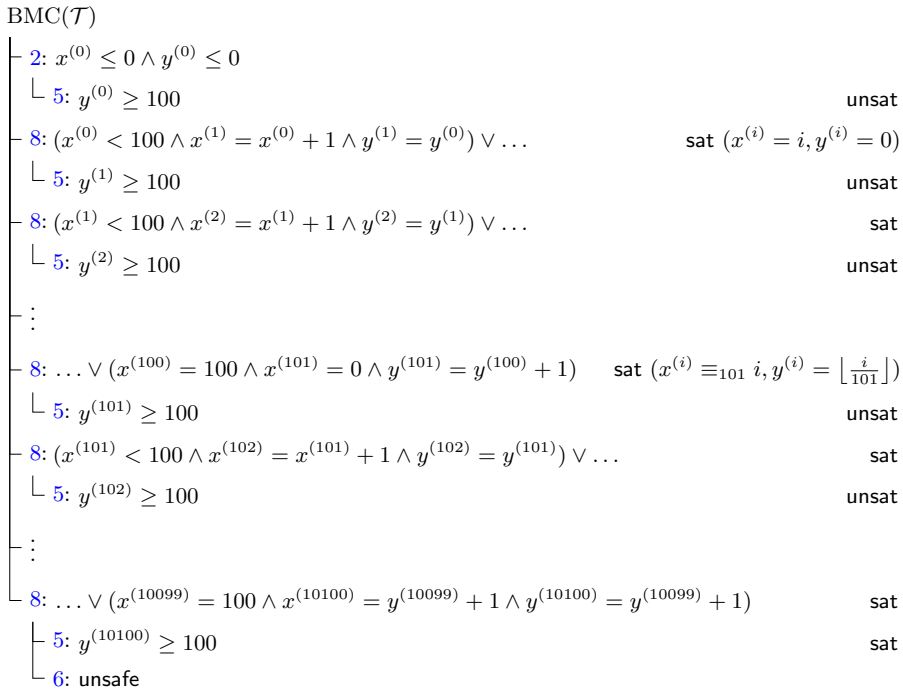


Fig. 1: Running BMC on Ex. 1

formula τ is added to the solver (Line 8). If doing so results in an unsatisfiable SMT problem, then the whole search space has been exhausted without proving unsafety, i.e., the problem is safe (Line 9). Otherwise, we enter the next iteration.

Example 3 (BMC). Fig. 1 illustrates how to prove unsafety of $\mathcal{T} = (\psi_{\mathsf{init}}, \tau, \psi_{\mathsf{err}})$ from Ex. 1 with BMC, where we show the formulas that are added to the SMT problem alongside the respective line numbers and depict push and pop by increasing and reducing the indentation. The keywords sat and unsat indicate the result of the subsequent call to the SMT solver, where sat is followed by a suitable model. If the model is not given explicitly, then it is analogous to the previous model.

With the given models, the first disjunct of τ applies in the first 100 iterations. Then we have $x^{(100)} = 100$, so the second disjunct of τ applies once and we get $y^{(101)} = y^{(100)} + 1$. In the given model, $x^{(i)} \equiv_{101} i$ means that $x^{(i)}$ and i are equivalent modulo 101. Then the first disjunct applies again 100 times, until the second disjunct applies again, ... After 100 applications of the second disjunct (and thus a total of 10100 steps), we get $y^{(10100)} = 100$, so that unsafety can be proven.

3.2 Accelerated Bounded Model Checking

To incorporate acceleration into BMC, we have to bridge the gap between the transition formula τ (which is usually disjunctive) and acceleration techniques,

which require conjunctive transition formulas. To this end, we use *syntactic implicants*.

Definition 4 (Syntactic Implicant Projection [14]). *Let $\tau \in \mathsf{QF}(\Sigma)$ be in NNF and let σ be a model of τ. We define:*

$$\mathsf{sip}(\tau, \sigma) := \bigwedge \{\lambda \mid \lambda \text{ is a literal of } \tau,\ \sigma \models \lambda\}$$
$$\mathsf{sip}(\tau) := \{\mathsf{sip}(\tau, \sigma) \mid \sigma \models \tau\}$$

We refer to the elements of $\mathsf{sip}(\tau)$ as syntactic implicants.

Since τ is in NNF, $\mathsf{sip}(\tau, \sigma)$ implies τ, and it is easy to see that $\tau \equiv \bigvee \mathsf{sip}(\tau)$. Whenever the call to the SMT solver in Line 9 of Alg. 1 yields sat, the resulting model gives rise to a sequence of syntactic implicants, called the *trace*. To define the trace formally, note that when we integrate acceleration into BMC, we may not only add τ to the SMT formula as in Line 8, but also *learned transitions* that result from acceleration. Thus, the following definition allows for changing the transition formula. In the sequel, $\circ$ also denotes composition of substitutions, i.e.,

$$\theta' \circ \theta := [x/\theta'(\theta(x)) \mid x \in \mathrm{dom}(\theta') \cup \mathrm{dom}(\theta)].$$

Definition 5 (Trace). *Let $[\tau_i]_{i=0}^{b-1}$ be a sequence of transition formulas and let σ be a model of $\bigwedge_{i=0}^{b-1} \mu_i(\tau_i)$. Then the* trace induced by σ *is*

$$\mathsf{trace}_b(\sigma, [\tau_i]_{i=0}^{b-1}) := [\mathsf{sip}(\tau_i, \sigma \circ \mu_i)]_{i=0}^{b-1}.$$

We write $\mathsf{trace}_b(\sigma)$ instead of $\mathsf{trace}_b(\sigma, [\tau_i]_{i=0}^{b-1})$ if $[\tau_i]_{i=0}^{b-1}$ is clear from the context.

So each model σ of $\bigwedge_{i=0}^{b-1} \mu_i(\tau_i)$ corresponds to a sequence of steps with the relations $\to_{\tau_0}, \to_{\tau_1}, \ldots, \to_{\tau_{b-1}}$, and the trace induced by σ contains the syntactic implicants of τ_i that were used in this sequence.

Example 6 (Trace). Reconsider Ex. 3. After two steps, the SMT problem consists of the following formulas:

$$x^{(0)} \leq 0 \wedge y^{(0)} \leq 0 \qquad (\psi_{\mathsf{init}})$$
$$(x^{(0)} < 100 \wedge x^{(1)} = x^{(0)} + 1 \wedge y^{(1)} = y^{(0)}) \vee (x^{(0)} = 100 \wedge x^{(1)} = 0 \wedge y^{(1)} = y^{(0)} + 1) \qquad (\tau)$$
$$(x^{(1)} < 100 \wedge x^{(2)} = x^{(1)} + 1 \wedge y^{(2)} = y^{(1)}) \vee (x^{(1)} = 100 \wedge x^{(2)} = 0 \wedge y^{(2)} = y^{(1)} + 1) \qquad (\tau)$$

With the model $\sigma = [x^{(i)}/i, y^{(i)}/0 \mid 0 \leq i \leq 2]$, we get

$$\mathsf{sip}(\tau, \sigma \circ [x/x^{(0)}, y/y^{(0)}, x'/x^{(1)}, y'/y^{(1)}]) = \mathsf{sip}(\tau, [x/0, y/0, x'/1, y'/0]) = \tau_{x<100}$$
$$\mathsf{sip}(\tau, \sigma \circ [x/x^{(1)}, y/y^{(1)}, x'/x^{(2)}, y'/y^{(2)}]) = \mathsf{sip}(\tau, [x/1, y/0, x'/2, y'/0]) = \tau_{x<100}$$

and thus $\mathsf{trace}_2(\sigma) = [\tau_{x<100}, \tau_{x<100}]$.

Algorithm 2: ABMC

Input: a safety problem $\mathcal{T} = (\psi_{\mathsf{init}}, \tau, \psi_{\mathsf{err}})$

```
1  b ← 0; V ← ∅; E ← ∅
2  add(μ_b(ψ_init))
3  if check_sat() = unsat then return safe
4  while ⊤ do
5  |  push(); add(μ_b(ψ_err))
6  |  if check_sat() = sat then return unsafe
7  |  pop()
8  |  σ ← get_model(); τ⃗ ← trace_b(σ)
9  |  V ← V ∪ τ⃗; E ← E ∪ {(τ1, τ2) | [τ1, τ2] is an infix of τ⃗}
10 |  if τ⃗ = π⃗ :: π⃗^⟳ ∧ π⃗^⟳ is (V, E)-cyclic ∧ should_accel(π⃗^⟳) then
11 |  |  add(μ_b(τ ∨ accel(π⃗^⟳)))
12 |  else
13 |  |  add(μ_b(τ))
14 |  if check_sat() = unsat then return safe
15 |  b ← b + 1
```

To detect situations where applying acceleration techniques pays off, we need to distinguish "recursive" from "non-recursive" traces. Since transition formulas are unstructured, the usual techniques for detecting recursion (based on, e.g., program syntax or control flow graphs) do not apply in our setting. Instead, we rely on the *dependency graph* of the transition formula.

Definition 7 (Dependency Graph). *Let τ be a transition formula. Its* dependency graph $\mathcal{DG} = (V, E)$ *is a directed graph whose vertices $V := \mathsf{sip}(\tau)$ are τ's syntactic implicants, and we have $\tau_1 \to \tau_2 \in E$ if $\circledcirc(\tau_1, \tau_2)$ is satisfiable. We call $\vec{\tau} \in \mathsf{sip}(\tau)^c$ $\mathcal{DG}$-*cyclic *if $c > 0$ and $(\tau_1 \to \tau_2), \ldots, (\tau_{c-1} \to \tau_c), (\tau_c \to \tau_1) \in E$.*

So intuitively, the syntactic implicants correspond to the different cases of $\to_\tau$, and τ's dependency graph corresponds to the control flow graph of $\to_\tau$.

Example 8 (Dependency Graph). For Ex. 1, the dependency graph is

$$(\{\tau_{x<100}, \tau_{x=100}\}, \{\tau_{x<100} \to \tau_{x<100}, \tau_{x<100} \to \tau_{x=100}, \tau_{x=100} \to \tau_{x<100}\}).$$

However, as the size of $\mathsf{sip}(\tau)$ is worst-case exponential in the number of disjunctions in τ, we do not compute τ's dependency graph eagerly. Instead, ABMC maintains an under-approximation, i.e., a subgraph $\mathcal{G}$ of the dependency graph, which is extended whenever two transitions that are not yet connected by an edge occur consecutively on the trace. As soon as a $\mathcal{G}$-cyclic suffix $\vec{\tau}^{\circlearrowleft}$ is detected on the trace, we may accelerate it. Therefore, the trace may also contain the learned transition $\mathsf{accel}(\vec{\tau}^{\circlearrowleft})$ in subsequent iterations. Hence, to detect cyclic suffixes that contain learned transitions, they have to be represented in $\mathcal{G}$ as well. Thus, $\mathcal{G}$ is in fact a subgraph of the dependency graph of $\tau \vee \bigvee \mathcal{L}$, where $\mathcal{L}$ is the set of all transitions that have been learned so far.

This gives rise to the ABMC algorithm, which is shown in Alg. 2. In the following, we just write "cyclic" instead of (V, E)-cyclic. The difference to Alg. 1 can be seen in Lines 8–11. First, the trace is constructed from the current model

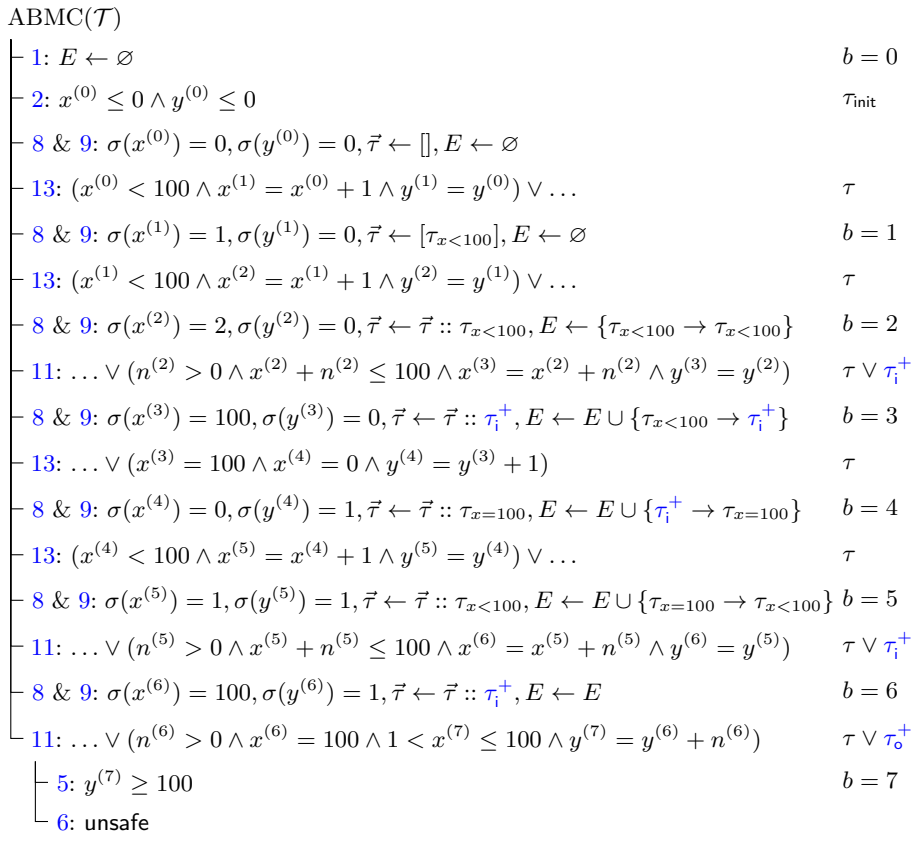


Fig. 2: Running ABMC on Ex. 1

(Line 8). Then, the approximation of the dependency graph is refined such that it contains vertices for all elements of the trace, and edges for all transitions that occur consecutively on the trace (Line 9). If the trace has a cyclic suffix (Line 10), then it may get accelerated (Line 11), provided that the call to should_accel (which will be discussed in more detail in Sect. 3.3) returns $\top$. In this way, in the next iteration the SMT solver can choose a model that satisfies $\mathsf{accel}(\vec{\pi}^{\circlearrowleft})$ and thus simulates several instead of just one $\to_\tau$-step. Note, however, that we do *not* update τ with $\tau \vee \mathsf{accel}(\vec{\pi}^{\circlearrowleft})$. So in every iteration, at most one learned transition is added to the SMT problem. In this way, we avoid blowing up τ unnecessarily.

Fig. 2 shows a run of Alg. 2 on Ex. 1 (where we skip calls to the SMT solver that yield unsat in Line 6 for brevity). For simplicity, we assume that should_accel always returns $\top$, and the model σ is only extended in each step, i.e., $\sigma(x^{(i)})$ and $\sigma(y^{(i)})$ remain unchanged for all $0 \leq i < b$. In general, the SMT solver can choose different values for $\sigma(x^{(i)})$ and $\sigma(y^{(i)})$ in every iteration. On the right, we show the current bound b, as well as the formulas that are added to the SMT problem (after renaming their variables suitably with μ_b). Initially, (the approximation of) the dependency graph $\mathcal{G} = (V, E)$ is empty. When $b = 2$, the

trace is $[\tau_{x<100}, \tau_{x<100}]$, and the corresponding edge is added to $\mathcal{G}$. Thus, the trace has the cyclic suffix $\tau_{x<100}$ and we accelerate it, resulting in τ_{i}^+, which is added to the SMT problem. Then we obtain the trace $[\tau_{x<100}, \tau_{x<100}, \tau_{\mathsf{i}}^+]$, and the edge $\tau_{x<100} \to \tau_{\mathsf{i}}^+$ is added to $\mathcal{G}$. Note that Alg. 2 does not enforce the use of τ_{i}^+, so τ might still be unrolled thousands of times instead, depending on the models found by the SMT solver. We will address this issue in Sect. 4.

Next, $\tau_{x=100}$ already applies with $b = 4$ (whereas it only applied with $b = 100$ in Ex. 3). So the trace is $[\tau_{x<100}, \tau_{x<100}, \tau_{\mathsf{i}}^+, \tau_{x=100}]$, and the edge $\tau_{\mathsf{i}}^+ \to \tau_{x=100}$ is added to $\mathcal{G}$. Then we obtain the trace $[\tau_{x<100}, \tau_{x<100}, \tau_{\mathsf{i}}^+, \tau_{x=100}, \tau_{x<100}]$, and add $\tau_{x=100} \to \tau_{x<100}$ to $\mathcal{G}$. Since the suffix $\tau_{x<100}$ is again cyclic, we accelerate it and add τ_{i}^+ to the SMT problem. After one more step, the trace $[\tau_{x<100}, \tau_{x<100}, \tau_{\mathsf{i}}^+, \tau_{x=100}, \tau_{x<100}, \tau_{\mathsf{i}}^+]$ has the cyclic suffix $[\tau_{x=100}, \tau_{x<100}, \tau_{\mathsf{i}}^+]$. Accelerating it yields τ_{o}^+, which is added to the SMT problem. Afterwards, unsafety can be proven with $b = 7$.

Since using acceleration is just a heuristic to speed up BMC, all basic properties of BMC immediately carry over to ABMC (assuming a complete SMT solver).

Theorem 9 (Properties of ABMC). *ABMC is*
Refutationally Complete: *If* $\mathcal{T}$ *is unsafe, then* $\mathsf{ABMC}(\mathcal{T})$ *returns* unsafe.
Sound: *If* $\mathsf{ABMC}(\mathcal{T})$ *returns* safe, *then* $\mathcal{T}$ *is safe.*
If $\mathsf{ABMC}(\mathcal{T})$ *returns* unsafe, *then* $\mathcal{T}$ *is unsafe.*
Non-Terminating: *If* $\mathcal{T}$ *is safe, then* $\mathsf{ABMC}(\mathcal{T})$ *may not terminate.*

3.3 Fine Tuning Acceleration

We now turn our attention to $\mathsf{should\_accel}$. First, acceleration should be applied to cyclic suffixes consisting of a single *original* (i.e., non-learned) transition.

Requirement 1. $\mathsf{should\_accel}([\pi]) = \top$ *if* $\pi \in \mathsf{sip}(\tau)$.

However, applying acceleration to a single learned transition is pointless, as

$$\to_{\mathsf{accel}(\mathsf{accel}(\tau))} = \to^+_{\mathsf{accel}(\tau)} = (\to_\tau^+)^+ = \to_\tau^+ = \to_{\mathsf{accel}(\tau)}.$$

Requirement 2. $\mathsf{should\_accel}([\pi]) = \bot$ *if* $\pi \notin \mathsf{sip}(\tau)$.

Next, for every cyclic sequence $\vec{\pi}$, we have

$$\to_{\mathsf{accel}(\vec{\pi}::\mathsf{accel}(\vec{\pi}))} = \to^+_{\vec{\pi}::\mathsf{accel}(\vec{\pi})} = (\to_{\vec{\pi}} \circ \to_{\mathsf{accel}(\vec{\pi})})^+ = (\to_{\vec{\pi}} \circ \to^+_{\vec{\pi}})^+$$
$$= \to_{\vec{\pi}} \circ \to^+_{\vec{\pi}} = \to_{\vec{\pi}} \circ \to_{\mathsf{accel}(\vec{\pi})} = \to_{\vec{\pi}::\mathsf{accel}(\vec{\pi})},$$

and thus accelerating $\vec{\pi} :: \mathsf{accel}(\vec{\pi})$ is pointless, too. Hence, we obtain:

Requirement 3. $\mathsf{should\_accel}(\vec{\pi} :: \mathsf{accel}(\vec{\pi})) = \bot$.

However, Req. 3 is too specific, as it, e.g., does not prevent acceleration of other sequences $\vec{\pi}_2 :: \mathsf{accel}(\vec{\pi}) :: \vec{\pi}_1$ where $\vec{\pi} = \vec{\pi}_1 :: \vec{\pi}_2$. For such sequences, we have

$$\to^2_{\vec{\pi}_2::\mathsf{accel}(\vec{\pi})::\vec{\pi}_1} = \to_{\vec{\pi}_2::\mathsf{accel}(\vec{\pi})::\vec{\pi}::\mathsf{accel}(\vec{\pi})::\vec{\pi}_1} \subseteq \to_{\vec{\pi}_2::\mathsf{accel}(\vec{\pi})::\vec{\pi}_1}$$

and thus $\to_{\mathsf{accel}(\vec{\pi}_2::\mathsf{accel}(\vec{\pi})::\vec{\pi}_1)} = \to^+_{\vec{\pi}_2::\mathsf{accel}(\vec{\pi})::\vec{\pi}_1} = \to_{\vec{\pi}_2::\mathsf{accel}(\vec{\pi})::\vec{\pi}_1}$, so accelerating them is pointless, too. Thus in general, the problem is that the cyclic suffix of the trace might consist of a cycle $\vec{\pi}$ and $\mathsf{accel}(\vec{\pi})$, but it might not necessarily start with either of them. Hence, we generalize Req. 3 using the notion of *conjugates*.

Definition 10 (Conjugate). *We say that two vectors* $\vec{v}, \vec{w}$ *are* conjugates *(denoted* $\vec{v} \equiv_\circ \vec{w}$*) if* $\vec{v} = \vec{v}_1 :: \vec{v}_2$ *and* $\vec{w} = \vec{v}_2 :: \vec{v}_1$*.*

So a conjugate of a cycle corresponds to the same cycle with another entry point.

Requirement 4. $\mathsf{should\_accel}(\vec{\pi}') = \bot$ *if* $\vec{\pi}' \equiv_\circ \vec{\pi} :: \mathsf{accel}(\vec{\pi})$ *for some* $\vec{\pi}$*.*

In general, however, we also want to accelerate cyclic suffixes that contain learned transitions to deal with nested loops, as in the last acceleration step of Fig. 2.

Requirement 5. $\mathsf{should\_accel}(\vec{\pi}') = \top$ *if* $\vec{\pi}' \not\equiv_\circ \vec{\pi} :: \mathsf{accel}(\vec{\pi})$ *for all* $\vec{\pi}$*.*

Req. 1, 2, 4, and 5 give rise to a complete specification for should_accel: If the cyclic suffix is a singleton, the decision is made based on Requirements 1 and 2, and otherwise the decision is made based on Requirements 4 and 5.

$$\mathsf{should\_accel}(\vec{\pi}') := (|\vec{\pi}'| = 1 \wedge \vec{\pi}' \in \mathsf{sip}(\tau)) \vee (|\vec{\pi}'| > 1 \wedge \forall \vec{\pi}.\ (\vec{\pi}' \not\equiv_\circ \vec{\pi} :: \mathsf{accel}(\vec{\pi})))$$

However, this specification misses one important case: Recall that the trace was $[\tau_{x<100}, \tau_{x<100}]$ before acceleration was applied for the first time in Fig. 2. While both $[\tau_{x<100}]$ and $[\tau_{x<100}, \tau_{x<100}]$ are cyclic, the latter should not be accelerated, since $\mathsf{accel}([\tau_{x<100}, \tau_{x<100}])$ is a special case of τ_{i}^+ that only represents an even number of steps with $\tau_{x<100}$. Here, the problem is that the cyclic suffix contains a *square*, i.e., two adjacent repetitions of the same non-empty sub-sequence.

Requirement 6. $\mathsf{should\_accel}(\vec{\pi}) = \bot$ *if* $\vec{\pi}$ *contains a square.*

Thus, we obtain the following specification for should_accel:

$$\begin{aligned}\mathsf{should\_accel}(\vec{\pi}') := {} & |\vec{\pi}'| = 1 \wedge \vec{\pi}' \in \mathsf{sip}(\tau) \quad \vee \\ & |\vec{\pi}'| > 1 \wedge \vec{\pi}' \text{ is square-free} \wedge \forall \vec{\pi}.\ (\vec{\pi}' \not\equiv_\circ \vec{\pi} :: \mathsf{accel}(\vec{\pi}))\end{aligned}$$

All properties that are required to implement should_accel can easily be checked automatically. To check $\vec{\pi}' \not\equiv_\circ \vec{\pi} :: \mathsf{accel}(\vec{\pi})$, our implementation maintains a map from learned transitions to the corresponding cycles that have been accelerated.

However, to implement Alg. 2, there is one more missing piece: As the choice of the cyclic suffix in Line 10 is non-deterministic, a heuristic for choosing it is required. In our implementation, we choose the *shortest* cyclic suffix such that should_accel returns $\top$. The reason is that, as observed in [14], accelerating short cyclic suffixes before longer ones allows for learning more general transitions.

4 Guiding ABMC with Blocking Clauses

As mentioned in Sect. 3.2, Alg. 2 does not enforce the use of learned transitions. Thus, depending on the models found by the SMT solver, ABMC may behave just like BMC. In this section, we improve ABMC by integrating *blocking clauses* that prevent it from unrolling loops instead of using learned transitions.

Blocking clauses exploit the following straightforward observation: If the learned transition $\tau_\ell = \mathsf{accel}(\vec{\pi}^{\circlearrowleft})$ has been added to the SMT problem with bound b and an error state can be reached via a trace with prefix

$$\vec{\pi} = [\tau_0, \ldots, \tau_{b-1}] :: \vec{\pi}^{\circlearrowleft} \qquad \text{or} \qquad \vec{\pi}' = [\tau_0, \ldots, \tau_{b-1}, \tau_\ell] :: \vec{\pi}^{\circlearrowleft},$$

then an error state can also be reached via a trace with the prefix $[\tau_0, \ldots, \tau_{b-1}, \tau_\ell]$, which is not continued with $\vec{\pi}^{\circlearrowleft}$. Thus, we may remove traces of the form $\vec{\pi}$ and $\vec{\pi}'$ from the search space by modifying the SMT problem accordingly.

To do so, we assign a unique identifier to each learned transition, and we introduce a fresh integer-valued variable ℓ which is set to the corresponding identifier whenever a learned transition is used, and to 0, otherwise.

Example 11 (Blocking Clauses). Reconsider Fig. 2 and assume that we modify τ by conjoining $\ell = 0$, and τ_{i}^+ by conjoining $\ell = 1$. Thus, we now have

$$\tau_{x<100} \equiv x < 100 \wedge x' = x + 1 \wedge y' = y \wedge \ell = 0 \qquad \text{and}$$
$$\tau_{\mathsf{i}}^+ \equiv n > 0 \wedge x + n \leq 100 \wedge x' = x + n \wedge y' = y \wedge \ell = 1.$$

When $b = 2$, the trace is $[\tau_{x<100}, \tau_{x<100}]$, and in the next iteration, it may be extended to either $\vec{\pi} = [\tau_{x<100}, \tau_{x<100}, \tau_{x<100}]$ or $\vec{\tau} = [\tau_{x<100}, \tau_{x<100}, \tau_{\mathsf{i}}^+]$. However, as $\rightarrow_{\tau_{\mathsf{i}}^+} = \rightarrow^+_{\tau_{x<100}}$, we have $\rightarrow_{\vec{\pi}} \subseteq \rightarrow_{\vec{\tau}}$, so the entire search space can be covered without considering the trace $\vec{\pi}$. Thus, we add the blocking clause

$$\neg\mu_2(\tau_{x<100}) \tag{β_1}$$

to the SMT problem to prevent ABMC from finding a model that gives rise to the trace $\vec{\pi}$. Note that we have $\mu_2(\tau_{\mathsf{i}}^+) \models \beta_1$, as $\tau_{x<100} \models \ell = 0$ and $\tau_{\mathsf{i}}^+ \models \ell \neq 0$. Thus, β_1 blocks $\tau_{x<100}$ for the third step, but τ_{i}^+ can still be used without restrictions. Therefore, adding β_1 to the SMT problem does not prevent us from covering the entire search space.

Similarly, we have $\rightarrow_{\vec{\pi}'} \subseteq \rightarrow_{\vec{\tau}}$ for $\vec{\pi}' = [\tau_{x<100}, \tau_{x<100}, \tau_{\mathsf{i}}^+, \tau_{x<100}]$. Thus, we also add the following blocking clause to the SMT problem:

$$\ell^{(2)} \neq 1 \vee \neg\mu_3(\tau_{x<100}) \tag{β_2}$$

ABMC with blocking clauses can be seen in Alg. 3. The counter id is used to obtain unique identifiers for learned transitions. Thus, it is initialized with 0 (Line 1) and incremented whenever a new transition is learned (Line 14). Moreover, as explained above, $\ell = 0$ is conjoined to τ (Line 1), and $\ell = \mathsf{id}$ is conjoined to each learned transition (Line 14).

In Lines 15 and 16, the blocking clauses corresponding to the superfluous traces $\vec{\pi}$ and $\vec{\pi}'$ above are created, and they are added to the SMT problem in Line 17. Here, $\pi_i^{\circlearrowleft}$ denotes the i^{th} transition in the sequence $\vec{\pi}^{\circlearrowleft}$.

Importantly, Alg. 3 caches (Line 14) and reuses (Line 12) learned transitions. In this way, the learned transitions that are conjoined to the SMT problem have the same id if they stem from the same cycle, and thus the blocking clauses β_1 and β_2 can also block sequences $\vec{\pi}^{\circlearrowleft}$ that contain learned transitions.

Example 12 (Caching). Consider a transition formula whose dependency graph is (V, E) where $V = \{\tau_1, \tau_2, \tau_3\}$ and $E = \{\tau_1 \rightarrow \tau_2, \tau_2 \rightarrow \tau_2, \tau_2 \rightarrow \tau_3, \tau_3 \rightarrow \tau_1, \tau_3 \rightarrow \tau_2\}$. As Alg. 3 conjoins $\ell = 0$ to τ, assume $\tau_i \models \ell = 0$ for all $i \in \{1, 2, 3\}$. Moreover, assume that accelerating τ_2 yields τ_2^+ with $\tau_2^+ \models \ell = 1$. If we obtain the trace $[\tau_1, \tau_2^+, \tau_3]$, it can be accelerated. Thus, Alg. 3 would add

Algorithm 3: $\mathsf{ABMC}_{\mathsf{block}}$

Input: a safety problem $\mathcal{T} = (\psi_{\mathsf{init}}, \tau, \psi_{\mathsf{err}})$

$b \leftarrow 0$; $V \leftarrow \varnothing$; $E \leftarrow \varnothing$; $\mathsf{id} \leftarrow 0$; $\tau \leftarrow \tau \wedge \ell = 0$; $\mathsf{cache} \leftarrow \varnothing$
$\mathsf{add}(\mu_b(\psi_{\mathsf{init}}))$
if $\mathsf{check\_sat}() = \mathsf{unsat}$ **then return** safe
while $\top$ **do**
  $\mathsf{push}()$; $\mathsf{add}(\mu_b(\psi_{\mathsf{err}}))$
  if $\mathsf{check\_sat}() = \mathsf{sat}$ **then return** unsafe
  $\mathsf{pop}()$
  $\sigma \leftarrow \mathsf{get\_model}()$; $\vec{\tau} \leftarrow \mathsf{trace}_b(\sigma)$
  $V \leftarrow V \cup \vec{\tau}$; $E \leftarrow E \cup \{(\tau_1, \tau_2) \mid [\tau_1, \tau_2] \text{ is an infix of } \vec{\tau}\}$
  if $\vec{\tau} = \vec{\pi} :: \vec{\pi}^{\circlearrowleft} \wedge \vec{\pi}^{\circlearrowleft}$ *is* (V, E)*-cyclic* $\wedge\, \mathsf{should\_accel}(\vec{\pi}^{\circlearrowleft})$ **then**
    if $\exists \tau_c.\ (\vec{\pi}^{\circlearrowleft}, \tau_c) \in \mathsf{cache}$ **then**
      $\tau_\ell \leftarrow \tau_c$
    else
      $\mathsf{id} \leftarrow \mathsf{id} + 1$; $\tau_\ell \leftarrow \mathsf{accel}(\vec{\pi}^{\circlearrowleft}) \wedge \ell = \mathsf{id}$; $\mathsf{cache} \leftarrow \mathsf{cache} \cup \{(\vec{\pi}^{\circlearrowleft}, \tau_\ell)\}$
    $\beta_1 \leftarrow \neg \left(\bigwedge_{i=0}^{|\vec{\pi}^{\circlearrowleft}|-1} \mu_{b+i}(\pi_i^{\circlearrowleft}) \right)$
    $\beta_2 \leftarrow \ell^{(b)} \neq \mathsf{id} \vee \neg \left(\bigwedge_{i=0}^{|\vec{\pi}^{\circlearrowleft}|-1} \mu_{b+i+1}(\pi_i^{\circlearrowleft}) \right)$
    $\mathsf{add}(\mu_b(\tau \vee \tau_\ell) \wedge \beta_1 \wedge \beta_2)$
  else
    $\mathsf{add}(\mu_b(\tau))$
  if $\mathsf{check\_sat}() = \mathsf{unsat}$ **then return** safe
  $b \leftarrow b + 1$

$$\beta_1 \equiv \neg \left(\mu_3(\tau_1) \wedge \mu_4(\tau_2^+) \wedge \mu_5(\tau_3) \right)$$

to the SMT problem. If the next step yields the trace $[\tau_1, \tau_2^+, \tau_3, \tau_2]$, then τ_2 is accelerated again. Without caching, acceleration may yield a new transition $\tau_{2'}^+$ with $\tau_{2'}^+ \models \ell = 2$. As the SMT solver may choose a different model in every iteration, the trace may also change in every iteration. So after two more steps, we could get the trace $[\tau_1, \tau_2^+, \tau_3, \tau_1, \tau_{2'}^+, \tau_3]$. At this point, the "outer" loop consisting of τ_1, arbitrarily many repetitions of τ_2, and τ_3, has been unrolled a second time, which should have been prevented by β_1. The reason is that $\tau_2^+ \models \ell = 1$, whereas $\tau_{2'}^+ \models \ell = 2$, and thus $\tau_{2'}^+ \models \neg \tau_2^+$. With caching, we again obtain τ_2^+ when τ_2 is accelerated for the second time, such that this problem is avoided.

Remarkably, blocking clauses allow us to prove safety in cases where BMC fails.

Example 13 (Proving Safety with Blocking Clauses). Consider the safety problem $(x \leq 0, \tau, x > 100)$ with $\tau \equiv x < 100 \wedge x' = x + 1$. Alg. 1 cannot prove its safety, as τ can be unrolled arbitrarily often (by choosing smaller and smaller initial values for x). With Alg. 3, we obtain the following SMT problem with $b = 3$.

$$\mu_0(x \leq 0) \qquad \text{(initial states)}$$
$$\mu_0(\tau \wedge \ell = 0) \qquad (\tau)$$
$$\mu_1(\tau \wedge \ell = 0) \qquad (\tau)$$
$$\neg \mu_2(\tau \wedge \ell = 0) \qquad (\beta_1)$$

$$\ell^{(2)} \neq 1 \vee \neg\mu_3(\tau \wedge \ell = 0) \qquad (\beta_2)$$
$$\mu_2((\tau \wedge \ell = 0) \vee (n > 0 \wedge x + n \leq 100 \wedge x' = x + n \wedge \ell = 1)) \quad (\tau \vee \mathsf{accel}(\tau))$$
$$\mu_3(\tau \wedge \ell = 0) \qquad (\tau)$$

From the last formula and β_2, we get $\ell^{(2)} \neq 1$. On the other hand, the formula labeled with $(\tau \vee \mathsf{accel}(\tau))$ and β_1 imply $\mu_2(\ell = 1) \equiv \ell^{(2)} = 1$, resulting in a contradiction. Thus, $\mathsf{ABMC}_{\mathsf{block}}$ can prove safety with the bound $b = 3$.

Like ABMC, $\mathsf{ABMC}_{\mathsf{block}}$ preserves all important properties of BMC.

Theorem 14 (Properties of $\mathsf{ABMC}_{\mathsf{block}}$). $\mathsf{ABMC}_{\mathsf{block}}$ *is*
Refutationally Complete: *If* $\mathcal{T}$ *is unsafe, then* $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *returns* unsafe.
Sound: *If* $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *returns* safe*, then* $\mathcal{T}$ *is safe.*
If $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *returns* unsafe*, then* $\mathcal{T}$ *is unsafe.*
Non-Terminating: *If* $\mathcal{T}$ *is safe, then* $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *may not terminate.*

Proof. A detailed proof is provided in App. A.

5 Related Work and Experiments

We presented ABMC, an adaption of bounded model checking that makes use of acceleration techniques. By enabling BMC to find deep counterexamples, it targets one of the main limitations of BMC. However, without further precautions, ABMC is not guaranteed to make use of the transitions that result from applying acceleration, since whether they are used or not depends on the models found by the underlying SMT solver. Hence, we introduced *blocking clauses* to enforce the use of accelerated transitions in cases where doing so does not prevent us from exploring the entire search space. In this way, blocking clauses also enable ABMC to prove safety in cases where BMC fails to do so.

Related Work There is a large body of literature on bounded model checking that is concerned with encoding temporal logic specifications into propositional logic, see [3, 4] as starting points. This line of work is clearly orthogonal to ours.

Moreover, numerous techniques focus on proving *safety* or *satisfiability* of transition systems or CHCs, respectively (e.g., [9, 10, 15, 17, 18, 22]). A comprehensive overview is beyond the scope of this paper. Instead, we focus on techniques that, like ABMC, aim to prove unsafety by finding long counterexamples.

The most closely related approach is *Acceleration Driven Clause Learning* (ADCL) [13, 14]. The main difference between ABMC and ADCL is that ABMC performs breadth-first search, whereas ADCL performs depth-first search. Thus, ADCL requires a mechanism for backtracking to avoid getting stuck. To this end, ADCL relies on a notion of *redundancy*, which is difficult to automate. Thus, in practice, approximations are used [14, Sect. 4]. However, even with a complete redundancy check, ADCL might get stuck in a safe part of the search space [14, Thm. 18]. ABMC does not suffer from such deficits.

Our heuristics for deciding when to perform acceleration (see Sect. 3.3) rule out certain redundant traces. However, while avoiding redundancies is crucial for ADCL, it is a mere optimization for ABMC.

Both ADCL and ABMC rely on notions of *blocking clauses*. The main goal of ADCL's blocking clauses is to avoid visiting the same part of the search space twice. Due to its breadth-first strategy, ABMC does not suffer from this problem. Instead, ABMC's blocking clauses again rule out certain redundant traces.

On the other hand, ADCL has successfully been adapted for proving non-termination [13], and it is unclear whether a corresponding adaption of ABMC would be competitive. Furthermore, ADCL enumerates traces in a more systematic way, as the trace considered by ABMC may change in every iteration of its main loop, depending on the model that is found by the SMT solver. Thus, ADCL is advantageous for examples with deeply nested loops, where ABMC may require many steps until the SMT solver yields models that allow for accelerating the nested loops one after the other. Thus, both techniques are orthogonal.

Several other acceleration-based approaches [2, 7, 12] can be seen as generalizations of the classical state elimination method for finite automata: Instead of transforming finite automata to regular expressions, they transform transition systems to formulas that represent the runs of the transition system. During this transformation, acceleration is the counterpart to the Kleene star in the state elimination method. Clearly, these approaches differ fundamentally from ours.

In [19], under-approximating acceleration techniques are used to enrich the control-flow graph of C programs in order to find deep counterexamples. While [19] relies on external model checkers for finding counterexamples, we directly integrated acceleration into BMC.

Another related approach is described in [16], where acceleration is integrated into a CEGAR loop in two ways: (1) as preprocessing and (2) to generalize interpolants. In contrast to (1), we use acceleration "on the fly". In contrast to (2), we do not use abstractions, so our learned transitions can directly be used in counterexamples. Moreover, [16] only applies acceleration to conjunctive transition formulas, whereas we accelerate conjunctive variants of arbitrary transition formulas. So in our approach, acceleration techniques are applicable more often, which is particularly useful for finding long counterexamples.

Finally, *transition power abstraction* (TPA) [5] computes a sequence of over-approximations for transition systems where the n^{th} element captures 2^n instead of just n steps of the transition relation. So like ABMC, TPA can help to find long refutations quickly, but in contrast to ABMC, TPA relies on over-approximations.

Experiments We implemented ABMC in the tool LoAT, where our implementation is currently restricted to integer arithmetic. Thus, to evaluate our approach, we used the examples from the category LIA-Lin (linear CHCs with linear integer arithmetic) from the CHC competitions '22 and '23 [8], which contain numerous CHC problems resulting from actual program verification tasks.

We compared several configurations of LoAT with other leading CHC solvers. More precisely, we evaluated the following configurations:

LoAT We used LoAT's implementations of Alg. 1 (LoAT BMC), Alg. 2 (LoAT ABMC), and Alg. 3 (LoAT $\text{ABMC}_{\text{block}}$), as well as LoAT's implementation of ADCL (LoAT ADCL).

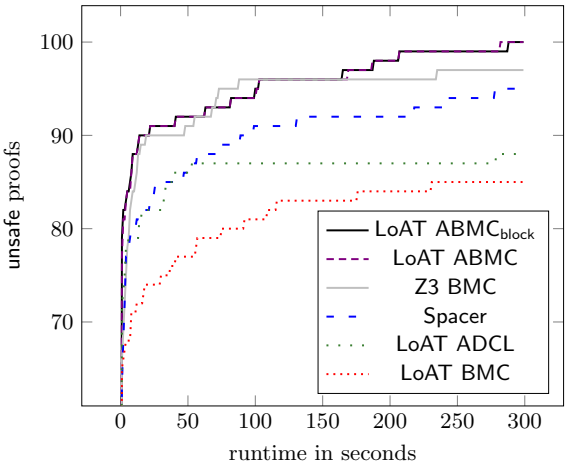
Z3 [20] We used Z3 4.12.2, where we evaluated its implementation of the Spacer algorithm (Spacer [18]) as well as its implementation of BMC (Z3 BMC).

Golem [5] We used Golem 0.4.3, where we evaluated its implementation of *transition power abstraction* (Golem TPA [5]) as well as its implementation of BMC (Golem BMC).

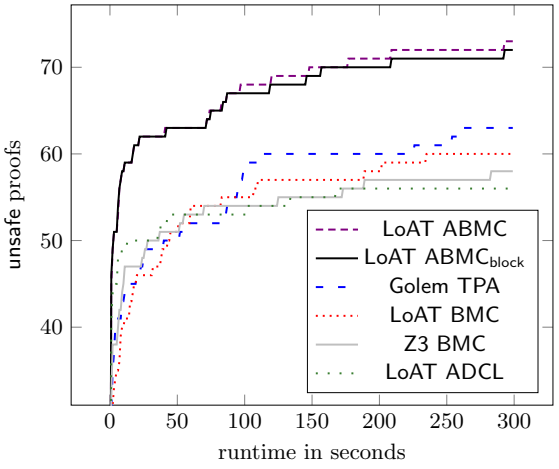
Eldarica [17] We used the default configuration of Eldarica 2.0.9.

We ran our experiments on StarExec [21] with a wallclock timeout of 300s, a cpu timeout of 1200s, and a memory limit of 128GB per example.

2022	unsafe		safe	
	✓	!	✓	!
LoAT ABMC$_{\text{block}}$	100	1	95	7
LoAT ABMC	100	–	53	–
Z3 BMC	97	3	29	–
Spacer	95	2	206	48
LoAT ADCL	88	0	0	–
LoAT BMC	85	–	61	0
Golem TPA	84	2	140	5
Golem BMC	80	–	31	–
Eldarica	59	0	175	17



2023	unsafe		safe	
	✓	!	✓	!
LoAT ABMC	73	–	31	–
LoAT ABMC$_{\text{block}}$	72	0	75	12
Golem TPA	63	5	88	3
LoAT BMC	60	–	36	0
Z3 BMC	58	1	21	–
LoAT ADCL	56	0	0	–
Golem BMC	55	–	20	–
Spacer	52	1	156	51
Eldarica	29	0	121	17



The results can be seen in the tables above, where the upper and lower one shows the results for the examples from the CHC competition '22 and '23, respectively. The columns marked with **!** show the number of unique proofs, i.e., the number of examples that could only be solved by the corresponding configuration. Clearly, such a comparison only makes sense if just one implementation of each algorithm is considered. Hence, we only considered LoAT ABMC$_{\text{block}}$, Z3 BMC,

Spacer, LoAT ADCL, Golem TPA, and Eldarica for unsafe instances; and only LoAT ABMC$_{\text{block}}$, LoAT BMC, Spacer, Golem TPA, and Eldarica for safe instances (LoAT ADCL cannot yet prove safety). We considered different implementations of BMC because, surprisingly, LoAT BMC outperforms the other BMC implementation on safe instances (whereas Z3 BMC is more powerful on the unsafe instances from '22). A look at the implementations of Golem BMC and Z3 BMC revealed that they do not return safe in the case that corresponds to Line 9 of Alg. 1. Thus, they only prove safety in corner cases (e.g., if τ_{init} is unsatisfiable).

The tables show that our implementation of ABMC is very powerful for proving unsafety. In particular, it shows a significant improvement over LoAT BMC, which is implemented very similarly, but does not make use of acceleration.

For unsafe examples from '22, Z3 BMC is more powerful than LoAT BMC, but it still solves fewer instances than ABMC. A manual review of examples where Z3 BMC outperforms LoAT indicated that this is due to Z3's pre-processings. So presumably, the results of LoAT can be improved further by enhancing its pre-processings.

Note that almost all unsafe instances that can be solved by ABMC can also be solved by other configurations. This is not surprising, as LoAT ADCL is also based on acceleration techniques. Hence, ABMC combines the strengths of ADCL and BMC, and conversely, unsafe examples that can be solved with ABMC can usually also be solved by one of these techniques. So for unsafe instances, the main contribution of ABMC is to have *one* technique that performs well both on instances with shallow counterexamples (which can be solved by BMC) as well as instances with deep counterexamples only (which can often be solved by ADCL).

Regarding safe examples, the tables show that our implementation of ABMC is not competitive with state-of-the-art techniques. However, it finds several unique proofs. This is remarkable, as LoAT is not at all fine-tuned for proving safety. For example, we expect that LoAT's results on safe instances can easily be improved by integrating over-approximating acceleration techniques. Then these techniques could be used in cases where LoAT's exact, but incomplete acceleration techniques fail. While such a variant of ABMC could not prove unsafety due to the use of over-approximations, it would presumably be much more powerful for proving safety. We leave that to future work.

The figures on the right show how many proofs of unsafety were found within a given runtime, where we only include the six best configurations for readability. They clearly show that ABMC is highly competitive on unsafe instances, not only in terms of solved examples, but also in terms of runtime.

Our results also show that blocking clauses have no significant impact on ABMC's performance on unsafe instances, neither regarding the number of solved examples, nor regarding the runtime. In fact, ABMC$_{\text{block}}$ solved one instance less than ABMC (which can, however, also be solved by ABMC$_{\text{block}}$ with a larger timeout). On the other hand, blocking clauses are clearly useful for proving safety, where they even allow LoAT to find several unique proofs.

Our implementation is open-source and available on Github. For the sources, a pre-compiled binary, and more information on our evaluation, we refer to [1].

References

1. Evaluation of "Accelerated Bounded Model Checking" (2023), https://loat-developers.github.io/abmc-eval/
2. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Acceleration from theory to practice. Int. J. Softw. Tools Technol. Transf. **10**(5), 401–424 (2008). https://doi.org/10.1007/s10009-008-0064-3
3. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003). https://doi.org/10.1016/S0065-2458(03)58003-2
4. Biere, A.: Bounded model checking. In: Handbook of Satisfiability - Second Edition, pp. 739–764. Frontiers in Artificial Intelligence and Applications 336, IOS Press (2021). https://doi.org/10.3233/FAIA201002
5. Blicha, M., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: Transition power abstractions for deep counterexample detection. In: TACAS '22. pp. 524–542. LNCS 13243 (2022). https://doi.org/10.1007/978-3-030-99524-9_29
6. 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
7. Bozga, M., Iosif, R., Konečný, F.: Relational analysis of integer programs. Tech. Rep. TR-2012-10, VERIMAG (2012), https://www-verimag.imag.fr/TR/TR-2012-10.pdf
8. CHC Competition, https://chc-comp.github.io
9. Dietsch, D., Heizmann, M., Hoenicke, J., Nutz, A., Podelski, A.: Ultimate TreeAutomizer (CHC-COMP tool description). In: HCVS/PERR@ETAPS '19. pp. 42–47. EPTCS 296 (2019). https://doi.org/10.4204/EPTCS.296.7
10. Fedyukovich, G., Prabhu, S., Madhukar, K., Gupta, A.: Solving constrained Horn clauses using syntax and data. In: FMCAD '18. pp. 1–9 (2018). https://doi.org/10.23919/FMCAD.2018.8603011
11. 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
12. Frohn, F., Naaf, M., Brockschmidt, M., Giesl, J.: Inferring lower runtime bounds for integer programs. ACM Trans. Program. Lang. Syst. **42**(3), 13:1–13:50 (2020). https://doi.org/10.1145/3410331
13. Frohn, F., Giesl, J.: Proving non-termination by Acceleration Driven Clause Learning. In: CADE '23. LNCS 14132 (2023). https://doi.org/10.1007/978-3-031-38499-8_13
14. Frohn, F., Giesl, J.: ADCL: Acceleration Driven Clause Learning for constrained Horn clauses. In: SAS '23. LNCS (2023), to appear. Full version available at CoRR **abs/2303.01827**, https://doi.org/10.48550/arXiv.2303.01827
15. Hoder, K., Bjørner, N.S.: Generalized property directed reachability. In: SAT '12. pp. 157–171. LNCS 7317 (2012). https://doi.org/10.1007/978-3-642-31612-8_13
16. Hojjat, H., Iosif, R., Konečný, F., Kuncak, V., Rümmer, P.: Accelerating interpolants. In: ATVA '12. pp. 187–202. LNCS 7561 (2012). https://doi.org/10.1007/978-3-642-33386-6_16
17. Hojjat, H., Rümmer, P.: The Eldarica Horn solver. In: FMCAD '18. pp. 1–7 (2018). https://doi.org/10.23919/FMCAD.2018.8603013
18. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Formal Methods Syst. Des. **48**(3), 175–205 (2016). https://doi.org/10.1007/s10703-016-0249-4

19. Kroening, D., Lewis, M., Weissenbacher, G.: Under-approximating loops in C programs for fast counterexample detection. Formal Methods Syst. Des. **47**(1), 75–92 (2015). https://doi.org/10.1007/s10703-015-0228-1
20. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS '08. pp. 337–340. LNCS 4963 (2008). https://doi.org/10.1007/978-3-540-78800-3_24
21. 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
22. Zhu, H., Magill, S., Jagannathan, S.: A data-driven CHC solver. In: PLDI '18. pp. 707–721 (2018). https://doi.org/10.1145/3192366.3192416

A Missing Proofs

A.1 Proof of Thm. 14

Theorem 14 (Properties of $\mathsf{ABMC}_{\mathsf{block}}$). $\mathsf{ABMC}_{\mathsf{block}}$ *is*
Refutationally Complete: *If* $\mathcal{T}$ *is unsafe, then* $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *returns* unsafe.
Sound: *If* $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *returns* safe, *then* $\mathcal{T}$ *is safe.*
If $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *returns* unsafe, *then* $\mathcal{T}$ *is unsafe.*
Non-Terminating: *If* $\mathcal{T}$ *is safe, then* $\mathsf{ABMC}_{\mathsf{block}}(\mathcal{T})$ *may not terminate.*

Proof. Soundness for unsafety and non-termination are trivial. In the following, we prove refutational completeness. More precisely, we prove that when assuming a complete SMT solver, unsafety of $\mathcal{T}$ implies that Alg. 3 eventually returns unsafe, independently of the non-determinisms of Alg. 3 (i.e., independently of the models that are found by the SMT solver and the implementation of should_accel). Together with soundness for unsafety, this means that Alg. 3 returns unsafe if and only if $\mathcal{T}$ is unsafe, which in turn implies soundness for safety.

The proof proceeds as follows: We consider an arbitrary but fixed unsafe safety problem $(\psi_{\mathsf{init}}, \tau, \psi_{\mathsf{err}})$. Thus, there is a model σ' and a sequence $[\tau_i]_{i=0}^{c-1} \in \mathsf{sip}(\tau)^c$ of original transitions such that $\sigma' \models \mu_0(\psi_{\mathsf{init}})$, $\sigma'(\vec{x}^{(i)}) \to_{\tau_i} \sigma'(\vec{x}^{(i+1)})$ for all $0 \leq i < c$, and $\sigma' \models \mu_c(\psi_{\mathsf{err}})$. So in particular, σ' can be extended to a substitution σ such that

$$\sigma \models \mu_0(\psi_{\mathsf{init}}) \wedge \bigwedge_{i=0}^{c-1} \mu_i(\tau_i), \tag{1}$$

i.e., $[\tau_i]_{i=0}^{c-1}$ and σ witness that the state $\sigma(\vec{x}^{(c)})$ is reachable from the initial state $\sigma(\vec{x}^{(0)})$. We now have to prove that every run of Alg. 3 yields unsafe.

To this end, we define a class of infinite sequences $[\varphi_j]_{j\in\mathbb{N}}$ of formulas with a certain structure. This class is defined in such a way that every run of Alg. 3 corresponds to a prefix of such an infinite sequence in the sense that φ_j is the formula that is added to the SMT problem in Line 17 or 19 when $b = j$. Note that the converse is not true, i.e., there are sequences of formulas $[\varphi_j]_{j\in\mathbb{N}}$ with the desired structure that do not correspond to a run of Alg. 3. Next, we choose such a sequence $[\varphi_j]_{j\in\mathbb{N}}$ arbitrarily. Let

$$\xi_m := \mu_0(\psi_{\mathsf{init}}) \wedge \bigwedge_{j=0}^{m} \varphi_j \text{ for any } m \in \mathbb{N} \quad \text{and} \quad \xi_{-1} := \mu_0(\psi_{\mathsf{init}}).$$

Then our goal is to prove that there is a $k \in \mathbb{N}$ and a model θ of ξ_{k-1} such that $\theta(\vec{x}^{(0)}) = \sigma(\vec{x}^{(0)})$ and $\theta(\vec{x}^{(k)}) = \sigma(\vec{x}^{(c)})$. Note that if $[\varphi_j]_{j\in\mathbb{N}}$ corresponds to a run of Alg. 3, then ξ_{k-1} is the formula that is checked by Alg. 3 in Line 20 when $b = k-1$. Now assume that $\sigma(\vec{x}^{(c)})$ is an error state, i.e., $\sigma \models \mu_c(\psi_{\mathsf{err}})$. Then Alg. 3 checks the formula $\xi_{k-1} \wedge \mu_k(\psi_{\mathsf{err}})$ in Line 6 in the next iteration (when $b = k$). As $\theta \models \xi_{k-1}$, $\sigma \models \mu_c(\psi_{\mathsf{err}})$, and $\theta(\vec{x}^{(k)}) = \sigma(\vec{x}^{(c)})$, we get $\theta \models \xi_{k-1} \wedge \mu_k(\psi_{\mathsf{err}})$. Thus, Alg. 3 returns unsafe when $b = k$. As $[\varphi_j]_{j\in\mathbb{N}}$ was chosen arbitrarily, this implies refutational completeness.

ρ: an original or learned transition

$[\varphi_j]_{j\in\mathbb{N}}$: an infinite sequence of formulas such that every $\mathsf{ABMC}_{\mathsf{block}}$-run corresponds to a prefix of such a sequence, as explained above. By choosing this sequence arbitrarily, we cover all possible $\mathsf{ABMC}_{\mathsf{block}}$-runs.

χ_j: a formula such that $\mu_j(\chi_j)$ is equivalent to φ_j without blocking clauses

π_j: the transition that is learned by Alg. 3 when $b = j$, if any

$\mathsf{id}(\rho)$: We define $\mathsf{id}(\rho) = 0$ if $\rho \in \mathsf{sip}(\tau)$, i.e., if ρ is an original transition. Otherwise, $\mathsf{id}(\rho)$ is a unique identifier of ρ.

$\vec{\rho}^j$: a vector of transitions such that $\rightarrow_\rho \subseteq \rightarrow_\tau^+$ for each $\rho \in \vec{\rho}^j$. Intuitively, $\vec{\rho}^j$ corresponds to the cyclic suffix $\vec{\pi}^\circlearrowleft$ when $b = j$, if any. More precisely, then we have $\vec{\pi}^\circlearrowleft = [\rho_i^j \wedge \ell = \mathsf{id}(\rho_i^j)]_{i=0}^{|\vec{\rho}^j|-1}$.

β_j: the conjunction of all blocking clauses that are added by Alg. 3 when $b = j$

$\beta_{1,j}$: the blocking clause that is constructed in Line 15 when $b = j$, if any

$\beta_{2,j}$: the blocking clause that is constructed in Line 16 when $b = j$, if any

$[\tau_i]_{i=0}^{c-1}$: a sequence of original transitions

σ: a model of $\mu_0(\psi_{\mathsf{init}})$ such that $\sigma(\vec{x}^{(0)}) \rightarrow_{\tau_0} \sigma(\vec{x}^{(1)}) \rightarrow_{\tau_1} \ldots \rightarrow_{\tau_{c-1}} \sigma(\vec{x}^{(c)})$, i.e., σ witnesses reachability of $\sigma(\vec{x}^{(1)}), \ldots, \sigma(\vec{x}^{(c)})$ (from an initial state)

k: a number for which we will prove that Alg. 3 shows reachability of $\sigma(\vec{x}^{(c)})$ when $b = k - 1$, at the latest

c_j: for each $j \in \{0, \ldots, k\}$, c_j is a number such that Alg. 3 shows reachability of $\sigma(\vec{x}^{(c_j)})$ when $b = j - 1$, at the latest

ξ_m: as defined above – the formula that the $\mathsf{ABMC}_{\mathsf{block}}$-run which corresponds to $[\varphi_j]_{j=0}^{k-1}$ (if any) checks in Line 20 when $b = m$. So ξ_m may contain $\vec{x}^{(0)}, \ldots, \vec{x}^{(m+1)}$.

θ_j: for each $j \in \{0, \ldots, k\}$, θ_j is a partial model of ξ_{j-1}, i.e., $\theta_j(\xi_{j-1})$ is satisfiable. So the domain of θ_j may contain $\vec{x}^{(0)}, \ldots, \vec{x}^{(j)}$.

η_j: for each $j \in \{0, \ldots, k\}$, η_j instantiates the variables that occur in π_j, but not in $\vec{\rho}^j$, i.e., the additional variables introduced by acceleration

θ: an extension of θ_k such that $\theta \models \xi_{k-1}$

μ_j^{pre}: a substitution such that $\mu_j^{\mathsf{pre}}(\vec{x}') = \vec{x}'$ and $\mu_j^{\mathsf{pre}}(x) = x^{(j)}$ if $x \notin \vec{x}'$. In other words, μ_j^{pre} behaves like μ_j on all but the post-variables, where it is the identity.

Fig. 3: Notation used throughout the proof

In the sequel, we introduce various notations, which are summarized in Fig. 3 for easy reference. We now define our class of infinite sequences $[\varphi_j]_{j\in\mathbb{N}}$. We have

$$\varphi_j := \mu_j(\chi_j) \wedge \beta_j \qquad \text{and} \qquad \chi_j := (\tau \wedge \ell = 0) \vee \pi_j$$

where one of the following holds for each $j \in \mathbb{N}$:

$$\pi_j \equiv \bot \qquad \text{and} \qquad \beta_j \equiv \top$$

or

$$\pi_j \equiv \mathsf{accel}(\vec{\rho}^j) \wedge \ell = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)),$$

$$\beta_{1,j} \equiv \neg \left(\bigwedge_{i=0}^{|\vec{\rho}^j|-1} \mu_{j+i}(\rho_i^j \wedge \ell = \mathsf{id}(\rho_i^j)) \right),$$

$$\beta_{2,j} \equiv \ell^{(j)} \neq \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \vee \neg \left(\bigwedge_{i=0}^{|\vec{\rho}^j|-1} \mu_{j+i+1}(\rho_i^j \wedge \ell = \mathsf{id}(\rho_i^j)) \right), \qquad \text{and}$$

$\beta_j \equiv \beta_{1,j} \wedge \beta_{2,j}$

Here, the literals $\ell = \mathsf{id}(\rho_i^j)$ are contained in $\beta_{1,j}$ and $\beta_{2,j}$ as Alg. 3 conjoins $\ell = 0$ to τ and $\ell = \mathsf{id}$ to learned transitions. So we assume that ρ_i^j is an (original or learned) transition without the additional literal for ℓ, and we made those literals explicit in the definitions of $\beta_{1,j}$ and $\beta_{2,j}$ for clarity. Thus, φ_j is equivalent to the formula that is added to the SMT problem in Line 19 or 17.

To obtain k and θ, we now define sequences $[c_j]_{j=0}^k$ and $[\theta_j]_{j=0}^k$ inductively such that $\theta_j(\xi_{j-1})$ is satisfiable and $\theta_j(x^{(j)}) = \sigma(x^{(c_j)})$ for all $x \neq \ell$ such that $x^{(c_j)} \in \mathrm{dom}(\sigma)$. Here, two problems have to be solved: The first problem is that Alg. 3 may replace many evaluation steps by a single step using an accelerated transition. Therefore, the j^{th} step in the algorithm does not necessarily correspond to the j^{th} step in the evaluation with the original transitions $[\tau_i]_{i=0}^{c-1}$, but to the c_j^{th} step where $c_j \geq j$. The second problem is that the accelerated transitions may contain additional variables which have to be instantiated appropriately (to this end, we use substitutions η_j).

a. $c_0 := 0$ and $\theta_0 := [x^{(0)}/\sigma(x^{(0)}) \mid x \notin \vec{x}']$
b. if $c_j = c$, then $k := j$, i.e., then the definition of $[c_j]_{j=0}^k$ and $[\theta_j]_{j=0}^k$ is complete
c. otherwise:
  c.a. if $\pi_j \equiv \bot$, then:
    c.a.a. $c_{j+1} := c_j + 1$
    c.a.b. $\theta_{j+1}(\ell^{(j)}) := 0$
  c.b. if $\pi_j \equiv \mathsf{accel}(\vec{\rho}^j) \wedge \ell = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j))$ let f_j be the maximal natural number such that $\sigma(\vec{x}^{(c_j)}) \to_{\pi_j} \sigma(\vec{x}^{(c_j+f_j)})$, or 0 if $\sigma(\vec{x}^{(c_j)}) \not\to_{\pi_j} \sigma(\vec{x}^{(c_j+f_j)})$ for all $f_j > 0$
    c.b.a. if $f_j = 0$, then:
      c.b.a.a. $c_{j+1} := c_j + 1$
      c.b.a.b. $\theta_{j+1}(\ell^{(j)}) := 0$
    c.b.b. if $f_j > 0$, then:
      c.b.b.a. $c_{j+1} := c_j + f_j$
      c.b.b.b. $\theta_{j+1}(x^{(j)}) := \eta_j(x^{(c_j)})$ for all $x^{(c_j)} \in \mathcal{V}(\mu_{c_j}(\pi_j)) \setminus \mathrm{dom}(\sigma)$, where η_j is a substitution with $\eta_j \circ \sigma \circ (\mu_{c_j}^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(c_j+f_j)}]) \models \pi_j$ (which exists due to $\sigma(\vec{x}^{(c_j)}) \to_{\pi_j} \sigma(\vec{x}^{(c_j+f_j)})$)
  c.c. $\theta_{j+1}(x^{(m)}) := \theta_j(x^{(m)})$ for all $x^{(m)} \in \mathrm{dom}(\theta_j)$
  c.d. $\theta_{j+1}(x^{(j+1)}) := \sigma(x^{(c_{j+1})})$ for all $x^{(c_{j+1})} \in \mathrm{dom}(\sigma)$

Note that the steps c.b.b.b. and c.c. do not contradict each other. This is obvious, except for the case that $m = j$ in Step c.c. Then $x^{(j)} \in \mathrm{dom}(\theta_j)$. Thus, $[x^{(j)}/\theta_j(x^{(j)})]$ has been added to θ_j in Step c.d. (in all other steps where entries are added to θ_j, the indices of the variable(s) and the substitution differ). Thus, we have $x^{(c_j)} \in \mathrm{dom}(\sigma)$, and Step c.b.b.b. explicitly excludes elements of $\mathrm{dom}(\sigma)$. Moreover, by construction, we have $c_m < c_{m+1}$, and thus the sequences are well defined.

Let $\theta(x) := \theta_k(x)$ if $x \in \mathrm{dom}(\theta_k)$, and for all $x \notin \mathrm{dom}(\theta_k)$, let $\theta(x)$ be a natural number which is larger than all constants that occur in ξ_{k-1}. To see why it is necessary to instantiate variables that are not contained in $\mathrm{dom}(\theta_k)$,

recall that ξ_{k-1} contains blocking clauses, and these blocking clauses may refer to "future steps", i.e., to steps that are not yet part of the SMT encoding. For example, it may happen that ξ_{k-1} contains a blocking clause that refers to $x^{(m)}$ with $m > k$. Similarly, ξ_{k-1} may contain a blocking clause referring to $\ell^{(k)}$, which is also not contained in $\mathrm{dom}(\theta_k)$. The reason is that $\ell^{(j)}$'s value is the identifier of the transition that is used for the step from $\sigma(\vec{x}^{(c_j)})$ to $\sigma(\vec{x}^{(c_{j+1})})$, but $\sigma(\vec{x}^{(c_k)}) = \sigma(\vec{x}^{(c)})$ is the last state of the run $\sigma(\vec{x}^{(0)}) \to_\tau \ldots \to_\tau \sigma(\vec{x}^{(c)})$ that we are considering.

We prove $\theta \models \xi_{k-1}$. To this end, we prove the following three statements individually:

$$\theta \models \mu_0(\psi_{\mathsf{init}}) \tag{2}$$

$$\theta \models \bigwedge_{j=0}^{k-1} \mu_j(\chi_j) \tag{3}$$

$$\theta \models \bigwedge_{j=0}^{k-1} \beta_j \tag{4}$$

Then $\theta \models \xi_{k-1}$ follows due to the definition of ξ_{k-1}.

For (2), we have

$$\begin{aligned}
& \sigma \models \mu_0(\psi_{\mathsf{init}}) && (1)\\
\curvearrowright\ & \theta_0 \models \mu_0(\psi_{\mathsf{init}}) && (\text{a.})\\
\curvearrowright\ & \theta_k \models \mu_0(\psi_{\mathsf{init}}) && (\text{c.c.})\\
\curvearrowright\ & \theta \models \mu_0(\psi_{\mathsf{init}}) && (\text{def. of } \theta)
\end{aligned}$$

For (3), let $j \in \{0, \ldots, k-1\}$ be arbitrary but fixed. We prove $\theta \models \mu_j(\chi_j)$. First consider the case $\pi_j \equiv \bot$, see (c.a.). Then:

$$\begin{aligned}
& \sigma \models \mu_{c_j}(\tau_{c_j}) && (((1), \text{as } j < k \text{ implies } c_j < c))\\
\curvearrowright\ & \sigma \models \mu_{c_j}(\tau) && (\tau_{c_j} \in \mathsf{sip}(\tau))\\
\curvearrowright\ & \sigma \circ \mu_{c_j} \models \tau\\
\curvearrowright\ & \sigma \circ (\mu_{c_j}^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(c_j+1)}]) \models \tau && (\text{def. of } \mu_{c_j} \text{ and } \mu_{c_j}^{\mathsf{pre}})\\
\curvearrowright\ & (\sigma \circ \mu_{c_j}^{\mathsf{pre}}) \uplus [\vec{x}'/\sigma(\vec{x}^{(c_j+1)})] \models \tau\\
\curvearrowright\ & (\sigma \circ \mu_{c_j}^{\mathsf{pre}}) \uplus [\vec{x}'/\sigma(\vec{x}^{(c_{j+1})})] \models \tau && (\text{c.a.a.})\\
\curvearrowright\ & (\theta_j \circ \mu_j^{\mathsf{pre}}) \uplus [\vec{x}'/\theta_{j+1}(\vec{x}^{(j+1)})] \models \tau && (\text{c.d.})\\
\curvearrowright\ & (\theta_{j+1} \circ \mu_j^{\mathsf{pre}}) \uplus [\vec{x}'/\theta_{j+1}(\vec{x}^{(j+1)})] \models \tau && (\text{c.c.})\\
\curvearrowright\ & \theta_{j+1} \circ (\mu_j^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(j+1)}]) \models \tau\\
\curvearrowright\ & \theta_{j+1} \circ \mu_j \models \tau && (\text{def. of } \mu_j \text{ and } \mu_j^{\mathsf{pre}})\\
\curvearrowright\ & \theta_{j+1} \models \mu_j(\tau)\\
\curvearrowright\ & \theta_{j+1} \circ [\ell^{(j)}/0] \models \mu_j(\chi_j) && (\tau \wedge \ell = 0 \models \chi_j)
\end{aligned}$$

$$\curvearrowright \quad \theta_{j+1} \models \mu_j(\chi_j) \qquad \text{(c.a.b.)}$$
$$\curvearrowright \quad \theta_k \models \mu_j(\chi_j) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta \models \mu_j(\chi_j) \qquad \text{(def. of } \theta\text{)}$$

The case (c.b.a.) is analogous. Now consider the case that $\pi_j \not\equiv \bot$ and $f_j > 0$, see (c.b.b.). Then:

$$\eta_j \circ \sigma \circ (\mu_{c_j}^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(c_j+f_j)}]) \models \pi_j \qquad \text{(c.b.b.b.)}$$
$$\curvearrowright \quad \eta_j \circ \sigma \circ (\mu_{c_j}^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(c_{j+1})}]) \models \pi_j \qquad \text{(c.b.b.a.)}$$
$$\curvearrowright \quad (\eta_j \circ \sigma \circ \mu_{c_j}^{\mathsf{pre}}) \uplus [\vec{x}'/\sigma(\vec{x}^{(c_{j+1})})] \models \pi_j \qquad (\mathrm{dom}(\eta_j) \cap \vec{x}^{(c_{j+1})} = \varnothing)$$
$$\curvearrowright \quad ((\eta_j \circ \mu_{c_j}^{\mathsf{pre}}) \circ (\sigma \circ \mu_{c_j}^{\mathsf{pre}})) \uplus [\vec{x}'/\sigma(\vec{x}^{(c_{j+1})})] \models \pi_j \qquad (\sigma \text{ is a ground substitution})$$
$$\curvearrowright \quad ((\eta_j \circ \mu_{c_j}^{\mathsf{pre}}) \circ (\theta_j \circ \mu_j^{\mathsf{pre}})) \uplus [\vec{x}'/\theta_{j+1}(\vec{x}^{(j+1)})] \models \pi_j \qquad \text{(c.d.)}$$
$$\curvearrowright \quad ((\eta_j \circ \mu_{c_j}^{\mathsf{pre}}) \circ (\theta_{j+1} \circ \mu_j^{\mathsf{pre}})) \uplus [\vec{x}'/\theta_{j+1}(\vec{x}^{(j+1)})] \models \pi_j \qquad \text{(c.c.)}$$
$$\curvearrowright \quad ((\theta_{j+1} \circ \mu_j^{\mathsf{pre}}) \circ (\theta_{j+1} \circ \mu_j^{\mathsf{pre}})) \uplus [\vec{x}'/\theta_{j+1}(\vec{x}^{(j+1)})] \models \pi_j \qquad \text{(c.b.b.b.)}$$
$$\curvearrowright \quad (\theta_{j+1} \circ \mu_j^{\mathsf{pre}}) \uplus [\vec{x}'/\theta_{j+1}(\vec{x}^{(j+1)})] \models \pi_j \qquad (\theta_{j+1} \text{ is a ground substitution})$$
$$\curvearrowright \quad \theta_{j+1} \circ (\mu_j^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(j+1)}]) \models \pi_j$$
$$\curvearrowright \quad \theta_{j+1} \circ \mu_j \models \pi_j \quad \text{(def. of } \mu_j^{\mathsf{pre}} \text{ and } \mu_j\text{)}$$
$$\curvearrowright \quad \theta_{j+1} \models \mu_j(\pi_j)$$
$$\curvearrowright \quad \theta_{j+1} \models \mu_j(\chi_j) \qquad (\pi_j \models \chi_j)$$
$$\curvearrowright \quad \theta_k \models \mu_j(\chi_j) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta \models \mu_j(\chi_j) \qquad \text{(def. of } \theta\text{)}$$

This finishes the proof of (3).

For (4), let $j \in \{0, \ldots, k-1\}$ be arbitrary but fixed. We prove $\theta \models \beta_j$. If $\beta_j \equiv \top$, the claim is trivial, so assume $\beta_j \not\equiv \top$, i.e., $\beta_j \equiv \beta_{1,j} \wedge \beta_{2,j}$. We prove $\theta \models \beta_{1,j}$ and $\theta \models \beta_{2,j}$ individually.

To prove $\theta \models \beta_{1,j}$, we have to show

$$\theta \models \neg \left(\bigwedge_{i=0}^{|\vec{\rho}^j|-1} \mu_{j+i}(\rho_i^j \wedge \ell = \mathsf{id}(\rho_i^j)) \right) \tag{5}$$

by definition of $\beta_{1,j}$. First consider the case $j + |\vec{\rho}^j| > k$. Then:

$$\ell^{(j+|\vec{\rho}^j|-1)} \notin \mathrm{dom}(\theta_k)$$
$$\curvearrowright \quad \theta \models \ell^{(j+|\vec{\rho}^j|-1)} \neq \mathsf{id}(\rho^j_{|\vec{\rho}^j|-1}) \quad \text{(def. of } \theta\text{, as } \mathsf{id}(\rho^j_{|\vec{\rho}^j|-1}) \text{ occurs in } \xi_{k-1}\text{)}$$
$$\curvearrowright \quad \theta \models \mu_{j+|\vec{\rho}^j|-1}(\ell \neq \mathsf{id}(\rho^j_{|\vec{\rho}^j|-1})) \qquad \text{(def. of } \mu_{j+|\vec{\rho}^j|-1}\text{)}$$

$\curvearrowright$ (5)

Now consider the case $j + |\vec{\rho}^j| \leq k$. First assume $f_j > 0$. Then:

$$\eta_j \circ \sigma \circ (\mu_{c_j}^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(c_j+f_j)}]) \models \pi_j \qquad \text{(c.b.b.b.)}$$
$$\curvearrowright \quad \eta_j \circ \sigma \circ (\mu_{c_j}^{\mathsf{pre}} \uplus [\vec{x}'/\vec{x}^{(c_j+f_j)}]) \models \ell = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \quad (\pi_j \models \ell = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)))$$
$$\curvearrowright \quad \eta_j \circ \sigma \models \ell^{(c_j)} = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(def. of } \mu_{c_j}^{\mathsf{pre}})$$
$$\curvearrowright \quad \eta_j \models \ell^{(c_j)} = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad (\ell^{(c_j)} \notin \mathrm{dom}(\sigma))$$
$$\curvearrowright \quad \theta_{j+1} \models \ell^{(j)} = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(c.b.b.b.)}$$
$$\curvearrowright \quad \theta_k \models \ell^{(j)} = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta \models \ell^{(j)} = \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(def. of } \theta)$$
$$\curvearrowright \quad \theta \models \ell^{(j)} \neq \mathsf{id}(\rho_0^j) \qquad (\mathsf{id}(\rho_0^j) \neq \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)))$$
$$\curvearrowright \quad \theta \models \neg\mu_j(\ell = \mathsf{id}(\rho_0^j)) \qquad \text{(def. of } \mu_j)$$
$$\curvearrowright \quad (5)$$

Now assume $f_j = 0$. Then:

$$\sigma(\vec{x}^{(c_j)}) \not\to_{\pi_j} \sigma(\vec{x}^{(c_j+g)}) \qquad \text{(for all } g > 0, \text{ as } f_j = 0)$$
$$\curvearrowright \quad \sigma(\vec{x}^{(c_j)}) \not\to_{\pi_j} \sigma(\vec{x}^{(c_{j+|\vec{\rho}^j|})}) \qquad (j + |\vec{\rho}^j| \leq k \text{ and } c_{j+|\vec{\rho}^j|} > c_j)$$
$$\curvearrowright \quad \sigma(\vec{x}^{(c_j)}) \not\to_{\vec{\rho}^j} \sigma(\vec{x}^{(c_{j+|\vec{\rho}^j|})}) \qquad (\to_{\pi_j} = \to_{\vec{\rho}^j}^+)$$
$$\curvearrowright \quad \theta_j(\vec{x}^{(j)}) \not\to_{\vec{\rho}^j} \theta_{j+|\vec{\rho}^j|}(\vec{x}^{(j+|\vec{\rho}^j|)}) \qquad \text{(c.d.)}$$
$$\curvearrowright \quad \theta_{j+|\vec{\rho}^j|}(\vec{x}^{(j)}) \not\to_{\vec{\rho}^j} \theta_{j+|\vec{\rho}^j|}(\vec{x}^{(j+|\vec{\rho}^j|)}) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta_{j+|\vec{\rho}^j|} \models \neg \left(\bigwedge_{i=0}^{|\vec{\rho}^j|-1} \mu_{j+i}(\rho_i^j) \right) \qquad \text{(def. of } \to_{\vec{\rho}^j})$$
$$\curvearrowright \quad \theta_{j+|\vec{\rho}^j|} \models \neg\mu_{j+i}(\rho_i^j) \qquad \text{(for some } 0 \leq i < |\vec{\rho}^j|)$$
$$\curvearrowright \quad \theta_k \models \neg\mu_{j+i}(\rho_i^j) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta \models \neg\mu_{j+i}(\rho_i^j) \qquad (\text{def. of } \theta, \text{ as } j + i < j + |\vec{\rho}^j| \leq k)$$
$$\curvearrowright \quad (5)$$

This finishes the proof of (5).

To prove $\theta \models \beta_{2,j}$, we have to show

$$\theta \models \ell^{(j)} \neq \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \vee \neg \left(\bigwedge_{i=0}^{|\vec{\rho}^j|-1} \mu_{j+i+1}(\rho_i^j \wedge \ell = \mathsf{id}(\rho_i^j)) \right) \tag{6}$$

by definition of $\beta_{2,j}$. First consider the case $j + |\vec{\rho}^j| \geq k$. Then:

$$\ell^{(j+|\vec{\rho}^j|)} \notin \mathrm{dom}(\theta_k)$$
$$\curvearrowright \quad \theta \models \ell^{(j+|\vec{\rho}^j|)} \neq \mathsf{id}(\rho^j_{|\vec{\rho}^j|-1}) \qquad (\text{def. of } \theta, \text{ as } \mathsf{id}(\rho^j_{|\vec{\rho}^j|-1}) \text{ occurs in } \xi_{k-1})$$

$$\curvearrowright \quad \theta \models \neg\mu_{j+|\vec{\rho}^j|}(\ell = \mathsf{id}(\rho^j_{|\vec{\rho}^j|-1})) \qquad \text{(def. of } \mu_{j+|\vec{\rho}^j|}\text{)}$$
$$\curvearrowright \quad (6)$$

Now consider the case $j + |\vec{\rho}^j| < k$. First assume $f_j = 0$. Then:

$$\theta_{j+1} \models \ell^{(j)} \neq \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(c.b.a.b.)}$$
$$\curvearrowright \quad \theta_k \models \ell^{(j)} \neq \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta \models \ell^{(j)} \neq \mathsf{id}(\mathsf{accel}(\vec{\rho}^j)) \qquad \text{(def. of } \theta\text{)}$$
$$\curvearrowright \quad (6)$$

Now assume $f_j > 0$. Then:

$$\sigma(\vec{x}^{(c_j+f_j)}) \not\to_{\pi_j} \sigma(\vec{x}^{(c_j+f_j+g)}) \qquad \text{(for all } g > 0\text{, as } f_j \text{ is maximal)}$$
$$\curvearrowright \quad \sigma(\vec{x}^{(c_{j+1})}) \not\to_{\pi_j} \sigma(\vec{x}^{(c_{j+1}+g)}) \qquad \text{(c.b.b.a.)}$$
$$\curvearrowright \quad \sigma(\vec{x}^{(c_{j+1})}) \not\to_{\pi_j} \sigma(\vec{x}^{(c_{j+1+|\vec{\rho}^j|})}) \qquad (j + |\vec{\rho}^j| < k \text{ and } c_{j+1} < c_{j+1+|\vec{\rho}^j|})$$
$$\curvearrowright \quad \sigma(\vec{x}^{(c_{j+1})}) \not\to_{\vec{\rho}^j} \sigma(\vec{x}^{(c_{j+1+|\vec{\rho}^j|})}) \qquad (\to_{\pi_j} = \to^+_{\vec{\rho}^j})$$
$$\curvearrowright \quad \theta_{j+1}(\vec{x}^{(j+1)}) \not\to_{\vec{\rho}^j} \theta_{j+1+|\vec{\rho}^j|}(\vec{x}^{(j+1+|\vec{\rho}^j|)}) \qquad \text{(c.d.)}$$
$$\curvearrowright \quad \theta_{j+1+|\vec{\rho}^j|}(\vec{x}^{(j+1)}) \not\to_{\vec{\rho}^j} \theta_{j+1+|\vec{\rho}^j|}(\vec{x}^{(j+1+|\vec{\rho}^j|)}) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta_{j+1+|\vec{\rho}^j|} \models \neg \left(\bigwedge_{i=0}^{|\vec{\rho}^j|-1} \mu_{j+1+i}(\rho^j_i) \right) \qquad \text{(def. of } \to_{\vec{\rho}^j}\text{)}$$
$$\curvearrowright \quad \theta_{j+1+|\vec{\rho}^j|} \models \neg\mu_{j+1+i}(\rho^j_i) \qquad \text{(for some } i \in \{0, \ldots, |\vec{\rho}^j| - 1\}\text{)}$$
$$\curvearrowright \quad \theta_k \models \neg\mu_{j+1+i}(\rho^j_i) \qquad \text{(c.c.)}$$
$$\curvearrowright \quad \theta \models \neg\mu_{j+1+i}(\rho^j_i) \qquad \text{(def. of } \theta\text{, as } j + 1 + i < j + 1 + |\vec{\rho}^j| \leq k\text{)}$$
$$\curvearrowright \quad (6)$$

This finishes the proof of (6), and hence also the proof of (4). Thus, we have proven $\theta \models \xi_{k-1}$.

Now assume that $\sigma(\vec{x}^{(c)})$ is an error state, i.e., $\sigma \models \mu_c(\psi_{\mathsf{err}})$. For all $x \neq \ell$ such that $x^{(c_k)} \in \mathrm{dom}(\sigma)$, we have

$$\begin{aligned} \theta(x^{(k)}) &= \theta_k(x^{(k)}) && \text{(by def. of } \theta\text{)} \\ &= \sigma(x^{(c_k)}) && \text{(by c.d.)} \\ &= \sigma(x^{(c)}). && \text{(by b.)} \end{aligned}$$

Thus, we get $\theta \models \xi_{k-1} \wedge \mu_k(\psi_{\mathsf{err}})$. Hence, the run of Alg. 3 that checks the formula ξ_{k-1} when $b = k - 1$ returns unsafe when $b = k$. Note that satisfiability of ξ_{k-1} implies satisfiability of all formulas that are checked by Alg. 3 in Line 20 when $b < k - 1$. Thus, Alg. 3 cannot return safe when $b < k$. As $[\varphi_j]_{j\in\mathbb{N}}$ was chosen arbitrarily, this implies refutational completeness. □