Int J Softw Tools Technol Transfer
DOI 10.1007/s10009-014-0346-x

REGULAR PAPER

Downward pattern refinement for timed automata

Martin Wehrle · Sebastian Kupferschmid

Abstract Directed model checking is a well-established approach for detecting error states in concurrent systems. A popular variant to find *shortest* error traces is to apply the A* search algorithm with distance heuristics that never overestimate the real error distance. An important class of such distance heuristics is the class of *pattern database* heuristics. Pattern database heuristics are built on abstractions of the system under consideration. In this paper, we propose *downward pattern refinement*, a systematic approach for the construction of pattern database heuristics for concurrent systems of timed automata. First, we propose a general framework for pattern databases in the context of timed automata and show that desirable theoretical properties hold for the resulting pattern database. Afterward, we formally define a concept to measure the accuracy of abstractions. Based on this concept, we propose an algorithm for computing succinct abstractions that are still accurate to produce informed pattern databases. We evaluate our approach on large and complex industrial problems. The experiments show the practical potential of the resulting pattern database heuristic.

Keywords Directed model checking · Heuristic search · Pattern databases · Bug finding · Timed Automata

1 Introduction

Model checking [4] is an automated approach for the verification of concurrent systems. For a given mathematical model $\mathcal{M}$ of a system and a given property φ, the objective of model checking is to prove that $\mathcal{M}$ satisfies φ, i. e., $\mathcal{M} \models \varphi$. To prove this, common model checking algorithms perform a search in the state space that is induced by $\mathcal{M}$, and check whether the induced state space satisfies φ.

Complementary to proving that a mathematical model satisfies a property, an important practical aspect of model checking is *bug finding*, i. e., to detect reachable *error states* in faulty systems where $\mathcal{M} \not\models \varphi$. This is important because during the development of systems, bugs do often occur, and therefore, effective and efficient approaches for bug finding are required. In addition, to be able to debug a system effectively, it is important to have *short* or preferably *shortest possible* error traces because short error traces are easier to understand than longer ones.

Directed model checking is a variant of model checking that is specifically optimized to find short error traces in faulty systems. Directed model checking has recently found much attention in different variants and contexts [9,12,14,19,22,23,30,32,34–37]. For a given model $\mathcal{M}$ of a system, the main idea is to focus the search on those parts of the state space of $\mathcal{M}$ that appear to be promising to contain a reachable error state. The required information to guide the search accordingly is obtained from a *distance heuristic*. For each encountered state, distance heuristics estimate the distance to a nearest error state, where states with lower estimated error distance are preferably explored. Distance heuristics are usually based on abstractions of the original system and computed fully automatically. As shorter error traces are easier to understand than longer ones, ultimately shortest possible error traces are desired to effectively debug the system. Shortest possible error traces can be found with *admissible* distance heuristics, i. e., heuristics that never overestimate the real error distance, together with the A* search algorithm [15,16].

M. Wehrle (✉)
University of Basel, Basel, Switzerland
e-mail: martin.wehrle@unibas.ch

S. Kupferschmid
ATRiCS Advanced Traffic Solutions GmbH, Freiburg, Germany
e-mail: sebastian.kupferschmid@atrics.com

Published online: 04 September 2014

An important class of admissible distance heuristics is the class of *pattern database heuristics*. Pattern databases (PDBs) have originally been introduced in the area of Artificial Intelligence [5,10]. More recently, PDBs have also been applied for directed model checking [23,30]. To compute a PDB for a given system, a subset of the automata and variables of the original system (the so-called *pattern*) is selected, which determines a corresponding abstraction of the original system. The resulting PDB is defined as a data structure that contains the abstract state space of this abstraction together with the abstract error distances. Pattern databases can be used as a distance heuristic by estimating the error distance for a concrete state with the abstract error distance of the corresponding abstract state in the PDB. Obviously, the most crucial part in the design of a pattern database heuristic is the choice of the pattern, which determines the heuristic's behavior and therefore the overall quality of the resulting heuristic. Ultimately, one seeks for patterns that are as small as possible (to be able to handle large systems) and that yield abstractions that are as "similar" to the original system as possible to appropriately reflect the original system behavior.

In recent years, directed model checking has particularly found increasing attention for finding bugs in concurrent systems of timed automata. The theory of timed automata [1] provides a formalism to model timed systems, which frequently occur in practice as embedded systems. In particular, timed systems often represent safety critical applications such as airbags or traffic light control systems. Obviously, especially for such safety critical systems, automated techniques are desired that help the engineer to systematically debug the system. However, model checking and specifically directed model checking becomes more difficult for such systems because of the clock variables that are real-valued and obey a special semantics. Although it is well-known that reachability is decidable for a certain class of timed automata (e.g., [3]), clocks generally cause an additional exponential blow-up of the state space that needs to be effectively addressed in order to make model checking approaches scalable for practically relevant timed automata systems.

In this paper, we present *downward pattern refinement*, a systematic approach to the pattern selection problem for concurrent systems of timed automata. We first present a general framework for pattern database heuristics in the context of timed automata, and particularly show that the resulting pattern database heuristic is *consistent*, which is a stronger property than admissibility and offers desirable properties that simplify the search algorithm [28]. Afterward, we present our downward pattern refinement approach as a mechanism for the pattern selection problem. The presentation of downward pattern refinement consists of two parts. In the first part, we present the underlying theory by identifying suitable criteria to estimate the quality of abstractions. For this purpose, we develop suitable criteria to estimate the similarity of systems. In the second part, based on these criteria, we present a pattern selection algorithm based on successively abstracting the original system as long as the resulting system remains similar according to these criteria (i.e., as long as only little spurious behavior is introduced). We have implemented downward pattern refinement and the corresponding architecture for pattern databases into our model checking tool MCTA [25,36]. We demonstrate that downward pattern refinement can result in small patterns that still lead to very informed pattern database heuristics. This yields a powerful approach that is able to handle large problems that could not be solved optimally before. In particular, we show that the resulting pattern database heuristic recognizes many dead end states. Correctly identifying dead end states is useful to reduce the search effort significantly, since such states can be excluded from the search process without losing completeness. This even allows us to efficiently verify *correct* systems with abstraction-based directed model checking techniques.

Let us have a look at the relationship of directed model checking with abstraction-based distance heuristics to counterexample guided abstraction refinement (CEGAR). In contrast to CEGAR (which aims at computing accurate abstractions for proving or refuting a property), distance heuristics like PDBs can be computed based on abstractions that are not (yet) fine enough to entirely prove or refute the property—this is because for directed model checking, abstractions are used as the basis for a distance heuristic to guide the search. Accurate abstractions are expected to yield accurate search guidance, whereas coarse abstractions will possibly lead to rather uninformed search behavior. Overall, there is a relationship between the accuracy of the abstraction and the search behavior with the resulting distance heuristic. We will come back to this point later.

The remainder of the paper is organized as follows. In Sect. 2, we introduce the necessary background that is needed for this work, including the concepts of timed automata, directed model checking and pattern databases. Section 3 provides a generic framework for pattern databases when applied to timed automata. Based on these results, we present the theoretical basis for downward pattern refinement in Sect. 4. Based on this basis, we propose a corresponding pattern selection algorithm in Sect. 5. Our approach is evaluated within our MCTA model checker on large real-time benchmarks that stem from an industrial case study. The results show that our approach is able to outperform related approaches, including the state-of-the-art tool UPPAAL [2,26]. Finally, we conclude the paper and give a short outlook on future work.

2 Preliminaries

We present the preliminaries that are needed for this work. These include the computational model in Sect. 2.1 and a detailed introduction to directed model checking in Sect. 2.2. Furthermore, we describe the general approach for pattern database heuristics in Sect. 2.3.

2.1 Notation

Our computational model is based on the model of timed automata as proposed by Behrmann et al. [2]. We focus on a subclass thereof that is sufficiently rich to capture the central ideas of this paper. In order to define our model, we first need some more terminology. Let X be a finite set of *clocks*, V be a finite set of *bounded integer variables*, and Σ be a finite set of *synchronization labels* that particularly contains a special internal void label τ. For $n \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$, let *IntGrd* be the set of conjunctions of integer constraints over V of the form $v \bowtie n$ with $v \in V$, and let *ClockCstr* be the set of clock constraints of the form $x \bowtie n$ with $x \in X$. Furthermore, let *IntEff* be the set of integer effects over V, where an integer effect is defined as a set of integer assignments of the form $v := m$ with $v \in V$ and $m \in \mathbb{N}$. Finally, *Reset* $\subseteq X$ is defined as a subset of the clocks. Based on these notations, we define a network $\mathcal{M}$ of timed automata as a set of timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, where for each $i \in \{1, \ldots, n\}$, the timed automaton $\mathcal{A}_i = \langle L_i, I_i, E_i \rangle$ consists of a finite set of *locations* L_i, an *invariant* $I_i : L_i \rightarrow 2^{ClockCstr}$, and a set of *edges* E_i. The invariant represents a (possibly empty) set of clock constraints of the form $x \leq n$ for clocks $x \in X$ and integer values $n \in \mathbb{N}$. Furthermore, the set of edges is defined as $E_i \subseteq L_i \times \mathit{IntGrd} \times \mathit{ClockCstr} \times \Sigma \times \mathit{IntEff} \times \mathit{Reset} \times L_i$.

The semantics of $\mathcal{M}$ is defined as follows. As the domain of clocks is the set of the nonnegative real numbers, the state space of networks of timed automata is infinite. However, reachability can be decided because there are finite partitionings of the infinite state space that are sound and complete [1]. An efficient partitioning in practice is based on *zones*. A zone is a symbolic representation of clock values based on clock constraints. The corresponding state space is also called the *zone graph* of the system. In the following, we always refer to this symbolic setting. We define a *global symbolic system state* s (or *state* for short) of $\mathcal{M}$ as a valuation that maps each automaton $\mathcal{A}_i$ to a location $l \in L_i$, together with a valuation that maps each integer variable to an integer value of its domain, together with a zone, i. e., a conjunction of clock constraints that express the possible values of the clocks in s. A *global system transition* t (or *transition* for short) consists of an *action* transition t_a and a *delay* transition t_d. The action transition t_a is either asynchronous and consists of one edge that is annotated with the synchronization label τ, or it is synchronous and consists of two edges (from different automata) with a common synchronization label that is different to τ. If more than two edges in different automata share the same synchronization label $\neq \tau$, there are several corresponding action transitions, i. e., one action transition for every pair of edges with this label. The delay transition t_d lets time pass by simultaneously increasing the values of the clocks. A transition t is *applicable* in a state s if and only if s satisfies the location and the integer guards of t_a, and the zone of s satisfies the clock guards of t_a. The application of a transition t in a state s results in a state that is obtained by first applying t_a in s, which updates the locations and integer values. From this intermediate state, the zone of s is first updated according to the clock guard and the clock resets of t_a. Finally, t_d maximizes the resulting zone while preserving consistency with the invariants of the destination locations of t_a. We denote the resulting state with $s' = t[s]$.

We define a directed model checking (DMC) problem $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ as a tuple that consists of a network of timed automata $\mathcal{M}$, a set of integer variables V, a set of clock variables X, the initial state s_0 of Θ, and a property φ. A *trace* $\pi = t_1, \ldots, t_n$ in Θ is defined as a sequence of transitions that are sequentially applicable in a state reachable from the initial state s_0. The *length* $\|\pi\|$ of a trace π is defined as the number of transitions in π, i. e., $\|\pi\| = n$ for the example trace. An *error trace* is defined as a trace that ends with a state that violates the property φ. The *error distance* $d(s)$ of a state s is defined as the length of a shortest error trace that starts in s. When we want to stress that d is a function also of the system $\mathcal{M}$, we write $d(s, \mathcal{M})$. In this paper, we address the task to find a shortest possible error trace from s_0, i. e., a trace that starts in s_0 and ends in a state s_e with $s_e \not\models \varphi$.

2.2 Directed model checking

Directed model checking is a variant of explicit state model checking that directs the search toward error states with *distance heuristics*. In the context of timed automata, the search is performed on the zone graph of the system, where states are defined as described above. Directed model checking influences the order in which the states are explored until an error state is found, where error states are defined as states that violate a given property φ. We consider the situation where $\varphi = \bigwedge_{i=1}^{k} \varphi_i$ is a conjunction. Without loss of generality, we assume that each φ_i is a location constraint $\mathcal{A} = l_e$, which states that $\mathcal{A}$ is in location l_e (integer and clock constraints c can be modeled by introducing edges to a new location l_e with c as guard).

Figure 1 shows a basic DMC algorithm. The input of the algorithm is a network of timed automata $\mathcal{M}$, the initial state s_0 of $\mathcal{M}$, a property φ, and a distance heuristic h. In the following, a state is called *explored* if all its successor states have already been computed. For the search, we maintain a priority queue *open*, which maintains states that have been created but not yet explored. Furthermore, a *closed* list maintains the explored states in order to avoid exploring cycles in the state space. In the main loop of the algorithm, the

```
function dmc($\mathcal{M}$, $s_0$, $\varphi$, $h$):
    open = empty priority queue
    closed = $\emptyset$
    open.insert($s_0$, priority($s_0$))
    while open $\neq \emptyset$ do:
        $s$ = open.getMinimum()
        if $s \not\models \varphi$ then:
            generateErrorTrace($s$)
        closed = closed $\cup \{s\}$
        for each transition $t$ that is applicable in $s$ do:
            compute $s' = t[s]$
            if $s' \notin$ closed then:
                open.insert($s'$, priority($s'$))
    return True
```

Fig. 1 A basic directed model checking algorithm

method open.get-Minimum() takes a state s with lowest priority value from *open* (see below how the priority values are computed). Afterward, s is explored by first checking if it is an error state, i. e., if $s \not\models \varphi$. If this is the case, an error trace is generated by back-tracing from s (for this, in a practical implementation, we additionally store for each state s how s has been reached). If not, i. e., if $s \models \varphi$, the successor state $s' = t[s]$ is computed for each transition t that is applicable in s. If s' has not already been encountered before, then the priority value is computed and s' is inserted into *open*.

The priority of a state s is defined as $h(s) + c(s)$, where $h(s) \in \mathbb{N} \cup \{\infty\}$ reflects the estimation of the error distance of s, and $c(s)$ is defined as the length of a shortest trace from s_0 to s. A distance heuristic h is *admissible* if $h(s) \leq d(s)$ for all states s, i. e., if h never overestimates the real error distance d. Under the assumption that the distance heuristic h is *consistent*, the resulting algorithm is equal to the A* search algorithm. A distance heuristic h is defined as consistent if $h(s_e) = 0$ for all error states s_e, and $h(s) \leq h(s') + 1$ for all states s and successor states s'. Consistency implies admissibility, and therefore, we get shortest possible error traces with consistent distance heuristics and A* [28]. Furthermore, consistent distance heuristics have the property that once a state has been explored, it does not have to be reopened in any case, i. e., A* directly finds shortest possible traces to all explored states [28]. Overall, consistency is a desirable property for distance heuristics; we will show that our pattern database heuristic is consistent in Sect. 3.

2.3 Pattern database heuristics

Let $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem for $\mathcal{M} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, integer variables V and clock variables X. A pattern database heuristic for Θ is computed prior to directed model checking as follows. First, a *pattern* is selected, i. e., a subset of automata, integer variables and clock variables. For a given pattern $\mathcal{P} \subseteq \mathcal{M} \cup V \cup X$, a projection abstraction $\mathcal{M}|_{\mathcal{P}}$ of $\mathcal{M}$ is computed, where basically only automata and variables from $\mathcal{P}$ are kept. More precisely, given a system $\mathcal{M}$ and a pattern $\mathcal{P}$, the abstraction $\mathcal{M}|_{\mathcal{P}}$ is computed as follows. Every automaton $\mathcal{A} \in \mathcal{M}$ is replaced by an abstract automaton $\mathcal{A}'$, where $\mathcal{A}'$ is computed as follows. For all integer variables $v \notin \mathcal{P}$ and for all edges e of $\mathcal{A}$, $\mathcal{A}'$ is computed by removing all constraints in the guards and all integer assignments in the effect of e where v occurs. Accordingly, for all clock variables $c \notin \mathcal{P}$, all clock guards and resets where c occurs are removed. Furthermore, for all locations l in $\mathcal{A}$, clock constraints of invariants of l for such clock variables are removed as well. Finally, for all automata $\mathcal{A}$ that do not occur in $\mathcal{P}$, the abstracted automaton $\mathcal{A}'$ consists of only one location that contains self-loop edges, where the self-loop edges are obtained from the edges e of $\mathcal{A}$ by abstracting the guard of e according to the abstraction method described above, while keeping the effects (i. e., the integer assignments and clock resets of e are not removed). This ensures that the resulting abstraction $\mathcal{M}|_{\mathcal{P}}$ is an overapproximation of $\mathcal{M}$. To see that this abstraction method for automata is needed to obtain overapproximations, we provide a small example in Fig. 2, which will also serve as a running example in the paper.

The example shows a DMC problem that consists of a system $\mathcal{S}$ with two parallel automata P and Q. Initially, both automata are in their initial locations p_1 and q_1, respectively (indicated with an incoming edge in Fig. 2). The error property is given by $\varphi = Q.q_4$ (indicated with Q's double circled location q_4 in Fig. 2), i. e., the error states in $\mathcal{S}$ are defined as the set of states where Q has reached location q_4. All action transitions of $\mathcal{S}$ are asynchronous. We observe that an error state is reachable by, e. g., first applying the transitions from q_1 to q_2 and from q_2 to q_3 in Q, then setting a to 1 by applying the corresponding transitions in P, and finally applying the transition from q_3 to q_4 in Q. Suppose we abstract away automaton P. First, we observe that simply throwing P away would result in a system (only consisting of Q) where no error

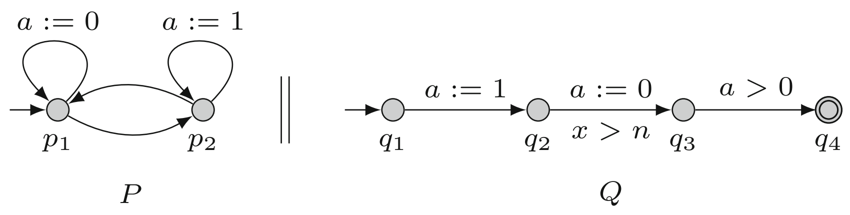

Fig. 2 Running example: System $\mathcal{S}$ consisting of two timed automata P and Q that share an integer variable a and a clock variable x

states are reachable any more because the variable a could not be set to a value greater than zero in location q_3. In contrast, abstracting P as described above yields an automaton P' with two self-loops that set the variable a to 0 and to 1, respectively. We obtain an overapproximation because, e. g., a can be set to 1 in P' already in the initial state.

For the computation of the pattern database, the abstract state space of $\mathcal{M}|_\mathcal{P}$ is enumerated exhaustively and stored in a lookup table (the *pattern database*) together with the abstract error distances for each abstract state. We define the pattern database heuristic $h^\mathcal{P}$ based on $\mathcal{P}$ as follows.

Definition 1 (*Pattern database heuristic*) Consider a DMC problem $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ and a pattern $\mathcal{P}$. The heuristic value $h^\mathcal{P}(s)$ for a state s is defined as

$$h^\mathcal{P}(s) := \min\{\|\pi\| \mid \pi \text{ is error trace in } \Theta^\# \text{ from } s|_\mathcal{P}\},$$

where $\Theta^\# = (\mathcal{M}|_\mathcal{P}, V \cap \mathcal{P}, X \cap \mathcal{P}, s|_\mathcal{P}, \varphi|_\mathcal{P})$ consists of the abstract system of $\mathcal{M}|_\mathcal{P}$, the abstracted variable sets w.r.t. $\mathcal{P}$, the abstract initial state $s|_\mathcal{P}$ (the projection of s onto $\mathcal{P}$), and the abstract property $\varphi|_\mathcal{P}$ (the projection of φ onto $\mathcal{P}$).

We remark that in terms of exact error distances d, we could equivalently define $h^\mathcal{P}(s)$ as $d(s|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$, namely as the minimum number of abstract transitions that is needed to reach an error state from $s|_\mathcal{P}$ to an abstract error state in $\mathcal{M}|_\mathcal{P}$. Overall, Definition 1 provides a general framework how pattern database heuristics are computed. However, in the context of timed systems, the question about the projection from a concrete state s to an abstract state $s|_\mathcal{P}$ remains. More precisely, what is the relationship between the zone of s and $s|_\mathcal{P}$? In general, s might correspond to *several* abstract states that agree on the discrete parts and have a non-empty intersection of the zones. We formally define abstract states $s|_\mathcal{P}$ for networks of timed automata in Sect. 3.

When designing a pattern database heuristic, the most important part is the automatic selection of a suitable pattern. At one extreme end of a spectrum of possible patterns, one could choose the empty pattern, which yields a pattern database heuristic $h^\mathcal{P}$ that is efficiently computable. However, it is obvious that $h^\mathcal{P} \equiv 0$ for all states s, and no further guidance information is obtained. At the other extreme end of the spectrum, one could choose the pattern that contains *all* automata and variables, which yields a pattern database heuristic that is equal to the real error distance function. However, computing this distance heuristic is as hard as solving the original DMC problem, and overall, no performance improvements compared to blind search are obtained either. The challenge we address in this paper is to automatically find "good" patterns that are somewhere in between these extreme ends. Ideally, a pattern should provide a small abstract system such that the abstract state space can be efficiently enumerated on the one hand, and retain as much of the original system behavior as possible on the other hand.

3 Pattern database heuristics for timed automata

In Sect. 2.3, we have introduced the general framework that is commonly used for the construction of pattern database heuristics. However, for networks of timed automata, the problem about the mapping from a state s to a corresponding abstract state remained. In this section, we provide the formal basis to address and to solve this problem. We remark that, although this problem is related to other approaches as well that deal with pattern databases in the context of timed automata, the formal framework and theoretical results that we are going to present in this section have not been stated elsewhere so far.

For a pattern $\mathcal{P}$ and a state s, we define the set of *abstract candidate states* $s^\#$ of the abstract system $\mathcal{M}|_\mathcal{P}$ as

$$\begin{aligned}\mathrm{ACS}(s) = \{&s^\# \text{ abstract state in } \mathcal{M}|_\mathcal{P} \mid dP(s) \models dP(s^\#) \\ &\text{and } \mathrm{Zone}(s) \models \mathrm{Zone}(s^\#)\},\end{aligned}$$

where $dP(s)$ denotes the *discrete part* of s, i. e., a formula that expresses the valuation of s for the automata locations and integer variables, and *Zone*(s) denotes the zone of s. As we are dealing with timed automata, *ACS*(s) contains more than one element in general. Based on this definition, we define a pattern database heuristic for timed automata as follows.

Definition 2 (*Pattern database for timed automata*) Let $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem and $\mathcal{P}$ be a pattern. The heuristic value $h^\mathcal{P}(s)$ for a state s is defined as in Definition 1, where the corresponding abstract state to s is defined as

$$s|_\mathcal{P} = \operatorname*{arg\,max}_{s^\# \in \mathrm{ACS}(s)} d(s^\#, \mathcal{M}|_\mathcal{P}),$$

i. e., s is mapped to the state in ACS(s) that maximizes the abstract error distance.

In the following, we show that the resulting distance heuristic $h^\mathcal{P}(s)$ is consistent. We remark that this result is also related to other approaches that deal with pattern databases and timed automata, but has not been stated elsewhere so far.

Proposition 1 *For a DMC problem $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$, the pattern database heuristic $h^\mathcal{P}$ is consistent, i. e.,*

$$h^\mathcal{P}(s) \le h^\mathcal{P}(s') + 1$$

for all states s and transitions t with $s' = t[s]$, and $h^\mathcal{P}(s_e) =$ 0 for all states $s_e \not\models \varphi$.

Proof First, $h^\mathcal{P}(s_e) = 0$ for all error states s_e holds by definition of abstract error states. In the following, we show that $h^\mathcal{P}(s) \le h^\mathcal{P}(t[s]) + 1$ for all states s and transitions t that are applicable in s. Recall that pattern database heuristics are defined over the real error distance function d in $\mathcal{M}|_\mathcal{P}$, i.e., $h^\mathcal{P}(s) = d(s|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$. As $\mathcal{M}|_\mathcal{P}$ is an overapproximation of $\mathcal{M}$, and by definition of abstract states $s|_\mathcal{P}$, the abstract transition $t|_\mathcal{P}$ that corresponds to t in $\mathcal{M}|_\mathcal{P}$ is applicable in $s|_\mathcal{P}$. As d is consistent, we have $d(s|_\mathcal{P}, \mathcal{M}|_\mathcal{P}) \leq d(t|_\mathcal{P}[s|_\mathcal{P}], \mathcal{M}|_\mathcal{P}) + 1$ for the resulting abstract state $t|_\mathcal{P}[s|_\mathcal{P}]$. Furthermore, $t|_\mathcal{P}[s|_\mathcal{P}] \in ACS(t[s])$ because the discrete part of $t[s]$ implies the discrete part of $t|_\mathcal{P}[s|_\mathcal{P}]$, and $\text{Zone}(t[s]) \models \text{Zone}(t|_\mathcal{P}[s|_\mathcal{P}])$. Therefore, $h^\mathcal{P}(t[s]) \geq d(t|_\mathcal{P}[s|_\mathcal{P}], \mathcal{M}|_\mathcal{P})$. Overall, we can observe that the following equation holds for $h^\mathcal{P}$: $h^\mathcal{P}(s) = d(s|_\mathcal{P}, \mathcal{M}|_\mathcal{P}) \leq d(t|_\mathcal{P}[s|_\mathcal{P}], \mathcal{M}|_\mathcal{P}) + 1 \leq h^\mathcal{P}(t[s]) + 1$. This proves the claim.

As a consequence of the above consistency result, states in the *closed* list of the algorithm shown in Fig. 1 do not have to be considered again in any case, as it is guaranteed that the priority value of states does not have to be updated in any case either. Furthermore, we will exploit this consistency result for our downward pattern refinement approach, which is described in the subsequent sections.

4 Downward pattern refinement: the theory

In this section, we describe the underlying theory that our pattern selection algorithm is based on. As already outlined, the selection of the pattern is crucial for the entire approach because the pattern determines the overall behavior of the resulting pattern database heuristic. Obviously, there is a tradeoff: pattern should be as small as possible (because the abstract state space of the corresponding abstraction has to be enumerated exhaustively to computed the pattern database), but should also reflect the original system as accurately as possible—in other words, the abstraction should be as "similar" to the original system as possible. An obvious question in this context is the question about similarity: What does it mean for a system to be "similar" to an abstract system? In the following, we derive precise, but computationally hard properties of similarity of abstract systems. Furthermore, we provide ways to efficiently approximate these properties in practice. These approximations lend themselves to a pattern selection algorithm, which will be described afterward in Sect. 5.

4.1 Sufficiently and relatively accurate distance heuristics

We derive a precise measure for abstractions to obtain informed pattern database heuristics. As already outlined above, the most important question in this context is the question about similarity. At the extreme end of the spectrum of possible abstractions, one could choose a pattern that leads to bisimilar abstractions to the original system. This yields a pattern database heuristic $h^\mathcal{P}$ that is *perfect*, i.e., $h^\mathcal{P}(s) = d(s)$ for all states s, where d is the real error distance function. However, apart from being not feasible in practice, we will see that this condition is stricter than needed for obtaining perfect search behavior. It suffices to require $h^\mathcal{P}(s) = d(s)$ only for states s that are possibly explored by A*. In this context, Pearl [28] gives a necessary and sufficient condition for a state to be explored by A*. Consider a DMC problem $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ and let $d(s_0)$ denote the length of a shortest error trace of Θ. Recall that the priority function of A* is $priority(s) = h(s) + c(s)$, where $c(s)$ is the length of a shortest trace from s_0 to s. Pearl shows that if h is consistent, and if $priority(s) < d(s_0)$, then s is necessarily explored by A*, whereas exploring s implies that $priority(s) \leq d(s_0)$. This gives rise to the following definition for a distance heuristic to be *sufficiently accurate*.

Definition 3 (*Sufficiently accurate*) Consider a DMC problem $(\mathcal{M}, V, X, s_0, \varphi)$ with shortest error trace length $d(s_0)$. Furthermore, let $\mathcal{P}$ be a pattern, and $h^\mathcal{P}$ be the pattern database heuristic for $\mathcal{P}$. If $h^\mathcal{P}(s) = d(s)$ for all states s with $h^\mathcal{P}(s) + c(s) \leq d(s_0)$, then $\mathcal{M}|_\mathcal{P}$ is called a *sufficiently accurate abstraction* of $\mathcal{M}$, and $h^\mathcal{P}$ is called *sufficiently accurate distance heuristic* for $\mathcal{M}$.

Obviously, the requirement for a distance heuristic $h^\mathcal{P}$ to be sufficiently accurate is weaker than the requirement $h^\mathcal{P}(s) = d(s)$ for *all* possible states. However, with the results given by Pearl, we still know that A* with a PDB heuristic $h^\mathcal{P}$ (which is consistent by Proposition 1) that is also sufficiently accurate delivers perfect search behavior, i.e., the same search behavior as that of A* with d. This justifies Definition 3 and is stated formally in the following proposition.

Proposition 2 *Let $(\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem, $h^\mathcal{P}$ be a distance heuristic that is sufficiently accurate for $\mathcal{M}$. Then the set of explored states with A* applied with d is equal to the set of explored states of A* applied with $h^\mathcal{P}$.*

Proof The claim follows immediately from the results given by Pearl [28] and from Proposition 1. As $h^\mathcal{P}$ is consistent and sufficiently accurate, we know that for every state s that is possibly explored by A* applied with $h^\mathcal{P}$ it holds $h^\mathcal{P}(s) = d(s)$. Therefore, the behavior of A* with d and $h^\mathcal{P}$ is identical.

As an immediate result of the above considerations, it suffices to have patterns that lead to sufficiently accurate distance heuristics to obtain perfect search behavior with A*. On the one hand, this notion is intuitive and reasonable. On the other hand, it is still of rather theoretical nature. It should be obvious that a sufficiently accurate heuristic is hard to compute as it relies on *exact* error distances d; as a side remark, if d was given, the overall model checking problem would be already solved, and there would be no need to compute a pattern database heuristic. However, Definition 3 also provides a first intuitive way for approximating this property, an approximation which is described next.

According to Definition 3, an abstraction $\mathcal{M}|_\mathcal{P}$ is sufficiently accurate if $h^\mathcal{P}(s) = d(s)$ for all states s that are possibly explored by A*. In this case, the pattern database heuristic based on $\mathcal{M}|_\mathcal{P}$ is sufficiently accurate for $\mathcal{M}$. For the following considerations, note that $h^\mathcal{P}(s) = d(s|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$, and therefore, a direct way to approximate this test is to use a *distance heuristic* h instead of d. This is reasonable as distance heuristics are designed exactly for the purpose of approximating d, and various distance heuristics have been proposed in the directed model checking literature. Furthermore, as checking *all* states that are possibly explored by A* is not feasible either, we check this property only for the *initial* system state. This is the only state for which we know a priori that it is explored by A*. Overall, this gives rise to the following definition of *relatively accurate* abstractions.

Definition 4 (*Relatively accurate*) Consider a DMC problem $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ with system $\mathcal{M}$ and initial state s_0. Further, let $\mathcal{P}$ be a pattern of $\mathcal{M}$, and let $\mathcal{M}|_\mathcal{P}$ be the corresponding abstraction to $\mathcal{P}$ with abstract initial state $s_0|_\mathcal{P}$. Furthermore, let h be a distance heuristic, $h(s_0, \mathcal{M})$ the distance estimate of $s_0 \in S(\mathcal{M})$, and $h(s_0|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$ the distance estimate of $s_0|_\mathcal{P} \in S(\mathcal{M}|_\mathcal{P})$. If

$$h(s_0, \mathcal{M}) = h(s_0|_\mathcal{P}, \mathcal{M}|_\mathcal{P}),$$

then $\mathcal{M}|_\mathcal{P}$ is called the *relatively accurate abstraction* of $\mathcal{M}$ induced by h and $\mathcal{P}$.

We observe that the notion of relatively accurate abstractions is a 2-stage approximation for computing sufficiently accurate abstractions. This approximation is obtained by applying a *second* distance heuristic (instead of the exact error distance), and additionally, by checking the necessary distance equation only for the initial state (instead of for all states that are possibly explored by A*). Obviously, the quality of this approximation strongly depends on the quality of the applied distance heuristic h. In the experimental section, we will see that even this rather simple approximation of sufficient accuracy can yield informed abstract systems.

Example 1 Consider again the system $\mathcal{S}$ in Fig. 2, and assume that we want to abstract a, i. e., $\mathcal{P} = \{P, Q, x\}$. Further assume that we want to check if $\mathcal{S}|_\mathcal{P}$ is the relatively accurate abstraction of $\mathcal{S}$ induced by the h^L heuristic [22] and $\mathcal{P}$. The h^L heuristic relaxes the concrete system semantics by assuming that discrete variables become set-valued, and all values obtained during the computation of a trace are unified to this set. Constraints are evaluated to true if there exist corresponding values in these sets. (A more detailed discussion is provided in Sect. 6.) We observe that

$$h^L(s_0, \mathcal{S}) = h^L(s_0|_\mathcal{P}, \mathcal{S}|_\mathcal{P}) = 3$$

for the example system in Fig. 2, i. e., $\mathcal{S}|_\mathcal{P}$ is the relatively accurate abstraction of $\mathcal{S}$ induced by h^L and $\mathcal{P}$: First, we observe that $h^L(s_0, \mathcal{S}) = 3$ because starting in s_0 and after applying the two transitions in Q leading from q_1 to q_3, a has been assigned both values 0 and 1, and hence, the guard in Q's third transition evaluates to true. Second, we observe that $h^L(s_0|_\mathcal{P}, \mathcal{S}|_\mathcal{P}) = 3$ because after abstracting a, the h^L value in the resulting abstraction $\mathcal{S}|_\mathcal{P}$ boils down to the graph distance from q_1 to q_4 in Q (as clocks are ignored by h^L).

4.2 Largely informed distance heuristics

As observed in the last section, the notion of relatively accurate abstractions is a 2-stage approximation for computing sufficiently accurate abstractions. On the positive side, given that the second distance heuristic is cheap to compute, relatively accurate abstractions can be efficiently computed as well. However, relatively accurate abstractions in this general form have two main drawbacks. First, on a theoretical level, we do not get any guarantees on the resulting abstraction because we did not formulate any restrictions to the second distance heuristic. Second, on a practical level for timed automata, although there are several distance heuristics "on the market" (as discussed in the introduction), many of these distance heuristics are not able to properly deal with clock variables. In more detail, there are powerful distance heuristics that have been proposed for timed automata that just *ignore* the clock variables, and focus on the automata and integer variables instead [9,22]. However, considering the definition of relatively accurate abstractions, we observe that if the applied distance heuristic does ignore clocks, then abstracting *all* clocks will always lead to relatively accurate abstractions. Therefore, relatively accurate abstractions with respect to clock-ignoring distance heuristics are suited best for the data (i. e., automaton and integer) part of a given system of timed automata, whereas more sophisticated methods are needed for the clock part. In the following, as a special case of the generic definition of relatively accurate abstractions, we formulate a more fine grained criterion to overcome these limitations. This criterion will turn out to be more expensive to compute than the general definition, but the resulting abstractions will satisfy certain quality guarantees.

Let $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem and $\mathcal{P}$ be a pattern for Θ. Furthermore, recall that $c(s)$ denotes the depth of a state s, and that the following results given by Pearl [28] hold for A*. If $h(s) + c(s) < d(s_0)$ for a consistent distance heuristic h and a state s, then s is definitely explored by A*, whereas if $h(s) + c(s) > d(s_0)$, s is definitely *not* explored. For states s with $h(s) + c(s) = d(s_0)$, it depends on the tie-breaking of the priority queue *open* if s is explored or not. This gives rise to the following definition.

Definition 5 (*Largely informed*) Let h_1 and h_2 distance functions for a DMC problem Θ. Then h_1 is *largely as informed as* h_2 if $\{s \mid h_1(s) + c(s) < d(s_0)\} = \{s \mid h_2(s) + c(s) < d(s_0)\}$, and $\{s \mid h_2(s) + c(s) = d(s_0)\} \subseteq \{s \mid h_1(s) + c(s) = d(s_0)\}$.

According to the above definition, h_1 is largely as informed as h_2 if the set of states that is definitely explored by A* with h_1 is the same as with h_2, and only *some* states may be explored by h_1 and not by h_2 depending on the implementation of the priority queue. From the consistency result of Proposition 1, it follows that a pattern database heuristic $h^{\mathcal{P}}$ is largely as informed as the real error distance function d if $\mathcal{M}|_{\mathcal{P}}$ is the relatively accurate abstraction of $\mathcal{M}$ induced by $\mathcal{P}$ and the real error distance function d. This is formally stated in the following proposition.

Proposition 3 *Let $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem. Let $\mathcal{P}$ be a pattern with $\mathcal{P} \subseteq \mathcal{M} \cup V \cup X$. If $\mathcal{M}|_{\mathcal{P}}$ is the relatively accurate abstraction of $\mathcal{M}$ induced by $\mathcal{P}$ and the real error distance function d, i. e., if*

$$h^{\mathcal{P}}(s_0) = d(s_0),$$

then $h^{\mathcal{P}}$ is largely as informed as d.

Proof When A* is applied with d, then $d(s) + c(s) \geq d(s_0)$ for all reachable states s.[1] We show by induction over $c(s)$ that the same holds for $h^{\mathcal{P}}$, i. e., $h^{\mathcal{P}}(s) + c(s) \geq d(s_0)$ for all reachable states s from s_0. For the base case, we have $c(s) = 0$ for $s = s_0$. By assumption, $h^{\mathcal{P}}(s_0) = d(s_0)$, therefore $h^{\mathcal{P}}(s_0) \geq d(s_0)$. For the induction step, consider states s and s' with $c(s') = c(s)+1$. In Proposition 1, we have shown that $h^{\mathcal{P}}$ is consistent, i. e., $h^{\mathcal{P}}(s) \leq h^{\mathcal{P}}(s') + 1$. It follows that $h^{\mathcal{P}}(s') + c(s') \geq h^{\mathcal{P}}(s) + c(s)$. By induction hypothesis, $h^{\mathcal{P}}(s) + c(s) \geq d(s_0)$. Furthermore, $\{s \mid d(s) + c(s) = d(s_0)\} \subseteq \{s \mid h^{\mathcal{P}}(s) + c(s) = d(s_0)\}$: Assume there is a state s with $d(s) + c(s) = d(s_0)$, but $h^{\mathcal{P}}(s) + c(s) > d(s_0)$. Then $h^{\mathcal{P}}(s) > d(s)$, which contradicts that $h^{\mathcal{P}}$ is admissible.

From Proposition 3, it follows that it actually suffices to have pattern database heuristics that agree with d on the initial state to get largely as informed distance heuristics as d. In other words, specializing the (general) 2-stage approximation of sufficiently accurate abstractions (where we do not get any guarantees) to a 1-stage approximation where the exact error distance is used as a distance function yields abstractions that already *guarantee* to be largely as informed as the real error distance function. Let us point out the relationship of relatively accurate abstractions and largely informed distance heuristics in more detail. For general relatively accurate abstractions, we seek abstractions with the property that the *estimated* error distance of the initial abstract state is equal to the *estimated* error distance in the original system w.r.t. a second distance heuristic. This technique is used as an *approximation* of sufficiently accurate abstractions, and we do not get any guarantees for the resulting (relatively accurate) abstraction. As a special case, we can use the exact error distance function as a second "distance heuristic", which *guarantees* to obtain largely informed distance heuristics.

Example 2 Consider again the system $\mathcal{S}$ in Fig. 2, and assume that we want to abstract a, i. e., $\mathcal{P} = \{P, Q, x\}$. We observe that $h^{\mathcal{P}}(s_0) = 3$ as we only need the three transitions in Q to reach q_4 from q_1 in $\mathcal{S}|_{\mathcal{P}}$. In contrast, $d(s_0) = 5$, i. e., the exact error distance from s_0 is equal to 5 in the original system $\mathcal{S}$ because we additionally need to apply two transitions in P to set a to 1 in order to apply the transition from q_3 to q_4 in Q. Hence, we cannot conclude with Proposition 3 that $h^{\mathcal{P}}$ is largely as informed as d.

Proposition 3 particularly allows us to deal with clock variables explicitly because we do no longer rely on clock-ignoring distance heuristics. Overall, this suggests to use relatively accurate abstractions (that are cheaply computable) for the data part of the system, whereas for the timed part, a more refined approach motivated by largely informed distance heuristics (which is more expensive to compute, but able to properly deal with clock variables) is suitable. We will come back to these points in Sect. 5.

We close the section with the observation that for a restricted class of timed automata, abstracting clock variables x that only occur in simple clock constraints of the form $x > n$ for a fixed $n \in \mathbb{N}_0$ yields abstractions that are relatively accurate with respect to d and hence, with Proposition 3, yields pattern database heuristics that are largely as informed as d.

Proposition 4 *Let $\Theta = (\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem with $\mathcal{M} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$. Let $n \in \mathbb{N}_0$ be a natural number. Let $x \in X$ be a clock variable with the property that in guards of transitions, x only occurs in clock constraints of the form $x > n$, and there are no invariants in the automata $\mathcal{A}_i$ that contain x. Furthermore, for all $y \in X$, there are no constraints (i. e., clock guards or invariants) in $\mathcal{M}$ of the form $y \bowtie m$ for $\bowtie \in \{<, \leq, =\}$ and $m \in \{0, \ldots, n\}$. Let $\mathcal{P} = \mathcal{M} \cup V \cup X \setminus \{x\}$. Then $h^{\mathcal{P}}(s_0) = d(s_0)$.*

Proof Without loss of generality, $\varphi = \bigwedge_{i=1}^{k} \varphi_i$, where each φ_i is a location constraint (see the preliminaries section). We show that for all states s and transitions t: if $t|_{\mathcal{P}}$ is applicable in $s|_{\mathcal{P}}$, then t is applicable in s. Assume there is a state s and a transition t such that $t|_{\mathcal{P}}$ is applicable in $s|_{\mathcal{P}}$, but t is not applicable in s. As $\mathcal{P} = \mathcal{M} \cup V \cup X \setminus \{x\}$, s and $s|_{\mathcal{P}}$ have the same discrete part, and Zone$(s) \models$ Zone$(s|_{\mathcal{P}})$. It follows

[1] Note that, nevertheless, given that the tie-breaking criterion of *open* is chosen such that states with lower d values are explored first in case of equal lowest priority values, a shortest possible error trace π^* is found by A* with a linear number of state explorations in the length of π^*.

that there exists a clock constraint c in the clock guard of t or in the invariant of one of the source locations of t such that $\text{Zone}(s) \not\models c$ and $\text{Zone}(s|_{\mathcal{P}}) \models c$ (otherwise, t would be applicable in s as well). As x does not occur in invariants and only in the form $x > n$, it follows that for all clocks $y \in X$, only the lower bound of abstract zones are affected $(\text{lowerBound}(y, \text{Zone}(s|_{\mathcal{P}})) \leq \text{lowerBound}(y, \mathit{Zone}(s)))$, whereas the upper bound of y remains unchanged (formally: $\text{upperBound}(y, \text{Zone}(s|_{\mathcal{P}})) = \text{upperBound}(y, \text{Zone}(s)))$. Therefore, c must be of the form $y \bowtie m$ for a clock y, $\bowtie \in \{<, \leq, =\}$, and $m \in \{0, \dots, n\}$. However, such constraints do not exist by assumption.

Proposition 4 implies that a pattern selection algorithm can abstract clock variables that only occur in simple constraints (corresponding to the assumptions of Proposition 4) to obtain distance heuristics that are largely as informed as the real error distance function. Intuitively, abstracting such clock variables does not introduce shortcuts because in the original system, we only have to let time pass (if necessary) to satisfy such constraints.

Example 3 Consider again the system $\mathcal{S}$ in Fig. 2. Let $n \in \mathbb{N}_0$ be a natural number. We observe that the guard $x > n$ satisfies the requirements of Proposition 4, and hence, $h^{\mathcal{P}}(s_0) = d(s_0)$ for the pattern $\mathcal{P} = \{P, Q, a\}$. With Proposition 3, it follows that $h^{\mathcal{P}}$ is largely as informed as d.

4.3 Concretizable traces and safe abstractions

In addition to the criteria from the last sections, we derive a sufficient criterion for a distance heuristic to be sufficiently accurate that is still weaker than the requirement $h^{\mathcal{P}}(s) = d(s)$ for *all* states s. It is based on the observation that abstract systems where every spurious error trace is longer than $d(s_0)$ are not harmful.

Proposition 5 *Let $(\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem, $\mathcal{P}$ be a pattern such that every spurious error trace π in the corresponding abstraction $\mathcal{M}|_{\mathcal{P}}$ is longer than a shortest possible error trace in $\mathcal{M}$, i.e., $\|\pi\| > d(s_0)$. Then $h^{\mathcal{P}}$ is sufficiently accurate for $\mathcal{M}$.*

Proof First, recall that $h^{\mathcal{P}}(s) \leq d(s)$ for all states $s \in S(\mathcal{M})$ because $\mathcal{M}|_{\mathcal{P}}$ is an overapproximation of $\mathcal{M}$. We show that $h^{\mathcal{P}}(s) + c(s) > d(s_0)$ for all states $s \in S(\mathcal{M})$ with $h^{\mathcal{P}}(s) < d(s)$. Assume $h^{\mathcal{P}}(s) < d(s)$ for a state $s \in S(\mathcal{M})$. Let $s|_{\mathcal{P}} \in S(\mathcal{M}|_{\mathcal{P}})$ be the corresponding abstract state to s. As $h^{\mathcal{P}}(s) < d(s)$, there is an abstract trace $\pi_{\mathcal{P}}$ that is spurious and contains $s|_{\mathcal{P}}$. As all spurious error traces are longer than $d(s_0)$ by assumption, we have $\|\pi_{\mathcal{P}}\| > d(s_0)$. Therefore, $\|\pi_{\mathcal{P}}\| = c^{\mathcal{P}}(s|_{\mathcal{P}}) + d^{\mathcal{P}}(s|_{\mathcal{P}}) > d(s_0)$, where $c^{\mathcal{P}}(s|_{\mathcal{P}})$ denotes the length of a shortest abstract trace from the initial abstract state to $s|_{\mathcal{P}}$, and $d^{\mathcal{P}}(s|_{\mathcal{P}})$ denotes the abstract error distance of $s|_{\mathcal{P}} \in S(\mathcal{M}|_{\mathcal{P}})$. As $d^{\mathcal{P}}(s|_{\mathcal{P}}) = h^{\mathcal{P}}(s)$ and $c(s) \geq c^{\mathcal{P}}(s|_{\mathcal{P}})$, we have $c(s) + h^{\mathcal{P}}(s) > d(s_0)$.

Again, identifying abstractions with the property given by the above proposition is computationally hard as it relies on checking *all* possible spurious error traces. In the following, we show that a subclass of abstractions for a slightly stronger condition can be identified efficiently. To be more precise, we focus on abstractions that only introduce spurious error traces that can be *concretized* in the following sense.

Definition 6 (*Concretizable Trace*) Consider a DMC problem $(\mathcal{M}, V, X, s_0, \varphi)$. Let $\mathcal{P}$ be a pattern, and $\mathcal{M}|_{\mathcal{P}}$ be the corresponding abstraction of $\mathcal{M}$. Let $\pi_{\mathcal{P}} = t_1^{\#}, \dots, t_n^{\#}$ be an abstract error trace of $\mathcal{M}|_{\mathcal{P}}$ with corresponding concrete transitions $t_1, \dots, t_n$ of $\mathcal{M}$. Let $\pi_{\mathcal{P}}$ be spurious, i.e., $t_1, \dots, t_n$ is not a concrete error trace of $\mathcal{M}$. The error trace $\pi_{\mathcal{P}}$ is *concretizable* in $\mathcal{M}$ if and only if there is a concrete error trace

$$\pi = \pi_0, t_1, \pi_1, t_2, \pi_2, \dots, \pi_{n-1}, t_n, \pi_n$$

in $\mathcal{M}$ that embeds $t_1, \dots, t_n$. The π_i are traces in $\mathcal{M}$ with $\|\pi_i\| \geq 0$, for $i \in \{0, \dots, n\}$.

Informally speaking, an abstract trace $\pi_{\mathcal{P}}$ in $\mathcal{M}|_{\mathcal{P}}$ is concretizable in $\mathcal{M}$ if there is a concrete trace in $\mathcal{M}$ so that the corresponding abstract trace in $\mathcal{M}|_{\mathcal{P}}$ is equal to $\pi_{\mathcal{P}}$. Note that from the above definition, concretizable error traces are a subclass of spurious error traces; as a side remark, these are exactly those error traces that preserve dead ends in $\mathcal{M}$, i.e., states from which no error state is reachable. In the following, we focus on finding abstractions that do not introduce error traces that are not concretizable. We observe that *safe abstraction* is an effective technique for this purpose.

Safe abstraction for directed model checking has been introduced by Wehrle and Helmert [34]. Essentially, automata and integer variables identified by safe abstraction can change their values independently of and without affecting any other automaton or variable, and every possible value of its domain is reachable. In the following, we briefly give a declarative definition of safe automata and safe integer variables that is sufficient for the need of this paper. For a more detailed description, the reader is referred to the literature [34].

Definition 7 (*Safe automata and safe integer variables*) Let $(\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem.

1. An automaton $\mathcal{A} \in \mathcal{M}$ is called *safe* if all of its locations are reachable from all other locations independently of the current locations of other automata, and independently of the current values of the variables.
2. An integer variable v is called *safe* if all of its values of its domain can be obtained, independently of the current

locations of the automata, and independently of the values of all variables (including the values of v).

We observe that being safe is a rather strict requirement. For example, integer variables v and w can only be safe if there is no transition that reads v and writes to w, there is no transition that reads w and writes to v, and additionally, there must not be a transition that both writes to v and w. However, safe automata and safe integer variables can be efficiently identified by a static analysis of the causal dependencies of the automata and integer variables of the given system.

Example 4 Consider again the system $\mathcal{S}$ in Fig. 2. We observe that a is a safe integer variable because it can reach all of its possible values in $\{0, 1\}$ independently of the current locations and values of the other automata and variables.

Wehrle and Helmert exploit this property by performing directed model checking directly on a system obtained by safe abstraction (i. e., on the abstract system that is obtained by removing safe automata and safe integer variables), where abstract error traces in $\mathcal{M}|_\mathcal{P}$ are finally extended to concrete error traces in $\mathcal{M}$. Doing so, however, is not optimality preserving: shortest abstract error traces in $\mathcal{M}|_\mathcal{P}$ may not correspond to *any* shortest error trace in $\mathcal{M}$.

In this work, we use safe abstraction in a different context, namely to select patterns for a pattern database. In particular, we will observe that safe abstraction offers desirable properties for the pattern selection. Safe abstractions have the property that every abstract error trace can be concretized. This is summarized in the following proposition. A proof is given by Wehrle and Helmert [34].

Proposition 6 *Let $(\mathcal{M}, V, X, s_0, \varphi)$ be a DMC problem and let p be a safe automaton or a safe integer variable of $\mathcal{M}$. Let $\mathcal{P}$ be the pattern that is obtained from the full pattern where p has been removed, and let $\mathcal{M}|_\mathcal{P}$ be the corresponding abstract system. Then every abstract error trace in $\mathcal{M}|_\mathcal{P}$ is concretizable.*

We observe that under the assumptions of Proposition 6, the set of unconcretizable abstract error traces is empty, and of course, the same holds for the set of shorter or equally long abstract traces $\{\pi_\mathcal{P} \mid \pi_\mathcal{P}$ is not concretizable and $\|\pi_\mathcal{P}\| \leq d(s_0)\}$. In other words, abstracting safe automata and safe integer variables does not introduce error traces that are longer than $d(s_0)$ and that are not concretizable. Therefore, we observe that safe abstraction provides an effective technique to approximate Proposition 5, where the condition of spuriousness is strengthened to concretizability. The causal analysis required for safe abstraction can be done statically, is cheap to compute, and identifies system components with the property that corresponding abstract systems approximate the conditions of Proposition 5. Overall, we observe that safe abstraction can be effectively applied to pattern selection and hence, to find shortest possible error traces with PDBs and A*, which is a different purpose than it was originally introduced.

As a summary of this section, we have introduced concepts to systematically compute abstract systems for pattern database heuristics. In the next section, we formulate a corresponding pattern selection algorithm.

5 Downward pattern refinement: the algorithm

In the last section, we have provided the theoretical background for a pattern selection algorithm for timed automata: Based on the concepts provided by Definition 4 and Propositions 3, 4 and 6, we have derived tractable criteria to estimate the similarity of abstractions for the purpose of computing pattern database heuristics. Based on these criteria, we present the algorithm *dprc* (*d*ownward *p*attern *r*efinement with *c*locks) in Fig. 3. The resulting pattern database heuristic is called h^{dprc}. Starting with a large pattern $\mathcal{P}$, the focus of the *dprc* algorithm is to refine the current pattern by finding smaller patterns $\mathcal{P}' \subseteq \mathcal{P}$ that yield pattern database heuristics $h^{\mathcal{P}'}$ such that the resulting pattern satisfies the criteria from

Fig. 3 The downward pattern refinement algorithm for pattern selection

```
1  function dprc ($\mathcal{M}, V, X, s_0, \varphi, h, c$):
2      $\mathcal{P} := \mathcal{M} \cup V \cup X \setminus \{x \in X \mid$ in clock guards, $x$ occurs only in simple constraints $\}$
3      $\mathcal{P} := \mathcal{P} \setminus (\{v \mid v$ safe integer variable in $\mathcal{M}\} \cup \{p \mid p$ safe automaton in $\mathcal{M}\})$
4      if (error distance $d(s_0|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$ can be identified within $c$ seconds) then:
5          $\mathcal{P} := \mathcal{P} \setminus \{x \in X \mid h^\mathcal{P}$ is largely as informed as $h^{\mathcal{P}\setminus\{x\}}\}$
6          return $\mathcal{P}$
7      for each $v \in \mathcal{P}$ do:
8          if $h(s_0, \mathcal{M}) = h(s_0|_{\mathcal{P}\setminus\{v\}}, \mathcal{M}|_{\mathcal{P}\setminus\{v\}})$ then:
9              $\mathcal{P} := \mathcal{P} \setminus \{v\}$
10             goto 7
11     if (error distance $d(s_0|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$ can be identified within $c$ seconds) then:
12         $\mathcal{P} := \mathcal{P} \setminus \{x \in X \mid h^\mathcal{P}$ is largely as informed as $h^{\mathcal{P}\setminus\{x\}}\}$
13     return $\mathcal{P}$
```

the last section. The algorithm is parametrized with a DMC problem, a distance heuristic h to compute relatively accurate abstractions, and a constant $c \in \mathbb{N}$. To compute the pattern, the algorithms starts with the full pattern $\mathcal{P} = \mathcal{M} \cup V \cup X$. If $\mathcal{M}$ satisfies the preconditions of Proposition 4, we first remove clocks x if in every clock guard of every transition, x only occurs in simple clock constraints (line 2). Afterward, based on Proposition 6, we remove automata and integer variables that are safe in $\mathcal{M}$ (line 3). Subsequently, motivated by Proposition 3, we try to find out the length $d(s_0|_{\mathcal{P}}, \mathcal{M}|_{\mathcal{P}})$ of a shortest error trace within c seconds in the resulting abstraction $\mathcal{M}|_{\mathcal{P}}$ (line 4). Therefore, we apply *search* in this abstraction (i. e., the algorithm given in Fig. 1). If it is possible to find such a shortest error trace, we check for every clock variable x if removing x yields a largely as informed distance heuristic as $h^{\mathcal{P}}$; more precisely, we check if $d(s_0|_{\mathcal{P}}, \mathcal{M}|_{\mathcal{P}}) = d(s_0|_{\mathcal{P}\setminus\{x\}}, \mathcal{M}|_{\mathcal{P}\setminus\{x\}})$, and remove x from $\mathcal{P}$ if this is the case (line 4–5). If it is not possible to identify $d(s_0|_{\mathcal{P}}, \mathcal{M}|_{\mathcal{P}})$ within c seconds, then we remove automata and variables that yield relatively accurate abstractions according to h (line 7–10) in order to get a smaller abstraction. In the resulting abstraction, we try again to find a shortest error trace within c seconds and remove the corresponding clock variables (line 11–13).

6 Related work

Directed model checking has found increasing attention in the last years. A survey on this topic is provided by Edelkamp et al. [14]. In particular, directed model checking has been applied to timed systems.

The most related approach to this work is probably the "Russian Doll" abstraction that has been proposed by Kupferschmid et al. [23]. In this approach, the pattern selection problem is addressed by computing an abstract error trace π using the so-called *monotonicity abstraction* [22], and then collecting all variables that occur in π for the pattern. The underlying idea is that variables that occur in the abstract error trace π are likely to be important for the abstract system as well. In contrast to our work, Kupferschmid et al. do not iteratively refine the pattern. A further approach based on pattern databases has been proposed by Qian and Nymeyer [30]. Starting with the variables that occur in the property φ, Qian and Nymeyer select a pattern by using a cone-of-influence analysis up to a certain depth. The underlying assumption of this approach is that variables that are "closer" to φ in this sense are more important for the PDB, which is intuitively true for the variables that do occur in φ. Moreover, pattern database heuristics have also been proposed in the more general setting of predicate abstraction by Smaus and Hoffmann [32]. More precisely, Smaus and Hoffmann use predicate abstraction (rather than projection abstractions) to generate the abstract state space. The pattern selection problem generalizes to the question, which predicates should be used for the abstraction. Smaus and Hoffmann propose to use abstraction refinement for this purpose.

In addition, PDB heuristics play an important role in the area of AI planning. For planning, PDB heuristics have been introduced by Edelkamp [10]. In addition, Haslum et al. [17] have provided a powerful algorithm for selecting pattern collections based on a search in the pattern space.

A further abstraction-based, but non-PDB approach called *distance preserving abstractions* has been proposed by Dräger et al. [8,9]. For a concurrent system of timed automata, distance preserving abstractions are computed by successively computing the cross-product of two automata. This process is interleaved by merging states in the resulting abstraction to avoid the state explosion problem, where the merging is performed until a certain threshold provided by the user is no longer exceeded. This procedure is repeated until only one automaton is left, which represents the final abstraction. Distance heuristics based on distance preserving abstractions have been generalized to the area of automated planning by Helmert et al. [18]. In the planning community, such distance heuristics are called *merge-and-shrink heuristics*.

Apart from abstraction-based distance heuristics like PDB heuristics and the distance preserving abstraction heuristic, approaches based on other simplifications of the original system have been proposed. One of these approaches is based on the above-mentioned monotonicity abstraction proposed for directed model checking by Kupferschmid et al. [22]. First, let us remark that despite the name, the monotonicity abstraction is not an abstraction in the sense we described so far, but rather a *relaxation* of the original system semantics with the following underlying idea: System variables keep track of the values achieved so far, and transitions are applicable if there is a combination of so far achieved values such that the guards are satisfied. Starting in the current state s, distance heuristics based on this approach iteratively apply all transitions that are applicable under this relaxed semantics until an error state or a fixed point is reached. Kupferschmid et al. propose the two distance heuristics h^L and h^U that exploit this principle in different ways. We remark that clock variables are ignored because their values trivialize rather quickly under this abstraction. More precisely, as soon as a location is reached with no invariant for clock x that restricts its values, the value range of x is equal to all nonnegative real values. A further distance heuristic based on analyzing the causal dependencies of the given system is proposed by Wehrle and Helmert [34]. This *causal graph heuristic* computes an approximation of the number of steps that are needed to achieve the faulty values of the variables that occur in the error property. For the computation, the corresponding steps that are needed to achieve the required values of the transitions' preconditions are taken into account recursively. Again, the clock variables are ignored by the causal graph heuristic. A further simplification of the causal graph heuristic is proposed by Edelkamp et al. [12]. In this approach, the graph distance of automata that occur in the error property are considered, whereas other variables and synchronization behavior is ignored completely.

In addition, external search has been applied for model checking real-time systems. In this context, Edelkamp and Jabbar [11] have proposed three external search algorithms for priced timed automata, which they have implemented in the `Uppaal Cora` model checker.

7 Experiments

In this section, we provide an experimental analysis of downward pattern refinement which we have implemented in the MCTA model checker. This model checker is described in more detail in Sect. 7.1. Furthermore, the benchmarks are described in Sect. 7.2. Finally, the experimental setup and results are presented in Sect. 7.3.

7.1 The MCTA model checker

MCTA [25,36] is a model checking tool for concurrent timed automata systems. It is released under the GPL and available at http://mcta.informatik.uni-freiburg.de.

MCTA supports various (heuristic) search algorithms and distance heuristics. Furthermore, MCTA currently supports a subset of the input language that is supported by the UPPAAL model checker. For the evaluation of our approach, we have implemented the general framework for pattern database heuristics, and the algorithm for downward pattern refinement in particular.

7.2 Benchmarks

The first set of benchmarks stems from an industrial case study called "Single-Tracked Line Segment", which comes from an industrial project partner of the UniForM-project [20]. It models a distributed real-time controller for a segment of tracks where trams share a particular piece of track. A distributed controller is supposed to ensure that there cannot be trams in the critical section simultaneously if they drive in different directions. The controller has been modeled in terms of PLC automata [7], which have been afterward transformed with the tool MOBY/RT [27] to concurrent systems of timed automata. For the evaluation of our approach, we chose the property that never both directions are given permission to enter the shared segment simultaneously. We use three problem families to evaluate our approach, denoted with C, D, and E. The problem families differ as they have been obtained by applying different abstractions to the case study. For each of them, nine models of increasing size have been constructed, where the increasing size is obtained by decreasing the number of abstracted variables. The total number of variables in the C instances ranges from 15 to 28, the number of automata ranges from 5 to 10. The corresponding numbers in the D problems range from 29 to 54 (variables) and from 7 to 13 (automata). The E instances have 44 to 54 variables and 9 to 13 automata. For evaluating our directed model checking approach, an error into the C and D problems has been inserted by manipulating an upper time bound. The E instances are correct with respect to the chosen property. A further set of benchmarks stems from a case study called "Mutual Exclusion". This case study models a real-time controller that is supposed to ensure mutual exclusion in a distributed system, where the system components communicate via asynchronous communication. The protocol is described in more detail by Dierks [6]. We use two problem families called M and N, in which an error has been inserted. All these benchmarks are publicly available on the MCTA website.

7.3 Results and discussion

We have evaluated our implementation on a machine with an AMD Opteron Processor 6174 with 2.2 GHz, using a memory bound of 4 gigabyte. In addition to the theory described in the last section, our implementation uses the following optimizations. To filter more clock variables, we extend the largely informedness check (line 5 and line 12 in Fig. 3) such that if no clock is found to be abstracted, we additionally check every clock x again, where the corresponding abstraction is then obtained by abstracting only the clock guards that contain x, and keeping the invariants. Furthermore, the preconditions of Proposition 4 can sometimes be relaxed in practice and still guarantee correctness. In the following, we compare the resulting pattern database heuristic h^{dprc} with the h^L distance heuristic [22], which is also implemented in MCTA. Furthermore, we compare h^{dprc} to the Russian Doll heuristic h^{rd} [23] and to the distance preserving abstraction approach h^{aa} [9] as implemented in the tool UPPAAL/DMC [21]. Finally, we also compare MCTA and h^{dprc} to the efficient implementation of (uninformed) breadth-first search provided by the UPPAAL model checker (version 4.0.13). Table 1 shows the results for erroneous instances, where we set $c = 1$ second in the *dprc* algorithm (see below for a discussion for higher c values). For the computation of relatively accurate abstractions, we use the h^U distance heuristic [22], which is among the strongest (non-admissible) distance heuristics offered by MCTA. Note that although h^U is *not* admissible (and hence, A* with h^U is not guaranteed to find shortest possible error traces), h^U is well suited for computing *patterns* based on

Table 1 Results for erroneous instances

Inst.	Runtime in seconds					Explored states					Trace length
	h^{dprc}	h^L	h^{rd}	h^{aa}	UPPAAL	h^{dprc}	h^L	h^{rd}	h^{aa}	UPPAAL	
M_1	2.2 (0.3)	0.6	3.0 (0.2)	0.3	0.5	29,029	41,455	190	21,945	14,290	47
M_2	2.9 (0.9)	2.6	3.2 (0.2)	0.8	2.1	99,528	164,856	4,417	70,361	51,485	50
M_3	3.7 (1.7)	3.0	3.4 (0.4)	1.3	2.2	165,336	189,820	11,006	108,968	52,987	50
M_4	8.2 (6.2)	13.5	4.0 (1.0)	4.6	8.8	549,999	724,030	41,359	388,475	186,435	53
N_1	2.6 (0.1)	2.7	18.0 (0.4)	2.5	3.8	3,606	93,951	345	46,996	28,196	49
N_2	3.1 (0.6)	14.7	12.1 (0.5)	9.8	17.1	26,791	438,394	3,811	183,215	100,078	52
N_3	4.2 (1.7)	19.1	14.7 (4.5)	13.1	17.5	70,439	547,174	59,062	250,928	102,124	52
N_4	13.0 (10.4)	95.3	34.3 (27.8)	55.9	76.4	388,076	2,317,206	341,928	1,014,036	370,459	55
C_1	1.3 (0.1)	0.2	0.8 (0.1)	0.2	0.2	98	12,458	130	8,649	21,008	54
C_2	1.4 (0.1)	0.7	1.1 (0.7)	0.4	0.5	98	32,751	89,813	21,719	55,544	54
C_3	1.4 (0.1)	0.8	0.8 (0.0)	0.4	0.6	98	37,126	197	28,753	74,791	54
C_4	1.4 (0.1)	7.5	0.9 (0.1)	2.0	6.0	312	301,818	1,140	328,415	553,265	55
C_5	1.5 (0.1)	60.9	1.0 (0.1)	13.4	53.1	1,178	2,174,789	7,530	2,466,025	3,977,279	56
C_6	1.5 (0.1)	605.6	1.1 (0.3)	166.0	514.3	2,619	20,551,913	39,436	24,754,930	33,526,538	56
C_7	1.6 (0.1)	–	1.7 (0.8)	–	–	4,247	–	149,993	–	–	56
C_8	1.6 (0.1)	–	1.7 (0.9)	–	–	5,416	–	158,361	–	–	56
C_9	1.7 (0.2)	–	1.7 (0.8)	–	–	13,675	–	127,895	–	–	57
D_1	9.2 (0.3)	81.2	84.7 (65.0)	2.6	90.5	2,789	1,443,874	4,610,240	256,683	4,048,866	78
D_2	11.2 (0.4)	433.4	255.3 (5.4)	12.6	539.0	5,086	6,931,937	4,223	1,413,123	21,478,364	79
D_3	11.2 (0.4)	487.0	255.6 (5.4)	12.8	548.4	5,161	7,900,038	2,993	1,401,701	21,553,760	79
D_4	12.8 (0.3)	288.0	256.7 (5.4)	10.8	476.4	1,023	4,660,652	2,031	1,285,670	18,487,819	79
D_5	58.9 (6.4)	–	–	–	–	122,204	–	–	–	–	102
D_6	65.4 (10.5)	–	–	–	–	426,571	–	–	–	–	103
D_7	66.1 (7.9)	–	–	–	–	180,132	–	–	–	–	104
D_8	67.4 (6.3)	–	–	–	–	28,285	–	–	–	–	104
D_9	70.5 (6.3)	–	–	–	–	12,186	–	–	–	–	105

Runtime overall runtime including any preprocessing in seconds, *explored states* number of explored concrete states, *trace length* length of a shortest error trace, dashes indicate out of memory (>4 gigabyte). For the h^{dpr} and the h^{rd} distance heuristic that rely on PDBs, the pure search time in the concrete (i. e., time without preprocessing) is reported in parenthesis

relatively accurate abstractions, and hence, for computing the admissible pattern database heuristic h^{dprc}.

First, let us discuss the results of h^{dprc} compared to related directed model checking approaches (see below for a discussion with the UPPAAL results). The experimental data shows that MCTA with the h^{dprc} distance heuristic outperforms the other distance heuristics on these benchmark sets. Although most of the overall runtime is often spent for computing the pattern database heuristic (i. e., for preprocessing), we observe that this preprocessing mostly pays off in better search guidance and overall model checking runtime. Furthermore, comparing the results of h^{dprc} to the results provided by the UPPAAL model checker, we observe that significant improvements are obtained in many instances as well. However, especially in the M and N problems, we also observe that the number of explored states is often (remarkably) higher than with UPPAAL. Although we are not aware of the exact reason, we suppose that this is the case because UPPAAL uses a more efficient representation of the zone graph than MCTA. Finally, let us have a closer look at the c parameter of the *dprc* algorithm. As mentioned above, the results in Table 1 have been obtained by setting $c = 1$, which means that one second is spent for searching for exact error distances $d(s_0|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$ in intermediate abstractions $\mathcal{M}|_\mathcal{P}$. It turns out that even for this rather low bound of c, exact error distances could often identified (M, N, and C), whereas this was not possible in the D problem family. Increasing this c bound leaves the results for M, N and C unchanged, whereas the total runtime increases (linearly with c) in the D problems as long as $d(s_0|_\mathcal{P}, \mathcal{M}|_\mathcal{P})$ can still not be determined.

Table 2 shows the results for problem instances that are correct with respect to the given property. We observe that similar to the results in Table 1, MCTA with our h^{dprc} distance heuristic outperforms the other approaches on the E problems. At first glance, this might look astonishing—how can a distance heuristic (which is rather supposed to guide the

Table 2 Results for correct instances

Inst.	Runtime in seconds					Explored states				
	h^{dprc}	h^{L}	h^{rd}	h^{aa}	UPPAAL	h^{dprc}	h^{L}	h^{rd}	h^{aa}	UPPAAL
E_1	0.8	0.7	0.5	0.2	0.4	0	28,858	1	24,842	43,108
E_2	31.9	289.0	–	65.5	183.1	1	5,951,782	–	6,367,202	11,015,906
E_3	64.3	–	–	–	–	1	–	–	–	–
E_4	167.5	–	–	–	–	1	–	–	–	–
E_5	168.6	–	–	–	–	1	–	–	–	–
E_6	173.9	–	–	–	–	2	–	–	–	–
E_7	181.8	–	–	–	–	5	–	–	–	–
E_8	186.1	–	–	–	–	13	–	–	–	–
E_9	194.5	–	–	–	–	33	–	–	–	–

Abbreviations as in Table 1

search) efficiently be applied for proving correctness? The answer is as follows. Admissible distance heuristics h admit pruning power in the state space because from $h(s) = \infty$ for a state s, it follows that the real error distance of s is infinity as well. This effectively means that no error trace starts from s, and hence, s can safely be pruned in this case. Again, we observe the close relationship of abstraction-based distance heuristics and abstractions for efficient model checking in general: the abstraction that h^{dprc} is based on is an overapproximation of the original problem, which means that if no error state is reachable in the abstraction, then the system is safe. However, even if the abstraction is not fine enough to entirely prove this (i. e., if the abstraction is not fine enough such that the initial state of the system is evaluated to infinity with the corresponding distance heuristic), the abstraction might nevertheless already provide a distance heuristic with strong pruning power. Apparently, with h^{dprc}, this occurs in the larger E problems, where the initial state is not evaluated to infinity, but most other encountered states actually *are*, and only few states are explored until the system is proved correct. In contrast, more traditional approaches like counter-example-guided abstraction refinement would rely on (at least) one more (possibly costly) refinement step in the abstraction refinement loop in such cases.

Overall, we have observed that admissible distance heuristics are interesting not only because they allow us to find shortest possible error traces in faulty systems, but also because they provide an effective pruning method that allows for efficiently proving correctness without exploring the entire reachable state space. We think that this is an interesting relationship that should be further explored in the future.

8 Conclusions

We have presented downward pattern refinement, a systematic approach to the pattern selection problem in the context of pattern database heuristics for concurrent systems of timed automata. Therefore, we have first provided the necessary theoretical framework for pattern databases and timed automata in general, and a formal basis for measuring the similarity of abstractions in particular. These theoretical results have been put into practice by the downward pattern refinement algorithm, which has shown superior performance compared to other directed model checking approaches as well as UPPAAL on challenging real-time benchmarks of industrial size. In particular, as admissible distance heuristics like h^{dprc} admit pruning power, we have been able to efficiently verify correct systems with directed model checking. This is particularly interesting because directed model checking has originally been proposed to efficiently guide the search to error states, rather than for proving correctness.

While the result of being able to efficiently proving correctness has been rather a side effect so far, it points us to interesting questions for future research. In particular, we observe the close relationship of computing accurate abstractions for verification (i. e., for proving correctness) and for falsification (i. e., for finding error states). As already outlined, a well-established technique to compute abstractions for the former case is counter-example-guided abstraction refinement (CEGAR). CEGAR has already been applied to compute distance heuristics for directed model checking [32] and for domain-independent planning [31]. However, apart from computing accurate distance functions for guiding the search, CEGAR has not yet been investigated for computing distance heuristics with the additional purpose of efficiently proving system correctness. In this context, an important research problem will be to investigate principled approaches for determining to which extent CEGAR should be performed in order to obtain accurate abstractions.

Acknowledgments This work is an extended version of the paper "Abstractions and Pattern Databases: The Quest for Succinctness and Accuracy" [24]. This work was partly supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS) and by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, http://www.avacs.org/).

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT 2004). LNCS, vol. 3185. Springer, Berlin (2004)
3. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Berlin (2004)
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (2000)
5. Culberson, J.C., Schaeffer, J.: Pattern databases. Comput. Intell. **14**(3), 318–334 (1998)
6. Dierks, H.: Comparing model-checking and logical reasoning for real-time systems. Form. Asp. Comput. **16**(2), 104–120 (2004)
7. Dierks, H.: Time, Abstraction and Heuristics—Automatic Verification and Planning of Timed Systems using Abstraction and Heuristics. Habilitation thesis, University of Oldenburg, Germany (2005)
8. Dräger, K., Finkbeiner, B., Podelski, A.: Directed model checking with distance-preserving abstractions. In: Valmari [33], pp. 19–34
9. Dräger, K., Finkbeiner, B., Podelski, A.: Directed model checking with distance-preserving abstractions. Int. J. Softw. Tools Technol. Transf. **11**(1), 27–37 (2009)
10. Edelkamp, S.: Planning with pattern databases. In: Cesta, A., Borrajo, D. (eds.) Proceedings of the 6th European Conference on Planning (ECP 2001). pp. 13–24 (2001)
11. Edelkamp, S., Jabbar, S.: Real-time model checking on secondary storage. In: Edelkamp and Lomuscio [13], pp. 67–83
12. Edelkamp, S., Leue, S., Lluch-Lafuente, A.: Directed explicit-state model checking in the validation of communication protocols. Int. J. Softw. Tools Technol. Transf. **5**(2), 247–267 (2004)
13. Edelkamp, S., Lomuscio, A. (eds.): LNAI, vol. 4428. Springer, Berlin (2007)
14. Edelkamp, S., Schuppan, V., Bosnacki, D., Wijs, A., Fehnker, A., Aljazzar, H.: Survey on directed model checking. In: Peled and Wooldridge [29], pp. 65–89
15. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
16. Hart, P.E., Nilsson, N.J., Raphael, B.: Correction to a formal basis for the heuristic determination of minimum cost paths. SIGART Newsl. **37**, 28–29 (1972)
17. Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S.: Domain-independent construction of pattern database heuristics for cost-optimal planning. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007), pp. 1007–1012. AAAI Press, Menlo Park, California (2007)
18. Helmert, M., Haslum, P., Hoffmann, J.: Flexible abstraction heuristics for optimal sequential planning. In: Boddy, M., Fox, M., Thiébaux, S. (eds.) Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007), pp. 176–183. AAAI Press, Menlo Park, California (2007)
19. Hoffmann, J., Smaus, J.G., Rybalchenko, A., Kupferschmid, S., Podelski, A.: Using predicate abstraction to generate heuristic functions in Uppaal. In: Edelkamp and Lomuscio [13] , pp. 51–66
20. Krieg-Brückner, B., Peleska, J., Olderog, E.R., Baer, A.: The UniForM workbench, a universal development environment for formal methods. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM 1999). LNCS, vol. 1709, pp. 1186–1205. Springer, Berlin (1999)
21. Kupferschmid, S., Dräger, K., Hoffmann, J., Finkbeiner, B., Dierks, H., Podelski, A., Behrmann, G.: Uppaal/DMC—abstraction-based heuristics for directed model checking. In: Grumberg, O., Huth, M. (eds.) Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007). Lecture Notes in Computer Science, vol. 4424, pp. 679–682. Springer, Berlin Heidelberg (2007)
22. Kupferschmid, S., Hoffmann, J., Dierks, H., Behrmann, G.: Adapting an AI planning heuristic for directed model checking. In: Valmari [33], pp. 35–52
23. Kupferschmid, S., Hoffmann, J., Larsen, K.G.: Fast directed model checking via russian doll abstraction. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008). LNCS, vol. 4963. Springer, Berlin (2008)
24. Kupferschmid, S., Wehrle, M.: Abstractions and pattern databases: The quest for succinctness and accuracy. In: Abdulla, P.A., Leino, K.R.M. (eds.) Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011). LNCS, vol. 6605, pp. 276–290. Springer, Berlin (2011)
25. Kupferschmid, S., Wehrle, M., Nebel, B., Podelski, A.: Faster than Uppaal? In: Gupta, A., Malik, S. (eds.) Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008). LNCS, vol. 5123, pp. 552–555. Springer, Berlin (2008)
26. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1–2), 134–152 (1997)
27. Olderog, E.R., Dierks, H.: Moby/RT: A tool for specification and verification of real-time systems. J. Univers. Comput. Sci. **9**(2), 88–105 (2003)
28. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading (1984)
29. Peled, D., Wooldridge, M. (eds.): Proceedings of the 5th International Workshop on Model Checking and Artificial Intelligence (MOCHART 2008), LNAI, vol. 5348. Springer, Berlin (2009)
30. Qian, K., Nymeyer, A.: Guided invariant model checking based on abstraction and symbolic pattern databases. In: Jensen, K., Podelski, A. (eds.) Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004). LNCS, vol. 2988, pp. 497–511. Springer, Berlin (2004)
31. Seipp, J., Helmert, M.: Counterexample-guided Cartesian abstraction refinement. In: Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013). AAAI Press, Menlo Park, California (2013)
32. Smaus, J.G., Hoffmann, J.: Relaxation refinement: A new method to generate heuristic functions. In: Peled and Wooldridge [29], pp. 146–164
33. Valmari, A. (ed.): Proceedings of the 13th International SPIN Workshop (SPIN 2006), LNCS, vol. 3925. Springer, Berlin (2006)
34. Wehrle, M., Helmert, M.: The causal graph revisited for directed model checking. In: Palsberg, J., Su, Z. (eds.) Proceedings of the 16th International Symposium on Static Analysis (SAS 2009). LNCS, vol. 5673, pp. 86–101. Springer, Berlin (2009)

35. Wehrle, M., Kupferschmid, S.: Context-enhanced directed model checking. In: van de Pol, J., Weber, M. (eds.) Proceedings of the 17th International SPIN Workshop (SPIN 2010), LNCS, pp. 88–105. Springer, Berlin (2010)
36. Wehrle, M., Kupferschmid, S.: Mcta: Heuristics and search for timed systems. In: Jurdzinski, M., Nickovic, D. (eds.) Proceedings of the 10th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2012), LNCS, pp. 252–266. Springer, Berlin (2012)
37. Wehrle, M., Kupferschmid, S., Podelski, A.: Transition-based directed model checking. In: Kowalewski, S., Philippou, A. (eds.) Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009). LNCS, vol. 5505, pp. 186–200. Springer, Berlin (2009)