Abstraction-Based Guided Search for Hybrid Systems

Sergiy Bogomolov[1], Alexandre Donzé[2], Goran Frehse[3], Radu Grosu[4],
Taylor T. Johnson[5], Hamed Ladan[1], Andreas Podelski[1], and Martin Wehrle[6]

[1] University of Freiburg, Germany
{bogom,ladanh,podelski}@informatik.uni-freiburg.de
[2] University of California, Berkeley, USA
donze@eecs.berkeley.edu
[3] Université Joseph Fourier Grenoble 1 – Verimag, France
goran.frehse@imag.fr
[4] Vienna University of Technology, Austria
radu.grosu@tuwien.ac.at
[5] University of Illinois at Urbana-Champaign, USA
taylor.johnson@gmail.com
[6] University of Basel, Switzerland
martin.wehrle@unibas.ch

Abstract. Hybrid systems represent an important and powerful formalism for modeling real-world applications such as embedded systems. A verification tool like SpaceEx is based on the exploration of a symbolic search space (the *region space*). As a verification tool, it is typically optimized towards proving the absence of errors. In some settings, e.g., when the verification tool is employed in a feedback-directed design cycle, one would like to have the option to call a version that is optimized towards finding an error path in the region space. A recent approach in this direction is based on *guided search*. Guided search relies on a cost function that indicates which states are promising to be explored, and preferably explores more promising states first. In this paper, we introduce an abstraction-based cost function based on *pattern databases* for guiding the reachability analysis. For this purpose, we introduce a suitable abstraction technique that exploits the flexible granularity of modern reachability analysis algorithms. Our cost function is an effective extension of pattern database approaches that have been successfully applied in other areas. We have implemented our approach in the SpaceEx model checker. Our evaluation shows its practical potential.

1 Introduction

Hybrid systems are extended finite automata whose discrete states correspond to the various modes of continuous dynamics a system may exhibit, and whose transitions express the switching logic between these modes [1]. Hybrid systems have been used to model and to analyze various types of embedded systems [22, 27, 12, 6, 13, 4, 23].

A hybrid system is considered safe if a given set of bad states cannot be reached from the initial states. Hence, reachability analysis is a main concern for hybrid systems. Since the reachability analysis of hybrid systems is in general undecidable [1], modern reachability-analysis tools such as SpaceEx [15] resort to semi-decidable over-approximation techniques [9, 15]. In this paper, we explore the utility of guided search in order to improve the efficiency of such techniques.

Guided search is an approach that has recently found much attention for finding errors in large systems [20, 8]. As suggested by the name, guided search performs a search in the state space of a given system. In contrast to standard search methods like breadth-first or depth-first search, the search is guided by a cost function that estimates the search effort to reach an error state from the current state. This information is exploited by preferably exploring states with lower estimated costs. If accurate cost functions are applied, the search effort can significantly be reduced compared to uninformed search. Obviously, the cost function therefore plays a key role within the setting of guided search, as it should be as accurate as possible on the one hand, and as cheap to compute as possible on the other. Cost functions that have been proposed in the literature are mostly based on *abstractions* of the original system. An important class of abstraction-based cost functions is based on *pattern databases (PDBs)*. PDBs have originally been proposed in the area of Artificial Intelligence [10] and also have successfully been applied to model checking discrete and timed systems [25]. Roughly speaking, a PDB is a data structure that contains abstract states together with abstract cost values based on an abstraction of the original system. During the concrete search, concrete states s are mapped to corresponding abstract states in the PDB, and the corresponding abstract cost values are used to estimate the costs of s. Overall, PDBs have demonstrated to be powerful for finding errors in different formalisms. The open question is if guided search can be applied equally successfully to finding errors in hybrid systems.

A first approach in this direction [8] is to estimate the cost of a symbolic state based on the Euclidean distance from its continuous part to a given set of error states. This approach appears to be best suited for systems which behavior is dominated by the (continuous) differential equations. However, it suffers from the fact that discrete information like mode switches is completely ignored, which can lead to arbitrary degeneration of the search. To see this, consider the example in Fig. 1. It shows a simple hybrid system with one continuous variable which obeys the differential equation $\dot{x} = 1$ in every location (differential equations are omitted in the figure). The error states are given by the locations $l_{e1}, \dots, l_{en}$ and invariants $0 \leq x \leq 8$. In this example, the box-based distance heuristic wrongly explores the whole lower branch first (where no error state is reachable) because it only relies on the continuous information given by the invariants. These invariants suggest that the costs of the "lower" states are equal to 0, whereas the costs of the "upper" states are estimated to be equal to 4 (the distance of the centers of the bounding boxes of the invariants).

In this paper, we introduce a PDB-based cost function for hybrid systems to overcome these limitations. In contrast to the box-based approach based on

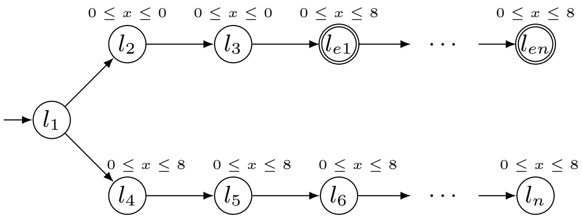

Fig. 1. A motivating example

Euclidean distances, this cost function is also able to properly reflect the discrete part of the system. However, compared to the "classical" discrete setting, the investigation of PDBs for hybrid systems becomes more difficult for several reasons. First, hybrid systems typically feature both discrete and continuous variables with complex dependencies and interactions. Therefore, the question arises how to compute a suitable (accurate) abstraction of the original system. Second, computations for symbolic successors and inclusion checks become more expensive than for discrete or timed systems – can these computations be performed or approximated efficiently to get an overall efficient PDB approach as well? In this paper, we provide answers to these questions, leading to an efficient guided search approach for hybrid systems. In particular, we introduce a technique leveraging properties of the set representations used in modern reachability algorithms. These algorithms typically alternate between using fully symbolic representations such as support functions and explicit representations such as template polyhedrons when required. By simply using much coarser parameters for the explicit representation, we obtain suitable and cheap abstractions for the behaviors of a given hybrid system. Furthermore, we present an approach to reduce the size and computation time of "classical" PDBs by introducing *partial* PDBs. Our implementation in the SpaceEx tool [15] shows the practical potential of our approach.

The remainder of the paper is organized as follows. After introducing the necessary background for this work in Sec. 2, we present our PDB approach for hybrid systems in Sec. 3. This is followed by a discussion about related work in Sec. 4. Afterwards, we present our experimental evaluation in Sec. 5. Finally, we conclude the paper in Sec. 6.

2 Preliminaries

In this section, we introduce the preliminaries that are needed for this work.

2.1 Notations

We consider models that can be represented by hybrid systems. A hybrid system is formally defined as follows.

Definition 1 (Hybrid System) *A* hybrid system *is a tuple* $\mathcal{H} = (\mathit{Loc}, \mathit{Var}, \mathit{Init}, \mathit{Flow}, \mathit{Trans}, \mathit{Inv})$ *defining*

- *the finite set of locations Loc,*
- *the set of continuous variables* $\mathit{Var} = \{x_1, \ldots, x_n\}$ *from* $\mathbb{R}^n$*,*
- *the initial condition, given by the constraint* $\mathit{Init}(\ell) \subset \mathbb{R}^n$ *for each location* ℓ*,*
- *for each location* ℓ*, a relation called* $\mathit{Flow}(\ell)$ *over the variables and their derivatives. We assume* $\mathit{Flow}(\ell)$ *to be of the form*

  $$\dot{x}(t) = Ax(t) + u(t), u(t) \in \mathcal{U},$$

  where $x(t) \in \mathbb{R}^n$*,* A *is a real-valued* $n \times n$ *matrix and* $\mathcal{U} \subseteq \mathbb{R}^n$ *is a closed and bounded convex set,*
- *the discrete transition relation, given by a set Trans of discrete transitions; a discrete transition is formally defined as a tuple* (ℓ, g, ξ, ℓ') *defining*
  - *the source location* ℓ *and the target location* ℓ'*,*
  - *the guard, given by a linear constraint* g*,*
  - *the update, given by an affine mapping* ξ*, and*
- *the invariant* $\mathit{Inv}(\ell) \subset \mathbb{R}^n$ *for each location* ℓ*.*

The semantics of a hybrid system $\mathcal{H}$ is defined as follows. A *state* of $\mathcal{H}$ is a tuple $(\ell, \mathbf{x})$, which consists of a location $\ell \in \mathit{Loc}$ and a point $\mathbf{x} \in \mathbb{R}^n$. More formally, $\mathbf{x}$ is a valuation of the continuous variables in *Var*. For the following definitions, let $\mathcal{T} = [0, \Delta]$ be an interval for some $\Delta \geq 0$. A *trajectory* of $\mathcal{H}$ from state $s = (\ell, \mathbf{x})$ to state $s' = (\ell', \mathbf{x}')$ is defined by a tuple $\rho = (L, \mathbf{X})$, where $L : \mathcal{T} \to \mathit{Loc}$ and $\mathbf{X} : \mathcal{T} \to \mathbb{R}^n$ are functions that define for each time point in $\mathcal{T}$ the location and values of the continuous variables, respectively. Furthermore, we will use the following terminology for a given trajectory ρ. A sequence of time points where location switches happen in ρ is denoted by $(\tau_i)_{i=0\ldots k} \in \mathcal{T}^{k+1}$. In this case, we define the *length* of ρ as $|\tau| = k$. Trajectories $\rho = (L, \mathbf{X})$ (and the corresponding sequence $(\tau_i)_{i=0\ldots k}$) have to satisfy the following conditions:

- $\tau_0 = 0$, $\tau_i < \tau_{i+1}$, and $\tau_k = \Delta$ – the sequence of switching points increases, starts with 0 and ends with Δ
- $L(0) = \ell$, $\mathbf{X}(0) = \mathbf{x}$, $L(\Delta) = \ell'$, $\mathbf{X}(\Delta) = \mathbf{x}'$ – the trajectory starts in $s = (\ell, \mathbf{x})$ and ends in $s' = (\ell', \mathbf{x}')$
- $\forall i \ \forall t \in [\tau_i, \tau_{i+1}) : L(t) = L(\tau_i)$ – the location is not changed during the continuous evolution
- $\forall i \ \forall t \in [\tau_i, \tau_{i+1}) : \ (\mathbf{X}(t), \dot{\mathbf{X}}(t)) \in \mathit{Flow}(L(\tau_i))$, i.e. $\dot{\mathbf{X}}(t) = A\mathbf{X}(t) + u(t)$ holds and thus the continuous evolution is consistent with the differential equations of the corresponding location
- $\forall i \ \forall t \in [\tau_i, \tau_{i+1}) : \ \mathbf{X}(t) \in \mathit{Inv}(L(\tau_i))$ – the continuous evolution is consistent with the corresponding invariants
- $\forall i \ \exists (L(\tau_i), g, \xi, L(\tau_{i+1})) \in \mathit{Trans} : \mathbf{X}_{end}(i) = \lim_{\tau \to \tau_{i+1}^-} \mathbf{X}(\tau) \wedge \mathbf{X}_{end}(i) \in g \wedge \mathbf{X}(\tau_{i+1}) = \xi(\mathbf{X}_{end}(i))$ – every continuous transition is followed by a discrete

one, $\mathbf{X}_{end}(i)$ defines the values of continuous variables right before the discrete transition at the time moment τ_{i+1} whereas $\mathbf{X}_{start}(i) = \mathbf{X}(\tau_i)$ denotes the values of continuous variables right after the switch at the time moment τ_i.

We say that state s' is *reachable* from state s if there exists a trajectory from s to s'.

In the following, we mostly refer to *symbolic states*. A symbolic state $s = (\ell, R)$ is defined as a tuple, where $\ell \in \mathit{Loc}$, and R is a convex and bounded set consisting of points $\mathbf{x} \in \mathbb{R}^n$. The continuous part R of a symbolic state is also called *region*. The symbolic state space of $\mathcal{H}$ is called the *region space*. The initial set of states S_{init} of $\mathcal{H}$ is defined as $\bigcup_{\ell}(\ell, \mathit{Init}(\ell))$. The reachable state space $\mathcal{R}(\mathcal{H})$ of $\mathcal{H}$ is defined as the set of symbolic states that are reachable from an initial state in S_{init}, where the definition of reachability is extended accordingly for symbolic states.

In this paper, we assume there is a given set of symbolic bad states S_{bad} that violate a given property. Our goal is to find a sequence of symbolic states which contains a trajectory from S_{init} to a symbolic *error state*, where a symbolic error state s_e has the property that there is a symbolic bad state in S_{bad} that agrees with s_e on the discrete part, and that has a non-empty intersection with s_e on the continuous part. A trajectory that starts in a symbolic state s and leads to a symbolic error state is called an *error trajectory* $\rho_e(s)$.

2.2 Guided Search

In this section, we introduce a guided search algorithm (Algorithm 1) along the lines of the reachability algorithm used by the current version of SpaceEx [15]. It works on the region space of a given hybrid system. The algorithm checks if a symbolic error state is reachable from a given set of initial symbolic states S_{init}. As outlined above, we define a symbolic state s_e in the region space of $\mathcal{H}$ to be a symbolic error state if there is a symbolic state $s \in S_{bad}$ such that s and s_e agree on their discrete part, and the intersection of the regions of s and s_e is not empty (in other words, the error states are defined with respect to the given set of bad states). Starting with the set of initial symbolic states from S_{init}, the algorithm explores the region space of a given hybrid system by iteratively computing symbolic successor states until an error state is found, no more states remain to be considered, or a (given) maximum number of iterations i_{max} is reached. The exploration of the region space is guided by the *cost* function such that symbolic states with lower cost values are considered first.

In the following, we provide a conceptual description of the algorithm using the following terminology. A symbolic state s' is called a symbolic *successor state* of a symbolic state s if s' is obtained from s by first computing the continuous successor of s, and then by computing a discrete successor state of the resulting (intermediate) state. Therefore, for a given symbolic state s_{curr}, the function continuousSuccessor (line 7) returns the symbolic state which is reachable from s_{curr} within the given time horizon according to the continuous evolution

Algorithm 1 A guided reachability algorithm

```
Input: Set of initial symbolic states $S_{init}$, set of symbolic bad states $S_{bad}$, cost function $cost$
Output: Can a symbolic error state be reached from a symbolic state in $S_{init}$ ?
 1: compute $cost(s)$ for all $s \in S_{init}$
 2: PUSH ($\mathcal{L}_{waiting}, \{(s, cost(s)) \mid s \in S_{init}\}$)
 3: $i := 0$
 4: while ($\mathcal{L}_{waiting} \neq \emptyset \wedge i < i_{max}$) do
 5:   $s_{curr}$ := GETNEXT ($\mathcal{L}_{waiting}$)
 6:   $i := i + 1$
 7:   $s'_{curr}$ := CONTINUOUSSUCCESSOR($s_{curr}$)
 8:   if $s'_{curr}$ is a symbolic error state then
 9:     return "Error state reached"
10:   end if
11:   PUSH ($\mathcal{L}_{passed}, s'_{curr}$)
12:   $S'$ := DISCRETESUCCESSORS($s'_{curr}$)
13:   for all $s' \in S'$ do
14:     if $s' \notin \mathcal{L}_{passed}$ then
15:       compute $cost(s')$
16:       PUSH ($\mathcal{L}_{waiting}, (s', cost(s'))$)
17:     end if
18:   end for
19: end while
20: if $i = i_{max}$ then
21:   return "Maximal number of iterations reached"
22: else
23:   return "Error state not reachable"
24: end if
```

described by the differential equations. Accordingly, the function DISCRETESUCCESSOR (line 12) returns the symbolic state that is reachable due to the outgoing discrete transitions.

A symbolic state s is called *explored* if its symbolic successor states have been computed. A symbolic state s is called *visited* if s has been computed but not yet necessarily explored. To handle encountered states, the algorithm maintains the data structures $\mathcal{L}_{passed}$ and $\mathcal{L}_{waiting}$. $\mathcal{L}_{passed}$ is a list containing symbolic states that are already explored; this list is used to avoid exploring cycles in the region space. $\mathcal{L}_{waiting}$ is a priority queue that contains visited symbolic states together with their cost values that are candidates to be explored next. The algorithm is initialized by computing the cost values for the initial symbolic states and pushing them accordingly into $\mathcal{L}_{waiting}$ (lines $1-2$). The main loop iteratively considers a best symbolic state s_{curr} from $\mathcal{L}_{waiting}$ according to the cost function (line 5), computes its symbolic continuous successor state s'_{curr} (line 7), and checks if s'_{curr} is a symbolic error state (lines $8-10$). (Recall that s'_{curr} is defined as a symbolic error state if there is a symbolic bad state $s \in S_{bad}$ such that s and s'_{curr} agree on their discrete part, and the intersection of the regions of s and s'_{curr} is not empty.) If this is the case, the algorithm terminates. If this is not the case, then s'_{curr} is pushed into $\mathcal{L}_{passed}$ (line 11). Finally, for the resulting symbolic state s'_{curr}, the symbolic discrete successor states are computed, prioritized and pushed into $\mathcal{L}_{waiting}$ if they have not been considered before (lines 12 – 18). Obviously, the search behavior of Algorithm 1 is crucially determined by the cost function that is applied. In the next section, we give a generic description of *pattern database* cost functions.

2.3 General Framework of Pattern Databases

For a given system $\mathcal{S}$, a pattern database (PDB) in the classical sense (i. e., in the sense PDBs have been considered for discrete and timed systems) is represented as a table-like data structure that contains abstract states together with abstract cost values. The PDB is used as a cost estimation function by mapping concrete states s to corresponding abstract states $s^{\#}$ in the PDB, and using the abstract cost value of $s^{\#}$ as an estimation of the cost value of s. The computation of a classical PDB is performed in three steps. First, a subset $\mathcal{P}$ of variables and automata of the original system $\mathcal{S}$ is selected. Such subsets $\mathcal{P}$ are called *pattern*. Second, based on $\mathcal{P}$, an abstraction $\mathcal{S}^{\#}$ is computed that only keeps the variables occurring in $\mathcal{P}$. Third, the entire state space of $\mathcal{S}^{\#}$ is computed and stored in the PDB. More precisely, all reachable abstract states together with their abstract cost values are enumerated and stored. The abstract cost value for an abstract state is defined as the shortest length of a path from that state to an abstract error state. The resulting PDB of these three steps is used as the *cost* function during the execution of Algorithm 1; in other words, the PDB is computed *prior* to the actual model checking process, where the resulting PDB is used as an input for Algorithm 1. In the next section, we will consider this PDB approach as a basis for a cost function for hybrid systems.

3 Pattern Databases for Hybrid Systems

In the last section, we have described the general approach for computing and using a PDB for guiding the search. However, for hybrid systems, there are several problems using the classical PDB approach. First, it is not clear how to effectively compute suitable abstractions for hybrid systems with complex variable dependencies. In Sec. 3.1, we address this problem with an abstraction technique based on varying the granularity of the reachability analysis. Second, in Sec. 3.2, we address the general problem that the precomputation of a PDB is often quite expensive. Moreover, in many cases, only a small fraction of the PDB is actually needed for the search [17]. This is undesirable in general, and specifically becomes problematic in the context of hybrid systems because reachability analysis in hybrid systems is typically much more expensive than, e. g., for discrete systems. In Sec. 3.2, we introduce *partial* PDBs to address these problems.

3.1 Abstractions Based on Coarse-Grained Space Exploration

A general question in the context of PDBs is how to compute suitable abstractions of a given system. For hybrid systems, one could apply one of the abstraction techniques that have been proposed based on simplifying the dynamics [16, 5]. In this paper, we propose a simpler yet elegant way to obtain a coarse grained and fast analysis: For the computation of the PDB, we observe that the LeGuernic-Girard (LGG) algorithm implemented in SpaceEx [15] uses support function representation (based on the chosen set of template directions) to compute and store over-approximations of the reachable states. Therefore, a reduced number of *template directions* and an increased *time step* results in an abstraction of the original region space in the sense that the dependency graph of the reachable abstract symbolic states is a discrete abstraction of the system. The granularity of the resulting abstraction is directly correlated with the parameter selection: Choosing coarser parameters in the reachability algorithm makes this abstraction coarser, whereas finer parameters lead to finer abstractions as well. This is a significant difference compared to the classical approaches that have been proposed in the literature for pattern databases (see Sec. 2.3): Instead of computing a (projection) abstraction based on a *subset* of all variables, we *keep* all variables (and hence, the original system), and instead choose a coarser exploration of the region space.

3.2 Partial Pattern Databases

A classical PDB for a hybrid system $\mathcal{H}$ is represented by a data structure that contains abstract states together with corresponding abstract cost values of a suitable abstraction $\mathcal{H}^\#$ of $\mathcal{H}$ (according to Sec. 3.1). The abstract states and corresponding cost values are obtained by a region space exploration of $\mathcal{H}^\#$. The abstract cost value of an abstract state $s^\#$ is defined as the length of the shortest found trajectory in $\mathcal{H}^\#$ from $s^\#$ to an abstract error state. The PDB computes the cost function

$$cost^P(s) := cost^\#(s^\#),$$

where s is a symbolic state, $s^\#$ is a corresponding abstract state to s in the PDB (see below for a more detailed description of *corresponding abstract state*), and $cost^\#$ is the length of the corresponding trajectory from $s^\#$ to an abstract error state as defined above. In this context, an abstract state $s^\#$ is called a *corresponding* state to s if s and $s^\#$ agree on their discrete part, the symbolic part of s is included in the symbolic part of $s^\#$, and $s^\#$ is an abstract state with minimal abstract costs that satisfies these requirements.

As already outlined, a general drawback of classical PDBs is the fact that their precomputation might become quite expensive. Even worse, in many cases, most of this precomputation time is often unnecessary because only a small fraction of the PDB is actually needed during the symbolic search in the region space [17]. One way that has been proposed in the literature to overcome this problem is to compute the PDB on demand: So-called *switchback search* maintains a family of abstractions with increasing granularity; these abstractions are used to compute the PDB to guide the search in the next-finer level [21]. In the following, we present a more radical way to address this problem in the context of hybrid systems: Instead of computing the whole abstract region space for a given abstraction, we restrict the abstract search to explore only a fraction of the abstract region space while focusing on those abstract states that are likely to be sufficient for the concrete search. This results in a *partially computed* PDB.

Definition 2 (Partial Pattern Database) *Let $\mathcal{H}$ be a hybrid system. A* partial pattern database *for $\mathcal{H}$ is a pattern database for $\mathcal{H}$ that contains only abstract state/cost value pairs for abstract states that are part of some trajectory of shortest length from an initial state to an abstract error state. The partial pattern database computes the function*

$$cost^{PP}(s) := \begin{cases} cost^{\#}(s^{\#}) & \text{if there is corresp. } s^{\#} \text{ to } s \\ +\infty & \text{otherwise} \end{cases}$$

where s, $s^{\#}$, and $cost^{\#}$ are defined as above, and $+\infty$ is a default value indicating that no corresponding abstract state to s exists.

Informally, a partial PDB for a hybrid system $\mathcal{H}$ only contains those abstract states of $\mathcal{H}^{\#}$ that are explored on some *shortest* trajectory (instead of containing *all* abstract states of a complete abstract region space exploration to *all* abstract error states as it would be the case for a classical PDB). In other words, partial PDBs are incomplete in the sense that there might exist concrete states with no corresponding abstract state in the PDB. In such cases, the default value $+\infty$ is returned with the intention that corresponding concrete states are only explored if no other states are available. Obviously, this might worsen the overall search guidance compared to the fully computed PDB. However, in special cases, a partial PDB is sufficient to obtain the same cost function as obtained with the original PDB. At the extreme end of this spectrum, this is the case when only abstract states are excluded from which no abstract error state is reachable anyway. More generally, a partial PDB suffices to deliver the same *search behavior* as the original PDB if at least one abstract error trace is feasible in the original, i. e., in the concrete region space. The search behavior is defined as the sequence of symbolic states the search algorithm explores.

Proposition 1. *Let $\mathcal{H}$ be a hybrid system. If there is a symbolic abstract error state $s_p = (l, \boldsymbol{R})$ in the partial PDB such that there is an error state $s = (l, \mathbf{x})$ with $\mathbf{x} \in \boldsymbol{R}$, where s is reachable in $\mathcal{H}$ from some initial state of $\mathcal{H}$, and the length of a shortest trajectory in $\mathcal{H}$ to reach s is equal to the length of a shortest abstract trajectory to reach s in the partial PDB, then the search behavior of Algorithm 1 with $cost^{PP}$ is equal to the search behavior of Algorithm 1 with $cost^{P}$, i.e., with respect to the fully computed PDB.*

Intuitively, if the preconditions of Prop. 1 are satisfied, then the abstract states in the partial PDB suffice to guide the search in the same way as the fully computed PDB would do (because we never "leave" the partial PDB). If the requirements are not satisfied, we can end up with less accurate cost functions. However, in practice, partial PDBs turn out to be powerful because even if Prop. 1 does not apply, they can often be computed significantly faster than full PDBs, and still contain enough abstract states to accurately guide the search. Overall, although in case the requirements of Prop. 1 are not fulfilled, partial PDBs can still be a good heuristic choice that lead to cost functions that are efficiently computable on the one hand, and that accurately guide the concrete search on the other hand. We will come back to this point in the evaluation section.

3.3 Discussion

Abstraction techniques for verification of hybrid systems have been studied intensively. Our pattern database approach for finding error states is based on a similar idea, but exploits abstractions in a different way than in common approaches for verification. Most notably, the main focus of our abstraction is to provide the basis for the cost function to guide the search, rather than to prove correctness (although, under certain circumstances, it can be efficiently used for verification as well – we will come back to this point in the experiments section). As a short summary of the overall approach, we first compute a symbolic abstract region space (as described in Sec. 3.1), where the encountered symbolic abstract states $s^{\#} = (L^{\#}, R^{\#})$ are stored in a table together with the corresponding abstract cost values of $s^{\#}$. To avoid the (costly) computation of an *entire* PDB, we only compute the PDB partially (as described in Sec. 3.2). This partial PDB is then used as the cost function of our guided reachability algorithm. As in many other approaches that apply abstraction techniques to reason about hybrid systems, the abstraction that is used for the PDB is supposed to accurately reflect the "important" behavior of the system, which results in accurate search guidance of the resulting cost function and hence, of our guided reachability algorithm.

An essential feature of the PDB-based cost function is the ability to reflect the continuous *and* the discrete part of the system. To make this more clear, consider again the motivating example from the introduction (Fig. 1). As we have discussed already, the box-based distance function first wrongly explores the whole lower branch of this system because no discrete information is used to guide the search. In contrast, a partial PDB is also able to reflect the discrete behavior of the system. In this example, the partial PDB consists of an abstract trajectory to the first reachable error state, which is already sufficient to guide the (concrete) region space exploration towards to first reachable error state as well. In particular, this example clearly shows the advantage of partial PDBs compared to fully computed PDBs (recall that fully computed PDBs would include *all* error states, whereas the partial PDB only contains the trajectory to the shortest one). In general, our PDB-approach is well suited for hybrid systems with a non-trivial amount of discrete behavior. However, the continuous behavior is still considered according to our abstraction technique as introduced in Sec. 3.1. Overall, partial PDBs appear to be an accurate approach for guided search because they accurately balance the computation time for the cost function on the one hand, and lead to efficient and still accurately informed cost functions on the other hand.

Finally, let us discuss the relationship of PDBs to counterexample-guided abstraction refinement (CEGAR) [3, 2]. Our approach shares with CEGAR the general idea of using an abstraction to analyze a concrete system. However, in contrast to CEGAR, where abstract counterexamples have to be validated and possibly used in further abstraction refinement, abstractions for PDBs are never refined and only used as a heuristic to *guide* the search within the concrete automaton. In other words, in contrast to CEGAR, the accuracy of the abstraction influences the *order* in which concrete states are explored and therefore the *performance* of the resulting model checking algorithm. Therefore, a crucial difference lies in the fact that CEGAR does the search in the abstract space, replays the counterexample in the concrete space, and stops if the error path cannot be followed. In contrast, our approach does the search in the concrete space and uses the PDBs for guidance, only. If an abstract path cannot be followed, the search does not stop, but tries other branches until either a counterexample is found, or all paths have been exhausted.

4 Related Work

Techniques to efficiently find error states in faulty hybrid systems have recently found increasing attention in the hybrid systems community. Bhatia and Frazzoli [7] propose using rapidly exploring random trees (RRTs). In the context of hybrid systems, the objective of a basic RRTs approach is to efficiently cover the region space in an "equidistant" way in order to avoid getting stuck in some part of the region space. Recently, RRTs were extended by adding guidance of the input stimulus generation [11]. However, in contrast to our approach, RRTs approaches are based on numeric simulations, rather than symbolic executions. Applying PDBs to RRTs would be an interesting direction for future work. In a further approach, Plaku, Kavraki and Vardi [24] propose to combine motion planning with discrete search for falsification of hybrid systems. The discrete search and continuous search components are intertwined in such a way that the discrete search extracts a high-level plan that is then used to guide the motion planning component. In a slightly different setting, Ratschan and Smaus [26] apply search to finding error states in hybrid systems that are deterministic. Hence, the search reduces to the problem of finding an accurate initial state. SpaceEx [15] is a recently developed, yet already prominent model checker for hybrid systems. As suggested by the name, it explores the region space by applying search. The most related approach to this paper has recently been presented by Bogomolov et al. [8], who propose a cost function based on Euclidean distances of the regions of the current state and error states. The resulting guided search algorithm is implemented in SpaceEx and has demonstrated to achieve significant guidance and performance improvements compared to the uninformed search of SpaceEx.

Table 1. Experimental results for the navigation benchmarks. Abbreviations: Uninformed DFS: Uninformed depth-first search, Box-heuristic: box-based distance heuristic, PDB: our PDB cost function $cost^{PP}$, #loc: number of locations, #it: number of iterations, length: length of the found error trajectory, time: total time in seconds including any preprocessing. For our PDB approach, the fraction of the total time that is needed for the PDB computation is additionally reported in parenthesis.

Inst.	#loc	Uninformed DFS			Box-heuristic			PDB		
		#it	length	time	#it	length	time	#it	length	time (time abs.)
1	400	122	15	145.756	62	15	70.548	16	15	**20.04** (1.984)
2	400	183	33	186.93	86	33	120.428	34	33	**53.998** (7.553)
3	625	75	33	70.717	34	33	**36.609**	34	33	44.718 (7.472)
4	625	268	158	261.86	231	158	209.637	159	158	**127.458** (10.458)
5	625	85	79	118.8	26	25	**37.775**	26	25	42.117 (3.728)
6	625	96	53	110.816	101	53	104.938	54	53	**76.296** (9.849)
7	625	227	34	198.95	105	34	96.978	35	34	**47.612** (9.385)
8	625	178	25	266.142	86	25	137.291	26	25	**43.541** (7.09)
9	625	297	17	356.042	102	17	131.965	18	17	**30.789** (7.595)
10	625	440	30	534.041	136	30	201.843	31	30	**60.91** (13.64)
11	900	234	72	269.314	129	21	149.086	22	21	**32.744** (8.107)
12	900	317	43	339.093	174	61	198.326	44	43	**62.829** (15.764)
13	900	367	37	421.902	148	37	190.355	38	37	**70.748** (20.132)
14	900	411	32	434.555	278	32	297.89	33	32	**57.692** (10.934)
15	900	379	44	445.863	107	44	137.757	45	44	**69.912** (9.011)

Moreover, guided search has been intensively and successfully applied to finding error states in a subclass of hybrid systems, namely to *timed* systems. In particular, PDBs have been investigated in this context [19, 20]. In contrast to this paper, the PDB approaches for timed systems are "classical" PDB approaches, i. e., a subset of the available automata and variables are selected to compute a projection abstraction. To select this subset, Kupferschmid et al. [19] compute an abstract error trace and select the automata and variables that occur in transitions in this abstract trace. In contrast, Kupferschmid and Wehrle [20] start with the set of all automata and variables (i. e., with the complete system), and iteratively remove variables as long as the resulting projection abstraction is "precise enough" according to a certain quality measure. In both approaches, the entire PDB is computed, which is more expensive than the partial PDB approach proposed in this paper.

5 Evaluation

We have implemented $cost^{PP}$ in the SpaceEx tool [15] and evaluated it on a number of challenging benchmarks. The implementation and the benchmarks are available at `http://www.informatik.uni-freiburg.de/~bogom/spaceex-pdb`.

Table 2. Experimental results for the satellite benchmarks. Abbreviations: Uninformed DFS: Uninformed depth-first search, Box-heuristic: box-based distance heuristic, PDB: our PDB cost function $cost^{PP}$, #loc: number of locations, #it: number of iterations, length: length of the found error trajectory, time: total time in seconds including any preprocessing, OOM: out of memory. For our PDB approach, the fraction of the total time that is needed for the PDB computation is additionally reported in parenthesis.

Inst.	#loc	Uninformed DFS			Box-heuristic			PDB		
		#it	length	time	#it	length	time	#it	length	time (time abs.)
1	36	116	32	27.112	75	10	13.44	16	10	**10.317** (7.413)
2	36	464	24	101.252	473	13	116.991	30	13	**16.306** (12.24)
3	64	718	87	31.514	278	87	**11.04**	263	121	20.362 (9.543)
4	100	111	107	38.085	44	15	21.073	23	14	**14.802** (6.029)
5	100	109	104	262.944	45	15	178.617	23	14	**62.985** (5.893)
6	159	2170	∞	78.95	1352	∞	49.853	0	∞	**15.587** (15.587)
7	324	323	102	105.589	1289	106	457.702	25	24	**32.102** (8.767)
8	557	1637	42	45.76	936	42	**26.297**	156	42	44.147 (39.674)
9	574	7113	41	223.648	561	10	17.45	14	10	**6.607** (6.224)
10	575	9092	4	284.783	387	5	12.315	15	4	**2.439** (2.032)
11	576	5693	3769	816.596	257	13	36.479	15	13	**9.937** (5.866)
12	576	32966	13	7059.52	826	13	118.947	15	13	**10.012** (5.813)
13	576	n/a	n/a	OOM	579	52	579.738	58	52	**163.206** (82.013)
14	1293	13691	∞	436.164	7719	∞	249.554	0	∞	**135.507** (135.507)
15	1296	n/a	n/a	OOM	1806	142	1869.72	206	139	**617.423** (434.675)

5.1 Benchmarks

We consider benchmark problems with problem spaces with a large discrete part, with a large branching factor and paths with dead-ends where search involves heavy backtracking.

As a first set of benchmarks, we consider a variant of the well-known navigation benchmark [14]. This benchmark models an object moving on the plane which is divided into a grid of cells. The dynamics of the object's planar position in each cell is governed by the differential equations $\dot{x} = v$, $\dot{v} = A(v - v_d)$ where v_d stands for the targeted velocity in this location. Compared to the originally proposed navigation benchmark problem, we address a slightly more complex version with the following additional constraints. First, we add inputs allowing perturbation of object coordinates, i. e., the system of differential equations is extended to: $\dot{x} = v + u$, $\dot{v} = A(v - v_d)$, $u_{min} \leq u \leq u_{max}$. Second, to make the search task even harder, the benchmark problems also feature obstacles between certain grid elements. This is particularly challenging because, in contrast to the original benchmark system, one can get stuck in a cell where no further transitions can be taken, and consequently, backtracking might become necessary. The size of the problem instances varies from 400 to 900 locations, and all instances feature 4 variables.

Second, we consider benchmarks that result from *hybridization*. For a hybrid system $\mathcal{H}$ with nonlinear continuous dynamics, hybridization is a technique for generating a hybridized hybrid automaton from $\mathcal{H}$. The hybridized automaton has simpler continuous dynamics (usually affine or rectangular) that over-approximate the behavior of $\mathcal{H}$ [5], and can be analyzed by SpaceEx. For our evaluation, we consider benchmarks from this hybridization technique applied to nonlinear *satellite orbital dynamics* [18], where two satellites orbit the earth with nonlinear dynamics described by Kepler's laws. The orbits in three-dimensional space lie in a two-dimensional plane and may in general be any conic section, but we assume the orbits are periodic, and hence circular or elliptical. Fixing some orbital parameters (e.g., the orientations of the orbits in three-space), the states of the satellites in three-dimensional space $x_1, x_2 \in \mathbb{R}^3$ can be completely described in terms of their true anomalies (angular positions). Likewise, one can transform between the three-dimensional state description and the angular position state description. The nonlinear dynamics for the angular position are $\dot{\nu}_i = \sqrt{\mu/p_i^3}(1 + e_i \cos \nu_i)^2$ for each satellite $i \in \{1, 2\}$, where μ is a gravitational parameter, $p_i = a_i(1 - e_i^2)$ is the semi-latus rectum of the ellipse, a_i is the length of the semi-major axis of the ellipse, and $0 \leq e_i < 1$ is the eccentricity of the ellipse (if $e_i = 0$, then the orbit is circular and p_i simplifies to the radius of the circle). These dynamics are periodic with a period of 2π, so we consider the bounded subset $[0, 2\pi]^2$ of the state-space $\mathbb{R}^2$, and add invariants and transitions to create a hybrid automaton ensuring $\nu_i \in [0, 2\pi]$. For the benchmark cases evaluated, we fixed $\mu = 1$ and varied p_i and e_i for several scenarios. For more details, we refer to the work of Johnson et al. [18]. The size of the problem instances varies from 36 to 1296 locations, and all instances feature 4 variables.

The verification problem is *conjunction avoidance*, i. e., to determine whether there exists a trajectory where the satellites come too close to one another and may collide. Some of the benchmark instances considered are particularly challenging because they feature several sources of non-determinism, including several initial states and several bad states. As an additional source of nondeterminism, some benchmarks model thrusting. A change in a satellite's orbit is usually accomplished by firing thrusters. This is usually modeled as an instantaneous change in the orbital parameters e_i and a_i. However, the angular position ν_i in this new orbit does not, in general, equal the angular position in the original orbit, and a change of variables is necessary, which can be modeled by a reset of the ν_i values when the thrusters are fired. The transitions introduced for thrusting add additional discrete nondeterminism to the system.

5.2 Experiments

The experiments have been performed on a machine running under Ubuntu 11.10 with a four-core Intel Core i3 2.4GHz processor and 4GB memory. In the following, we report results for our PDB implementation of $cost^{PP}$ in SpaceEx. For the navigation benchmarks, while conducting search in the concrete state space, we use octagonal template directions and sampling time equals to 0.05. In the abstract run, we use box template directions and sampling time equals to 0.5. For different satellite benchmark instances, we used different choices of the directions and sampling times for the concrete and abstract runs, based on

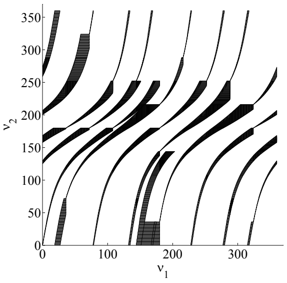

Fig. 2. Uninformed search error trajectory

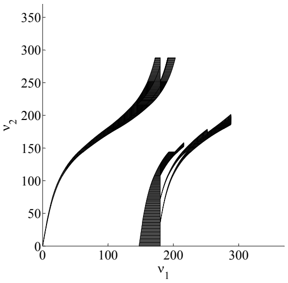

Fig. 3. Box-based heuristic search error trajectory

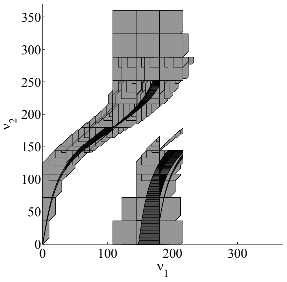

Fig. 4. PDB search error trajectories (abstract: light gray, concrete: dark gray)

the choice of the e_i and p_i parameters in the nonlinear dynamics prior to hybridization, since higher values of e_i result in greater overapproximation error from hybridization. We compared $cost^{PP}$ with uninformed depth-first search as implemented in SpaceEx, and with the recently proposed box-based distance function [8]. We compare the number of iterations of SpaceEx, the length of the error trajectory found as well as the overall search time (including the computation of the PDB for $cost^{PP}$) in seconds. For the PDB approach, we also report the fraction of the total time to compute the PDB in parenthesis. The results are reported in Table 1 and Table 2. Considering the overall run-time, the best results are given in bold fonts.

Our results in Table 1 and Table 2 show that the precomputation time for the PDB mostly pays off in terms of guidance accuracy and overall run-time. Specifically, the overall run-time could (sometimes significantly) be reduced compared to uninformed search and also compared to the box-based heuristic. For example, in satellite instance 5, the precomputation for the PDB only needs around 6 seconds, leading to an overall run-time of around 60 seconds, compared to around 178 seconds with the box-based heuristic and about 263 seconds with uninformed search. This search behavior for instance 5 is also visualized in Fig. 2, Fig. 3, and Fig. 4, where we observe that the part of the covered search space with our PDB approach is lower compared to the box-based heuristic and uninformed search. Fig. 4 particularly shows the part of the search space that is covered by the abstract run (which can be performed efficiently due to our abstraction as described in Sec. 3.1), showing that our PDB approach finds an accurate balance between the computation time and the accuracy of the resulting cost function.

Looking at the results in more detail, we first observe that the number of iterations of SpaceEx and also the length of the found error trajectories are mostly at most as high with PDB as with uninformed search and the box-based heuristic. In particular, our PDB approach could solve instances from the satellite problem where uninformed search ran out of memory. In some cases, the precomputation of the PDB does not pay off compared to the box-based heuristic (recall that the box-based heuristic does not have any precomputation time at all), however, in these cases, the pure search time is still similar to the pure search time of the box-based approach. Second, we observe that the length of the trajectories found by the box-based heuristic and the PDB heuristic is often similar or even equal, while the number of iterations is mostly decreased. This again shows that the search with the PDB approach is more focused than with the box-based heuristic in such cases, and less backtracking is needed. In particular, the box-based heuristic always tries to find a direct path to an error state, while ignoring possible obstacles. Therefore, the search can get stuck in a dead-end state if there is an obstacle, and as a consequence, backtracking becomes necessary. Furthermore, the box-based heuristic can perform worse than the PDB if several bad states are present. In such cases, the box-based heuristic might "switch" between several bad states, whereas the better accuracy of the PDB heuristic better focuses the search towards one particular bad state. In contrast, in problems that are structured more easily (e. g., where no "obstacles" exist and error states are reachable "straight ahead"), the box-based heuristic might yield better performance because the precomputation of the PDB does not pay off.

Finally, we remark that our approach is also able to effectively and efficiently *verify* systems where no bad states exist – this is the case in the satellite instances 6 and 14. In these instances, the *abstract* run (which is supposed to build the PDB) does not reveal any reachable error state. As our abstraction is an over-approximation, we can safely conclude that no reachable error state in the concrete system exists either, and do not need to start the concrete search at all. Being able to efficiently verify hybrid systems with PDBs (that are rather supposed to *guide* the search) is a significant advantage compared to the box-based heuristic.

6 Conclusion

We have explored the application of pattern databases (PDBs) for hybrid systems. For a given safety property and hybrid system with linear dynamics in each location, we compute an abstraction by coarsening the over-approximation SpaceEx computes in its reachability analysis. The abstraction is used to construct a PDB, by associating to each abstract symbolic state the distance in number of transitions to the symbolic error state. This distance is then used in guiding SpaceEx in the concrete search. Given a concrete symbolic state, the guiding heuristics returns the smallest distance to the error state of an enclosing abstract symbolic state. This distance is used to choose the most promising concrete symbolic successor. In our implementation, we have taken advantage of the SpaceEx parametrization support, and were able to report a significant speedup in counterexample detection and even for verification. Our new PDB support for SpaceEx can be seen as a nontrivial extension of our previous work on guided reachability analysis for hybrid systems where the discrete system structure was ignored completely [8]. For the future, it will be interesting to further refine and extend our approach by, e. g., considering even more fine grained abstraction techniques, or by combinations of *several* abstraction techniques and therefore, by combining several PDBs. We expect that this will lead to even more accurate cost functions and better model checking performance.

Acknowledgments

This work was partly supported 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/`) and by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicolin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. R. Alur, T. Dang, and F. Ivancic. Counter-example guided predicate abstraction of hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 250–271, 2003.
3. R. Alur, T. Dang, and F. Ivancic. Progress on reachability analysis of hybrid systems using predicate abstraction. In *Hybrid Systems: Computation and Control*, pages 4–19, 2003.
4. R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON. In *Hybrid Systems: Computation and Control*, pages 6–19, 2000.
5. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.
6. A. Balluchi, L. Benvenuti, M. D. D. Benedetto, C. Pinello, and A. L. Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: challenges and opportunities. *Proceedings of the IEEE*, 88(7):888–912, July 2000.
7. A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, pages 142–156, 2004.
8. S. Bogomolov, G. Frehse, R. Grosu, H. Ladan, A. Podelski, and M. Wehrle. A box-based distance between regions for guiding the reachability analysis of SpaceEx. In *Computer Aided Verification*, pages 479–494, 2012.
9. C. Chutinan and B. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.
10. J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.
11. T. Dang and T. Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.
12. A. Deshpande, D. Godbole, A. Göllü, and P. Varaiya. Design and evaluation of tools for automated highway systems. In *Hybrid Systems III*, pages 138–148, 1996.
13. M. Egerstedt. Behavior-based robotics using hybrid automata. In *Hybrid Systems: Computation and Control*, pages 103–116, 2000.

14. A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control*, pages 381–397, 2004.
15. G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395, 2011.
16. T. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. *Hybrid Systems III*, pages 377–388, 1996.
17. R. C. Holte, J. Grajkowski, and B. Tanner. Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation*, pages 121–133, 2005.
18. T. T. Johnson, J. Green, S. Mitra, R. Dudley, and R. S. Erwin. Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In *Formal Methods*, pages 252–266. 2012.
19. S. Kupferschmid, J. Hoffmann, and K. G. Larsen. Fast directed model checking via russian doll abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 203–217, 2008.
20. S. Kupferschmid and M. Wehrle. Abstractions and pattern databases: The quest for succinctness and accuracy. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 276–290, 2011.
21. B. J. Larsen, E. Burns, W. Ruml, and R. Holte. Searching without a heuristic: Efficient use of abstraction. In *AAAI Conference on Artificial Intelligence*, 2010.
22. C. Livadas, J. Lygeros, and N. A. Lynch. High-level modelling and analysis of tcas. In *IEEE Real-Time Systems Symposium*, pages 115–125, 1999.
23. J. Lygeros, G. J. Pappas, and S. Sastry. An approach to the verification of the center-tracon automation system. In *Hybrid Systems: Computation and Control*, pages 289–304, 1998.
24. E. Plaku, L. Kavraki, and M. Vardi. Hybrid systems: From verification to falsification. In *Computer Aided Verification*, pages 463–476, 2007.
25. K. Qian and A. Nymeyer. Guided invariant model checking based on abstraction and symbolic pattern databases. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 497–511, 2004.
26. S. Ratschan and J.-G. Smaus. Finding errors of hybrid systems by optimising an abstraction-based quality estimate. In *Tests and Proofs*, pages 153–168, 2009.
27. P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Trans. Automatic Control*, 38(2), 1993.