Directed Model Checking for PROMELA with Relaxation-Based Distance Functions

Ahmad Siyar Andisha[1] and Martin Wehrle[2] and Bernd Westphal[3]

[1] corix AG, 4562 Biberist, Switzerland
[2] University of Basel, Switzerland
[3] Albert-Ludwigs-Universität Freiburg, Germany

Abstract. Directed model checking uses distance functions to guide the state space exploration to efficiently find short error paths. Distance functions based on *delete-relaxation* have successfully been used for, e. g., model checking timed automata. However, such distance functions have not been investigated for formalisms with rich expression languages as provided by PROMELA. We present a generalization of delete-relaxation-based distance functions to a subclass of PROMELA. We have evaluated the resulting search behavior on a large number of models from the BEEM database within the HSF-SPIN model checker. Our experiments show significantly better guidance compared to the previously best distance function available in HSF-SPIN.

1 Introduction

A main obstacle for model checking tools is the state space explosion problem. A countermeasure is to use the memory efficient state space traversal procedure depth-first search (DFS). However, if the task is not to verify a property but to falsify it, DFS often performs badly in practice because it may unnecessarily search large error free regions of the state space first. In addition, reported error paths are often unnecessarily long, which makes it difficult for humans to understand the causes of an error. A technique to mitigate these problems is called *directed model checking* (DMC) and has been introduced by Edelkamp et al. [4]. Directed model checking applies a distance function to estimate the distance from a given state to an error state, and explores states with shortest estimated distance first. Guided by the distance function, error paths can often be found after exploring only a small fraction of the overall state space which results in time and memory savings. Furthermore, reported error paths are often shorter than those reported by so-called uninformed algorithms (like DFS) due to the guidance. Typically, there is a trade-off between the precision of the distance estimate and the cost of computing the distance function for a given state. An example for a computationally cheap distance function for PROMELA models is called h^c and is implemented in HSF-SPIN [4], a directed model checker based on version 3 of the PROMELA model checker SPIN. The h^c function estimates the distance between a given state and the end state of the model's never claim. A class of distance functions which is successful in the area of artificial intelligence

(AI) planning is based on *delete relaxation* [2]. Kupferschmid et al. [5] generalized this relaxation to simple statements over variables with arbitrary domains for a limited class of timed automata. In the latter context, the relaxation is based on collecting all values ever assigned to a variable along a path, yielding set-valued domains for variables and the current location. More formally, given a (concrete) state s, $h^+(s)$ denotes the length of the shortest path under the relaxation from s to an error state in the transition system over set-valued variables. $h^+(s)$ is often an accurate estimate of the length of a shortest path between s and an error state in the transition system over concrete states. As the computation of h^+ is NP-hard [3], Kupferschmid et al. considered approximations thereof, which showed favorable performance compared to the distance functions proposed by Edelkamp et al. [4] in the area of timed automata, and which have also found their way into timed automata model checking tools [6]. Although the idea of DMC roots in AI planning and relaxation-based distance functions have shown to be useful outside of AI planning, their potential has not yet been explored for PROMELA, i. e., for a richer expression language than considered until now.

In this work, we explore a generalization of relaxation-based distance functions to the PROMELA formalism. We have evaluated an implementation of our distance function (which generalizes the h^L function proposed by Kupferschmid et al. and is called h^L_P in the following) in HSF-SPIN on a large number of BEEM models. Our implementation supports an expressive subset of PROMELA (cf. [1]), which is sufficiently rich to cover a large range of models from the BEEM database [7]: Currently supported are basic control flow (if, goto), channel synchronisation, static processes, basic data types (no arrays, no structs), and most operators (except for modulo) in expressions. Our results show that h^L_P often provides significantly better guidance towards error states compared to HSF-SPIN's previously best-performing distance function in our experiments.

2 Relaxation for PROMELA

We start by defining *relaxed states* for PROMELA, which are the basis for the definition of our distance function h^L_P. In the PROMELA semantics, a (concrete) state assigns to each process a process location, and to each variable and channel a value. We use, e. g., $s(x)$ to denote the value assigned to variable x in state s. In contrast, a *relaxed* state s^+ assigns to each process a *set* of process locations, and to each variable and channel a *set* of values, i. e. $s^+(x)$ denotes a set of values from the domain of x. We say that relaxed state s^+ *subsumes* a state s, denoted by $s \sqsubseteq s^+$, if and only if each component of s is an element of the corresponding component of s^+, e. g., if for each variable x, $s(x) \in s^+(x)$. PROMELA expressions are *existentially* evaluated over relaxed states: Relaxed state s^+ *supports* value v for expression *expr* if there exists a state s such that $s \sqsubseteq s^+$ and *expr* evaluates to v over s. Note that a relaxed state may hence support both *true* and *false* for a Boolean expression. Statements obtain a collecting semantics on relaxed states. The effect of assignment $x = expr$ on relaxed state

```
int a=0, b=0; chan c=[1] of {int};
active proctype S() { again: c!a; a++; goto again; }
active proctype R() { again: c?b; goto again; }
```

Listing 1.1. Two processes `S` and `R`.

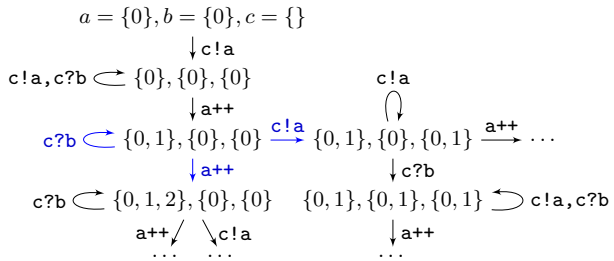

Fig. 1. Relaxed transition system of the PROMELA program shown in Listing 1.1.

s^+ is defined as adding all values of *expr* supported by s^+ to the set $s^+(x)$. Note that we thereby obtain a conservative generalization of [5].

The above interpretation of expressions and statements on relaxed states induces a transition system over relaxed states, called the *relaxed transition system*. It has the property that for each path π in the transition system over concrete states, there exists a path π^+ in the relaxed transition system with the same length such that the i-th relaxed state in π^+ subsumes the i-th state in π. That is, the relaxed transition system is an over-approximation of the concrete transition system: According to the idea of relaxation, the value sets in relaxed successor states grow monotonically. Thus, each statement that can be executed in a state s can also be executed in any relaxed state which subsumes s. An *error path* from relaxed state s^+ is a path which begins with s^+ and ends with a relaxed state which subsumes termination of the model's never claim. Following the literature, we denote the shortest length of an error path from *the* relaxation s^+ of state s by $h^+(s)$. We call a relaxed state s^+ the relaxation of s if and only if s^+ is the smallest relaxed state wrt. set-inclusion which subsumes s.

For an example, consider the PROMELA model in Listing 1.1. Process `S` repeatedly sends the value of variable `a` on channel `c` and increments `a`. Process `R` repeatedly receives a value from `c` and assigns it to variable `b`. In the relaxation, the variables and the channel become set-valued. Figure 1 shows a fragment of the computation tree of the relaxed transition system rooted at relaxed state $\{0\}, \{0\}, \{\}$. Edges are labeled with the executed statements. The relaxed state $\{0\}, \{0\}, \{\}$ is the relaxed state of the concrete initial state in which channel `c` is empty. Thus only the send statement `c!a` is executable. Executing `c!a` yields the relaxed state $\{0\}, \{0\}, \{0\}$. From this relaxed state on, channel `c` is always both empty and full, thus the synchronization in `R` is always enabled. So is the increment of `a`; executing it yields the relaxed state $\{0, 1\}, \{0\}, \{0\}$, i. e. the old value of `a` is not deleted but collected. The example particularly shows that the channels' capacities become unbounded in the relaxation.

Algorithm 1: Computation of h_P^L.

Input : Concrete state s.
Output: Distance estimate $l \in \mathbb{N}_0 \cup \{\infty\}$ to a relaxed error state.

```
1 l ← 0; s_0^+ ← process locations, variables, and channels of s;
2 while never claim not terminated in s_l^+ do
3 |   s^+' ← s_l^+;
4 |   for statement t enabled in s_l^+ do
5 |   |   s^+' ← s^+' ∪ effect of t on s_l^+;
6 |   if s^+' = s_l^+ then
7 |   |   return ∞;
8 |   l ← l + 1; s_l^+ ← s^+';
9 return l;
```

As the computation of $h^+(s)$ is NP-hard [3], we consider the distance function h_P^L, which is an approximation of h^+. The computation of h_P^L is provided in Algorithm 1. While h^+ is defined by the relaxed transition system of a PROMELA program, h_P^L is defined by an *acceleration* of the relaxed transition system: Given a relaxed state s^+, its relaxed successor state $s^{+\prime}$ is the union of the effects of executing all statements enabled in s^+ to s^+ (cf. Algo. 1, Line 4 ff.). Note that a concrete state $s' \sqsubseteq s^{+\prime}$ is not necessarily reachable from any concrete state $s \sqsubseteq s^+$, and if s' is reachable from s then the shortest concrete path may be longer than 1. As an example, consider again Listing 1.1 with the never claim $\neg(a = 2 \ \wedge \ b = 1)$, i. e., error states have the property $a = 2$ and $b = 1$.

- Reaching the error state from s_0 takes at least 6 steps in the concrete system, e. g., witnessed by the statement sequence `c!a`, `a++`, `c?b`, `c!a`, `c?b`, `a++`.
- Shortest error paths in the delete-relaxation (i. e., in the relaxed transition graph in Fig. 1) have length 5, i. e., $h^+(s_0) = 5$, e. g., witnessed by `c!a`, `a++`, `c!a`, `c?b`, `a++`. Compared to the concrete, *several* values can be sent in parallel in the relaxation, hence one fewer receive step for setting the value of b is required (in the above sequence, b receives $\{0, 1\}$ via statement `c?b`).
- The relaxed distance function delivers an estimate of 4, i. e., $h_P^L(s_0) = 4$: After starting with `c!a`, all the statements `c!a`, `c?b` and `a++` are repeatedly applicable in parallel in the following, yielding the sequence of relaxed states $\{0\}, \{0\}, \{\}$ (initial relaxed state), $\{0\}, \{0\}, \{0\}$ after one step, $\{0, 1\}, \{0\}, \{0\}$ after two steps, $\{0, 1, 2\}, \{0\}, \{0, 1\}$ after three steps, and finally the relaxed error state $\{0, 1, 2, 3\}, \{0, 1\}, \{0, 1, 2\}$ after four steps.
- In contrast, h^c is quite sensitive to (the model of) the never claim N. Assuming N consists of two locations and an edge between them guarded by $a = 2 \wedge b = 1$, h^c can only deliver 0 or 1, yielding an uninformed guidance.

Algorithm 1 computes $h_P^L(s)$ by taking as s_0^+ the relaxed state of s and iterating the accelerated transition relation. It always terminates because either a relaxed error state is found (Line 2) or a fixpoint is reached (Line 6). While

an error need not exist, a fixpoint always exists because the state space of a PROMELA model is finite and in the relaxation, states grow monotonically.

3 Evaluation

We have implemented h^L_P in the HSF-SPIN model checker to investigate the following research questions: To which extent can h^L_P improve the guidance of the state space traversal compared to h^c, the best-performing distance function in our experiments previously available in HSF-SPIN? Does the improved guidance pay off in shorter error paths? Ultimately, does the improved guidance pay off in terms of shorter model checking runtime? The latter question addresses the issue of the increased overhead to compute h^L_P in every encountered state: Compared to h^c, the computation of h^L_P naturally becomes more expensive because of the more precise treatment of the (rather expressive) structures handled by PROMELA. In particular, this is the case for the more sophisticated handling of linear arithmetic and the resulting subsumption checks for checking the enabledness of statements and the effects supported by statements. To investigate these questions, we have applied h^L_P on faulty PROMELA models from the BEEM database [7] (12 domains, more than 80 model instances in total), using greedy best first search (GBFS). The models stem from application areas such as mutual exclusion algorithms, controller software, puzzles and communication protocols. For computational efficiency, our implementation of h^L_P additionally applies interval abstraction: If x has been assigned the values a and b with $a < b$, then we keep the whole interval $[a, b]$ instead of $\{a, b\}$ as x's relaxed value.

In order to compactly provide the results, we visualize the data by scatter plots in Fig. 2. All axes are scaled logarithmically. Each cross represents one

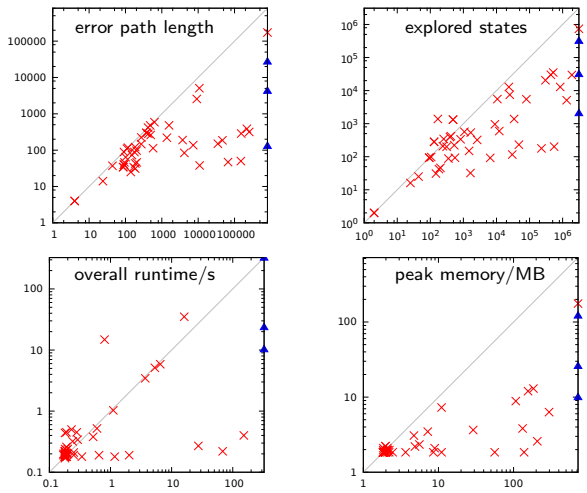

Fig. 2. ×: $x, y \in \mathbb{R}$, i. e., h^c and h^L_P reported a path within the time limit, ▲: $x \in \mathbb{R}$, y failed to find an error. (Athlon 64 2.4 GHz; mem. limit: 3GB, time limit: 30 min.)

successful run of both distance functions for one model instance, i. e., a run which neither violates our memory limit nor times out. A cross below the diagonal line indicates that h^L_P performs better (i. e., shorter path, fewer explored states, less time, less memory) than h^c on this model instance, a cross above this line indicates the opposite. A cross on the diagonal line indicates that both distance functions perform equally. We observe a significantly improved guidance of the state space traversal with h^L_P in terms of the number of explored states, which pays off in terms of a lower memory consumption, and also in reduced lengths of the error paths. In particular, we observe that the improvement is sometimes in the range of several orders of magnitude. The better guidance stems from the more accurate distance values delivered by h^L_P (e. g., in the `bopdp` problem instances, h^L_P's values range from 0 to 39, whereas h^c yields values between 0 and 1). More details are available online [1]. As discussed, the improved guidance naturally comes with an increased overhead to compute h^L_P. However, while the overhead does not always pay off, we also observe that there exist models where h^L_P can remarkably reduce the runtime. In addition, there are models which could be handled by h^L_P, whereas h^c failed to find an error (triangles in Fig. 2).

4 Conclusions

In this paper, we have explored a generalization of delete-relaxed distance functions for PROMELA. Our evaluation in HSF-SPIN on models from the BEEM benchmark suite show a significantly improved guidance compared to HSF-SPIN's previously best-performing distance function in our experiments. While the improved guidance mostly pays off in terms of shorter error paths and lower memory consumption, the benefits in terms of overall runtime are somewhat less significant due to the encountered computational overhead. It will be interesting to further investigate the correlation between this time overhead and the model structure, e. g., the language features used in a given PROMELA model.

Acknowledgments. The authors thank G. J. Holzmann for valuable clarifications of semantical and technical questions on PROMELA and SPIN.

References

1. `www.informatik.uni-freiburg.de/~westphal/spin15`.
2. B. Bonet and H. Geffner. Planning as heuristic search. *AIJ*, 129(1-2):5–33, 2001.
3. T. Bylander. The computational complexity of propositional STRIPS planning. *AIJ*, 69(1-2):165–204, 1994.
4. S. Edelkamp, S. Leue, et al. Directed explicit-state model checking in the validation of communication protocols. *STTT*, 5(2-3):247–267, 2004.
5. S. Kupferschmid, J. Hoffmann, H. Dierks, and G. Behrmann. Adapting an AI planning heuristic for directed model checking. In *SPIN*, pages 35–52, 2006.
6. S. Kupferschmid, M. Wehrle, B. Nebel, and A. Podelski. Faster than Uppaal? In *CAV*, pages 552–555, 2008.
7. R. Pelánek. BEEM: Benchmarks for explicit model checkers. In *SPIN*, pages 263–267, 2007.