Smaller Abstractions for $\forall$CTL* without Next*

Kai Engelhardt[1,2] and Ralf Huuck[2,1]

[1] CSE, UNSW, Sydney, NSW 2052, Australia.
[2] National ICT Australia Ltd. (NICTA), Locked Bag 6016, NSW 1466, Australia.[***]

Abstract. The success of applying model-checking to large systems depends crucially on the choice of good abstractions. In this work we present an approach for constructing abstractions when checking Next-free universal CTL* properties. It is known that functional abstractions are safe and that Next-free universal CTL* is insensitive to finite stuttering. We exploit these results by introducing a safe *Next-free abstraction* that is typically smaller than the usual functional one while at the same time more precise, i.e., it has less spurious counter-examples.

1 Introduction

Model-checking [8, 29] has matured to the perhaps most important industrial-strength formal method for system verification [26, 27, 3, 20, 11, 21].

Only relatively small systems are, however, amenable to standard model-checking. The size of systems is up to exponential in the number of their concurrent components—this is referred to as the *state explosion problem* [32].

This obstacle can often be overcome by working with a suitable abstraction instead of the considered system itself [10, 25, 13, 16, 4, 2, 12, 18]. To a hypothetical user of a model-checker T, an abstraction A is *suitable* for or a system C and a property φ, if it is both, safe and small enough. It is *safe* whenever proving that A satisfies φ established that also C satisfies φ. It is *small enough* if T manages to establish whether A satisfies φ. Heuristics have been proposed to automatically construct suitable abstractions [17, 2]. Not all suitable abstractions are necessarily appropriate for checking φ. Abstraction A might expose spurious counter-examples to φ and prove itself to be too coarse for proper reasoning. One popular approach to deal with spurious counter-examples is called *CEGAR* (for counter-example-guided abstraction refinement). In this approach the abstraction is iteratively refined based on analyses of spurious counter-examples. The process stops when the model checker finds either a proper counter-example or φ to hold. The advantage of CEGAR is that it can still be fully automated [9, 31, 7]. How to find good abstractions and how to improve and refine them has

been an active area of research. The standard approach to refining abstractions is to split abstract states. We suggest a simple method to make the abstraction's transition relation even smaller.

This paper is concerned with a particular class of abstractions only. We call them *functional abstractions* because our abstract systems are the images of concrete systems under some function from concrete to abstract states.[1] Functional abstractions are known to be safe for properties expressed in $\forall\text{CTL}^*$, which encompasses LTL.

In contrast to our approach Ranzato and Tapparo propose to construct strongly preserving abstractions in order to avoid the loss of temporal properties [30] when abstracting. Their construction does not allow the removal of transitions from an abstraction as we propose here. An earlier related approach is presented in [14] where the authors compute property preserving abstractions for CTL^* based on abstract interpretation. Their abstractions are data dependent and do not deal with loops as proposed here. A different approach taken by Kesten and Pnueli in [22] and also in [28] where the abstraction is paired with a progress monitor to enforce certain liveness properties. While this technique preserves full LTL including Next and is more general than our solution, it also orthogonal and more complicated. The two techniques can and should be combined. We present our work in the simplest setting, that is, Kripke structures, because we have no new insights related to explicit liveness constraints such as the justice and compassion sets of the fair discrete systems used by Pnueli et al.

Ball, Kupferman, and Sagiv also tackle the loop abstraction problem for large finite state systems [1]. They avoid Kesten and Pnueli's progress monitors by using *must* transition where possible to ensure progress in the abstraction. In their setting this requires identifying entry and exit ports within the abstract state representing a loop. In contrast to ours, their approach does not exploit the absence of Next in the liveness formula to be verified. Their method also leads to larger abstractions than ours.

The approaches described above all tackle the same problem: abstractions may introduce spurious counter-examples to progress properties whenever a terminating code fragment (e.g. a finite loop) in the concrete system is abstracted to a potentially diverging code fragment (e.g. a self loop). The contribution of this paper is the introduction of a *Next-free abstraction* which is based on a functional abstraction that exhibits such a spurious counter-example. The Next-free abstraction is even smaller than the functional one while at the same time more precise. The key idea is to investigate self-loops on the abstract level. The functional abstraction of a Kripke structure contains a self-loop for an abstract state s if, and only if, there is a transition in the pre-image of s. We propose to omit such a self-loop from the abstraction whenever the pre-image of s does not allow infinite paths. Self-loops are the natural enemy of liveness properties. Thus, by eliminating as many of these self-loops as possible, abstractions become more likely to be useful for checking liveness properties.

---

[1] Functional abstractions are also known as partition refinements or quotient systems [13].

The price to pay is twofold.

1. We lose the next-operator when stating temporal properties. Dropping the next-operator is often considered natural in a refinement setting [24]. Moreover, liveness properties are typically formulated without next.
2. We have to decide for each pre-image of an abstract state with a self-loop whether it allows an infinite path. This is not necessarily feasible. Many concrete systems have conceptually infinite state spaces. When abstracting to finite state spaces most pre-images of abstract states tend to be infinite themselves. Depth-bounded search could be used to approximate an answer to the decision problem. If that search is inconclusive it is always safe to keep the self-loop.

We show that Next-free abstractions are sound for the Next-free universal fragment $\forall\mathrm{CTL}^*_{\bigcirc}$ of CTL^*, but generally unsound for formulae containing $\bigcirc$.

The remainder of this work is organized as follows: In Section 2 we introduce basic notations for model-checking universal CTL^* and for abstractions. The subsequent Section 3 introduces our novel abstractions for Next-free $\forall\mathrm{CTL}^*$. We prove that these abstractions are sound, and generally smaller and more precise than the standard abstraction. We give an example of this in Section 4 before drawing final conclusions and pointing out to future work in Section 5.

2   Model-Checking for $\forall\mathrm{CTL}^*_{\bigcirc}$

To increase the accessibility of the paper we repeat some of the standard definitions for a.o. Kripke structures, paths, execution sequences, (minimal) functional abstractions, and the syntax and semantics of $\forall\mathrm{CTL}^*$. Readers familiar with these notions can skip to Section 3.

2.1   Kripke Structures and Abstractions

Let P be a set of *atomic propositions*. A *Kripke structure (over P)* [23] is characterized by a tuple (S, S_0, R, μ) such that: S is a set of *states*; $S_0 \subseteq S$ is a set of *initial states*; $R \subseteq S \times S$ is a *transition relation*, which is required to be total, i.e., for every state $s \in S$ there exists an $s' \in S$ such that $(s, s') \in R$; and $\mu : S \longrightarrow 2^P$ is a *labeling function* which assigns a set of propositions to every state.

Let $K = (S, S_0, R, \mu)$ be a Kripke structure. Let I be a non-void and possibly infinite segment of $\mathbb{N}$. An I-indexed state sequence $s = (s_i)_{i \in I} \in S^I$ is a *path of K* if, for all $i \in I$ such that also $i + 1 \in I$ we have that $(s_i, s_{i+1}) \in R$. Whenever I is infinite we say that s is a *full path (of K)*. If the first element of a path s is an initial state, we call s an *initial path (of K)*. Full initial paths are usually called *execution sequences*. The *property $\mathcal{O}_K$* of K is the set of all its execution sequences. Let $S' \subseteq S$. The set of all full paths in K containing only states of S' is denoted by $\mathcal{O}_K(S')$. For a path $\pi = (s_i)_{i \in \mathbb{N}}$, we write π^i for the suffix of π starting at s_i.

Given two sets of states S and S'. We call $h \subseteq S \times S'$ an *abstraction relation* iff it is the graph of a total function onto S'.

Let $h \subseteq S \times S'$ be an abstraction relation. Let $K' = (S', S'_0, R', \mu')$ be another Kripke structure. Say that K' is an *abstraction (of K, with respect to h)* if $h(S_0) \subseteq S'_0$ (concrete initial states are mapped to abstract initial states), $h^{-1}; R; h \subseteq R'$ (concrete transitions are mapped to abstract transitions), and (propositions are preserved) $\mu'(S') = \bigcup \mu(h^{-1}(S'))$.

We say that h is *faithful*, if, for all $s' \in S'$ and $s_1, s_2 \in h^{-1}(s')$ we have $\mu(s_1) = \mu(s_2)$. From now on, we shall only consider faithful abstraction relations.

The smallest such abstraction, $(S', h(S_0), h^{-1}; R; h, \bigcup \mu(h^{-1}(S^h)))$, is called the *minimal abstraction (of K with respect to h)* and referred to by K^h. Its components are referred to by S^h, S^h_0, R^h, and μ^h, respectively.

2.2 $\forall$CTL*

Syntax. Next we define the logic $\forall$CTL* (over P). The syntax of *state* and *path formulas* is given in BNF by:

state formulas: $\forall$CTL$^* \ni \varphi ::= p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \forall \Phi$
path formulas: $\Phi ::= \varphi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \bigcirc \Phi \mid \Phi \, \mathcal{U} \, \Phi \mid \Phi \, \mathcal{V} \, \Phi$

where $p \in P$.

The slightly uncommon *release* operator $\mathcal{V}$ denotes the dual of the until operator $\mathcal{U}$, i.e., it expresses the CTL* formula $\neg(\neg\varphi \, \mathcal{U} \, \neg\psi)$. Boolean constants (*true* $= p \vee \neg p$, *false* $= p \wedge \neg p$ for some $p \in P$) and further common temporal modalities (e.g., $\Diamond \Phi = true \, \mathcal{U} \, \Phi$, $\Box \Phi = \Phi \, \mathcal{V} \, false$) are defined as abbreviations (cf. also [14, 13]).

Semantics. The semantics of $\forall$CTL* is given by inductive definitions of satisfaction relations $\models$ for state and path formulas over Kripke structures.

Consider a Kripke structure $K = (S, S_0, R, \mu)$, a state $s \in S$, a proposition $p \in P$, a path $\pi \in \mathcal{O}_K$, state formulas φ and ψ, and path formulas Φ and Ψ.

- $K, s \models p$ iff $p \in \mu(s)$;
- $K, s \models \neg p$ iff $p \notin \mu(s)$;
- $K, s \models \varphi \vee \psi$ iff $K, s \models \varphi$ or $K, s \models \psi$;
- $K, s \models \varphi \wedge \psi$ iff $K, s \models \varphi$ and $K, s \models \psi$;
- $K, s \models \forall \Phi$ iff for all paths $\pi \in \mathcal{O}_K$ beginning with s, we have that $K, \pi \models \Phi$;
- $K, \pi \models \varphi$ iff $K, \pi_0 \models \varphi$;
- $K, \pi \models \Phi \vee \Psi$ iff $K, \pi \models \Phi$ or $K, \pi \models \Psi$;
- $K, \pi \models \Phi \wedge \Psi$ iff $K, \pi \models \Phi$ and $K, \pi \models \Psi$;
- $K, \pi \models \bigcirc \Phi$ iff $K, \pi^1 \models \Phi$;
- $K, \pi \models \Phi \, \mathcal{U} \, \Psi$ iff there exists $k \in \mathbb{N}$ such that $K, \pi^k \models \Psi$ and $K, \pi^j \models \Phi$ for all $0 \le j < k$;
- $K, \pi \models \Phi \, \mathcal{V} \, \Psi$ iff for all $k \in \mathbb{N}$, if for all $0 \le j < k$ we have $K, \pi^j \not\models \Phi$ then $K, \pi^k \models \Psi$.

If Q is a set Q of states or paths we write $K, Q \models \varphi$ to abbreviate $\forall q \in Q(K, q \models \varphi)$. We abbreviate $K, S_0 \models \varphi$ to $K \models \varphi$.

Recall that we use $\forall$CTL$^*_\bigcirc$ to denote the Next-free fragment of $\forall$CTL*. Among others, Browne et al. showed that minimal abstractions are safe for $\forall$CTL*, i.e., every $\forall$CTL* formula, which holds for a minimal abstraction does so for the original system [6, 19, 14]. Dams' thesis contains a thorough exposition of abstraction techniques for CTL* and related logics [13].

Theorem 1 ([6]). *Let K be a Kripke structure, let h be a faithful abstraction relation, and let φ be a formula of $\forall CTL^*$. Then $K^h \models \varphi$ implies $K \models \varphi$.*

Let π be a full path. A *stutter step* is a pair of consecutive equal states $\pi_i = \pi_{i+1}$. Call π *stutter-free* either if it has no stutter steps (i.e., $\forall i \in \mathbb{N} \, (\pi_i \neq \pi_{i+1})$) or if the only stutter steps are infinite consecutive repetitions of a single state, i.e., $\exists k \in \mathbb{N} \, (\forall i < k \, (\pi_i \neq \pi_{i+1}) \wedge \forall i > k \, (\pi_i = \pi_k))$. We denote by $\natural\pi$ the *stutter-free equivalent of π*, which is the stutter-free path obtained from π by contracting each maximal finite sequence of stutter steps to a single state. We call two paths π and π' *stutter equivalent* if $\natural\pi = \natural\pi'$ and denote this by $\pi \equiv_\natural \pi'$. Denote the $\natural$-equivalence class of π by $[\pi]_{\equiv_\natural}$.

De Nicola and Vaandrager showed that CTL$^*_\bigcirc$, the Next-free fragment of CTL* is insensitive to stuttering [15]. Thus, so is $\forall$CTL$^*_\bigcirc$.

Theorem 2. *Let K be a Kripke structure, let Φ be a path formula of $\forall CTL^*_\bigcirc$, and let $\pi, \pi' \in \mathcal{O}_K$ be stutter equivalent. Then $K, \pi \models \Phi$ iff $K, \pi' \models \Phi$.*

3   Minimal Abstractions for $\forall$CTL$^*_\bigcirc$

Definition 1 (Next-free abstraction). *Let $K = (S, S_0, R, \mu)$ be a Kripke structure and $h \subseteq S \times S^h$ be a faithful abstraction relation. The Kripke structure $K^h_\bigcirc = (S^h, S^h_0, R^h_\bigcirc, \mu^h)$ given by*

1. *$S^h_0 = h(S_0)$,*
2. *$R^h_\bigcirc = (h^{-1}; R; h) \setminus \{ (s, s) \mid \mathcal{O}_K(h^{-1}(s)) = \emptyset \}$, and*
3. *$\mu^h(S^h) = \bigcup \mu(h^{-1}(S^h))$*

is called K's Next-free abstraction with respect to h.

Observe that this is well-defined because $R^h_\bigcirc$ is necessarily total. Self-loops (stutter steps) in R are preserved since they give rise to full paths.

Next we investigate how Next-free abstractions relate to stuttering [24].

Lemma 1. *Let $K = (S, S_0, R, \mu)$ be a Kripke structure, S^h a non-empty set, and $h \subseteq S \times S^h$ an abstraction relation. Then, $h(\mathcal{O}_K) \subseteq [\mathcal{O}_{K^h_\bigcirc}]_{\equiv_\natural}$.*

Proof. Let $\pi = (s_i)_{i \in \mathbb{N}} \in \mathcal{O}_K$. We need to show that $h(\pi) \in [\mathcal{O}_{K^h_\bigcirc}]_{\equiv_\natural}$.

Observe that, by the definition of S^h_0, we have that $h(s_0)$ is an initial state of the Next-free abstraction. It remains to be shown that if the image $(h(s_i), h(s_{i+1}))$

of a step (s_i, s_{i+1}) in π is missing from $R^h_{\bigcirc}$, then (a) the missing step must be a stutter step, and (b) that the length of this stuttering is finite. In other words, for all $i \in \mathbb{N}$, should $(h(s_i), h(s_{i+1})) \notin R^h_{\bigcirc}$, then $h(s_i) = h(s_{i+1})$ and there exists $k > i$ such that $h(s_k) \neq h(s_i)$.

For (a), observe that $(h(s_i), h(s_{i+1})) \notin R^h_{\bigcirc}$ implies that $(h(s_i), h(s_{i+1})) \in R^h \setminus R^h_{\bigcirc}$. Thus, by the definition of $R^h_{\bigcirc}$ and R^h, it follows that $h(s_i) = h(s_{i+1})$.

For (b), let $i \in \mathbb{N}$ such that $h(s_i) = h(s_k)$ for all $k > i$. Note that the full path π^i is in $\mathcal{O}_K(h^{-1}(h(s_i)))$. Consequently, $(h(s_i), h(s_i)) \in R^h_{\bigcirc}$ by the definition of $R^h_{\bigcirc}$.   □

Theorem 3. *Let $K = (S, S_0, R, \mu)$ be a Kripke structure, S^h a non-empty set, $h \subseteq S \times S^h$ a faithful abstraction relation, and $\varphi \in \forall CTL^*_{\bigcirc}$. Then $K^h_{\bigcirc} \models \varphi$ implies $K \models \varphi$.*

Proof. We show by simultaneous induction over the structure of state and path formulas that $K^h_{\bigcirc}, s \models \varphi \Rightarrow K, h^{-1}(s) \models \varphi$ for all state formulas φ and that $K^h_{\bigcirc}, \pi \models \Phi \Rightarrow K, h^{-1}(\pi) \models \Phi$ for all path formulas Φ. We only show the interesting cases.

Case state formula φ is a proposition p:

$$
\begin{aligned}
K^h_{\bigcirc}, s \models p &\Rightarrow p \in \mu^h(s) \\
&\Rightarrow p \in \bigcup_{s' \in h^{-1}(s)} \mu(s') \\
&\Rightarrow \forall s' \in h^{-1}(s)\,(p \in \mu(s'))\quad \text{, as } h \text{ is faithful} \\
&\Rightarrow \forall s' \in h^{-1}(s)\,(K, s' \models p) \\
&\Rightarrow K, h^{-1}(s) \models p
\end{aligned}
$$

Case φ is $\forall \Phi$:

$$
\begin{aligned}
K^h_{\bigcirc}, s \models \forall \Phi &\Rightarrow \forall \pi \in \mathcal{O}_{K^h_{\bigcirc}} \left(\pi_0 = s \Rightarrow K^h_{\bigcirc}, \pi \models \Phi \right) \\
&\Rightarrow \forall \pi \in [\mathcal{O}_{K^h_{\bigcirc}}]_{\equiv_{\natural}} \left(\pi_0 = s \Rightarrow K^h_{\bigcirc}, \pi \models \Phi \right)\quad \text{, by Theorem 2} \\
&\Rightarrow \forall \pi \in [\mathcal{O}_{K^h_{\bigcirc}}]_{\equiv_{\natural}} \left(\pi_0 = s \Rightarrow K, h^{-1}(\pi) \models \Phi \right)\quad \text{, ind. hyp.} \\
&\Rightarrow \forall \pi \in h(\mathcal{O}_K) \left(\pi_0 = s \Rightarrow K, h^{-1}(\pi) \models \Phi \right)\quad \text{, by Lemma 1} \\
&\Rightarrow \forall \pi' \in \mathcal{O}_K \left(h(\pi'_0) = s \Rightarrow K, \pi' \models \Phi \right) \\
&\Rightarrow K, h^{-1}(s) \models \forall \Phi \qquad\qquad\qquad\qquad\qquad\qquad\qquad □
\end{aligned}
$$

4   Example

To illustrate that Next-free abstractions can indeed generate smaller and more precise abstractions, we use a simple traffic light controller example taken from [9]. Consider a traffic light as depicted in Fig. 1(a). There are three states: red, yellow,
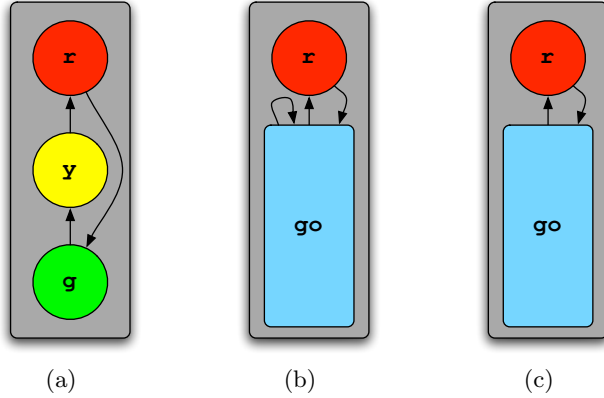
Fig. 1. Traffic light controller.

and green ($\{r, y, g\}$), where r is the initial state. We consider them to be labeled with the propositions go and stop as follows: $\mu(r) = $ stop and $\mu(g) = \mu(y) = $ go.

We want to prove the liveness property that the traffic light is infinitely often red, i.e., $\varphi = \forall \Box \forall \Diamond$ stop. We do admit that no model-checker struggles with this example. For the sake of argument let us, however, compare the minimal abstraction with the Next-free abstraction for the abstraction relation $h = \{(r, r), (g, go), (y, go)\}$.

The minimal abstraction is shown in Fig. 1(b). Note that φ no longer holds, since the self-loop introduced by contracting the two states g and y allows control to remain in go forever. This counter-example is, of course, spurious. On the other hand, the Next-free abstraction shown in Fig. 1(c) eliminates the self-loop, because there is no infinite path in $h^{-1}(go)$. This means, φ is still valid and there is no spurious counter-example. Moreover, the transition relation is smaller than the one of the minimal abstraction. Most known techniques of automatic abstraction refinement based on spurious counter-example detection amount to splitting the go state, leading to the original, concrete system. Kesten and Pnueli's construction would result in the parallel composition of 1(b) with a progress monitor. Ball, Kupferman, and Sagiv would identify g as entry port and y as exit port of go in Fig. 1(b) and conclude that the self-loop on go in that figure cannot diverge.

The additional effort for Next-free abstraction lies in checking the pre-image of abstractions for infinite paths. If the size of the pre-images allows, this can be done efficiently by checking it for self-loops and non-trivial strongly connected components. Otherwise, depth-bounded search can be used to approximate. The same idea as in the original CEGAR approach can be adapted to keep increasing the search depth in the pre-image of the abstract state with the self-loop responsible for the spurious counter-example.

5   Conclusion and Future Work

Tailoring abstractions according to the property to check is not a new idea. We suggest in this paper to look at effectively computable improvements of simple abstractions. We have illustrated this idea by one particular improvement, namely, the omission of certain self-loops in abstractions. This omission is shown to be sound for $\forall\text{CTL}^*_\bigcirc$, the Next-free fragment of $\forall\text{CTL}^*$.

The proposed elimination of self-loops appears to be optimal in the sense that generalizing it to other loops does not promise any gain over existing abstraction refinement techniques such as the ones proposed by Clarke et al. and by Saïdi [9, 31]. In the worst case, to remove a transition closing a loop of length n, introduce a state component to the abstraction—in other words split abstract states—to memorize the last $n-1$ states visited.

One future task is to implement the proposed abstraction technique in existing tools and to compare its performance to that of its precursors. Our attempts have so far been frustrated by the perceived lack of an open source CEGAR framework that actually works. The much-needed comparison should give an indication for the potential speed-up with respect to the usual functional abstraction as well as the relative number of abstractions that do not need further refinement.

Acknowledgment

The authors would like to thank Ed Clarke for his talk on abstractions and counter-examples given in Sydney in July 2003 during which the core idea of this paper was born. We would also like to thank the anonymous referees who saw an earlier version of this paper. One remark is in order: we do consider $\mu(y) = \text{go}$ correct because we never stop at yellow.

References

1. T. Ball, O. Kupferman, and M. Sagiv. Leaping loops in the presence of abstraction. In W. Damm and H. Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *LNCS*, pages 491–503. Springer-Verlag, 2007.
2. T. Ball, T. Millstein, and S. K. Rajamani. Polymorphic predicate abstraction. *ACM Transactions on Programming Languages and Systems*, 27(2):314–343, 2005.
3. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995.
4. S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems automatically and compositionally. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of the 10th International Conference On Computer Aided Verification*, volume 1427 of *LNCS*, pages 319–331. Springer-Verlag, 1998.
5. E. Brinksma and K. G. Larsen, editors. *Computer Aided Verification*, volume 2404 of *LNCS*. Springer-Verlag, July 27–31 2002.

6. M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

7. S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in C. In *International Conference on Software Engineering*, pages 385–395, May 2003.

8. E. M. Clarke and E. A. Emerson. Synthesis of synchronisation skeletons for branching time temporal logic. In D. Kozen, editor, *Workshop on Logic of Programs*, volume 131 of *LNCS*. Springer-Verlag, 1981.

9. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.

10. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*, pages 343–354, 1992.

11. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

12. P. Cousot. On abstraction in software verification. In Brinksma and Larsen [5], pages 37–56.

13. D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Technical University of Eindhoven, July 1996.

14. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, Mar. 1997.

15. R. de Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.

16. J. Dingel and T. Filkorn. Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939 of *LNCS*, pages 54–69. Springer-Verlag, 1995.

17. M. B. Dwyer, J. Hatcliff, R. Joehanes, S. Laubach, C. S. Pasareanu, Robby, W. Visser, and H. Zheng. Tool-supported program abstraction for finite-state verification. In *Proceedings of the 23rd International Conference on Software Engineering*, May 2001.

18. P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In Brinksma and Larsen [5], pages 137–150.

19. J. F. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M. Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, volume 443 of *LNCS*, pages 626–638. Springer-Verlag, 1990.

20. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

21. G. J. Holzmann. *The SPIN Model Checker*. Pearson Educational, 2003.

22. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation*, 163(1):203–243, 2000.

23. S. A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

24. L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress*, pages 657–668, Paris, France, 19–23 Sept. 1983. North-Holland.

25. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.
26. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
27. A. Olivero and S. Yovine. *KRONOS: A Tool for Verifying Real-Time Systems. User's Guide and Reference Manual.* VERIMAG, Grenoble, France, 1993.
28. A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, infty)-counter abstraction. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2002.
29. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Symposium on Programming*, volume 137 of *LNCS*, pages 337–351, Torino, Italy, 6–8 Apr. 1982. Springer-Verlag.
30. F. Ranzato and F. Tapparo. Generalized strong preservation by abstract interpretation. *Journal of Logic and Computation*, 17(1):157–197, 2007.
31. H. Saïdi. Model checking guided abstraction and analysis. In J. Palsberg, editor, *Seventh International Static Analysis Symposium (SAS'00)*, volume 1824 of *LNCS*, pages 377–339. Springer-Verlag, July 2000.
32. A. Valmari. The state explosion problem. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 429–528. Springer-Verlag, 1998.