Acta Informatica (2020) 57:671–687
https://doi.org/10.1007/s00236-020-00376-5

ORIGINAL ARTICLE

Characteristic invariants in Hennessy–Milner logic

Marc Jasper[1] · Maximilian Schlüter[1] · Bernhard Steffen[1]

Received: 15 July 2019 / Accepted: 21 February 2020 / Published online: 6 May 2020

Abstract
In this paper, we prove that Hennessy–Milner Logic (HML), despite its structural limitations, is sufficiently expressive to specify an initial property φ_0 and a characteristic invariant χ_I for an arbitrary finite-state process P such that $\varphi_0 \wedge \mathbf{AG}(\chi_I)$ is a characteristic formula for P. This means that a process Q, even if infinite state, is bisimulation equivalent to P iff $Q \models \varphi_0 \wedge \mathbf{AG}(\chi_I)$. It follows, in particular, that it is sufficient to check an HML formula for each state of a finite-state process to verify that it is bisimulation equivalent to P. In addition, more complex systems such as context-free processes can be checked for bisimulation equivalence with P using corresponding model checking algorithms. Our characteristic invariant is based on so called class-distinguishing formulas that identify bisimulation equivalence classes in P and which are expressed in HML. We extend Kanellakis and Smolka's partition refinement algorithm for bisimulation checking in order to generate concise class-distinguishing formulas for finite-state processes.

1 Introduction

Branching time semantics [34,35] and, in particular, variants of bisimulation [32,33,35,37,38,40] together with their congruence properties [12,36,37] are topics that Rob van Glabbeek has at heart. This also comprises probabilistic behavior [8], a topic that we (Rob and Bernhard) cooperated on almost thirty years ago [39]. Hennessy–Milner logic (HML) [17] can be considered the most basic modal logic for capturing branching time semantics. It is therefore not surprising that Rob investigated its congruence properties [10,11]. The famous theorem of Hennessy and Milner [17] establishes the link between (strong) bisimulation and HML: two finitely branching processes can be separated by means of an HML formula if and only if they are not bisimulation equivalent. This is often stated as HML and bisimulation having the same distinguishing power. Characteristic formulas have a more ambitious role than separating two processes, they are meant to characterize entire bisimulation equivalence

✉ Marc Jasper
marc.jasper@cs.tu-dortmund.de

Maximilian Schlüter
maximilian.schlueter@cs.tu-dortmund.de

Bernhard Steffen
steffen@cs.tu-dortmund.de

1 TU Dortmund University, Dortmund, Germany

classes: χ is a (bisimulation) characteristic formula for a process P if satisfaction of χ coincides with being bisimulation equivalent to P, i.e.:

$$\forall Q.\ (Q \models \chi \iff Q \sim_{\text{bisim}} P)$$

In [2], Browne et al. showed how characteristic formulas can be constructed for finite-state processes in computational tree logic (CTL).[1] Moreover, in [28], it was shown how such formulas can be generated within the modal μ-calculus (and HML with recursion, respectively).

In this paper, we prove that HML is sufficient to define an initial property φ_0 and a characteristic invariant χ_I such that $\chi_P =_{def} \varphi_0 \wedge \mathbf{AG}(\chi_I)$ is a characteristic formula for a given finite-state process P. This means, in particular, that it is sufficient to check an HML formula for each reachable state of a finite-state process to verify that it is bisimulation equivalent to P. In fact, using e.g. the model checking algorithms presented in [5–7], context-free processes, pushdown-processes, and sequential processes can be checked for bisimulation equivalence with P.

Key to the construction of χ_P is the separation of the specification of the one-step transition potential of states of P from their (loose) characterization. This separation essentially decomposes the global bisimulation property into a number of local-step properties in a way reminiscent of Floyd's inductive assertion method [9]: the global property is guaranteed via the consistency—here given by the local transition potential—of (loose) invariants. The latter are given by distinguishing formulas for the bisimulation equivalence classes of P.

Whereas the one-step transition potential can be specified in the way proposed in [2], the class-distinguishing formulas that separate non-bisimilar states can be constructed along the classical partition refinement process for minimization up to bisimulation [21]. Key is that the constructed set of distinguishing formulas Φ for P has the following two properties:

– Each state of P satisfies some distinguishing formula.
– The partition of reachable states in P that is induced by Φ defines a bisimulation relation.

Throughout this paper, we only compare processes from a given universe that is specified by an alphabet Σ. This alphabet Σ comprises all considered process actions (transition labels). Based on Σ and the before-mentioned set of distinguishing formulas Φ, the desired HML invariant χ_I for specifying the one-step potential of P can be derived as in [2]:

$$\chi_I =_{def} \bigwedge_{\varphi \in \Phi} \left(\varphi \implies \left(\bigwedge_{\substack{p \xrightarrow{a} p',\, \psi \in \Phi \\ p \models \varphi,\ p' \models \psi}} \langle a \rangle\, \psi \ \wedge \ \bigwedge_{a \in \Sigma} [a] \left(\bigvee_{\substack{p \xrightarrow{a} p',\, \psi \in \Phi \\ p \models \varphi,\ p' \models \psi}} \psi \right) \right) \right)$$

The resulting characteristic formula $\varphi_0 \wedge \mathbf{AG}(\chi_I)$ conceptually decomposes the verification process in a 'Floyd-like' manner using invariants and class-distinguishing formulas. In contrast, the characteristic formulas presented in [28] can rather be considered to be a monolithic, syntactic encoding of P.

After sketching some preliminaries in Sect. 2, Sect. 3 introduces our characteristic HML invariants based on class-distinguishing formulas and proves a corresponding characterization theorem. Subsequently, Sect. 4 presents the concept of a set of class-distinguishing formulas (SCDF) as a means to establish a sufficient condition for guaranteeing that our characteristic formula construction results in a formula that is satisfied by the argument LTS.

[1] Please note that it is not required that Q is finite state.

The effective construction of SCDFs from LTSs is described in Sect. 5. The underlying algorithm is an adaption of Kanellakis and Smolka's algorithm for checking bisimulation equivalence [21] to generate the required class-distinguishing formulas. Following a discussion of related work in Sect. 6, the paper closes with our conclusions and an outlook in Sect. 7.

2 Preliminaries

In this section, we recall the basic concepts that our approach is based on. Central in this regard are the definition of labeled transition systems and the usual corresponding notion of a process.

Definition 1 (*LTS and process*) A *labeled transition system* (LTS) is a triple $L = (S_L, \Sigma_L, \rightarrow_L)$ with set S_L of states, alphabet Σ_L,[2] and transition relation $\rightarrow_L \subseteq S_L \times \Sigma_L \times S_L$.

We use the notation $p \xrightarrow{a}_L q$ to denote that $(p, a, q) \in \rightarrow_L$ and omit the subscript L in cases where the LTS is unambiguous. Furthermore, given a state $s' \in S_L$ and a label $a \in \Sigma_L$, we define the following shorthand notation for the set of a-predecessors of state s'

$$a^{-1}s' \;=_{def}\; \{s \in S_L \mid s \xrightarrow{a} s'\}$$

This definition extends naturally to sets of states: for each $S' \subseteq S_L$, we have

$$a^{-1}S' =_{def} \bigcup_{s' \in S'} a^{-1}s'$$

Every state $s \in S_L$ of an LTS L defines a *process* $P_L(s)$ that constraints initial transitions to start in s. A state $s' \in S_L$ is *reachable* in $P_L(s)$, denoted by $s \rightarrow_L^* s'$, iff there exists a sequence of transitions $s_i \xrightarrow{a_i}_L s_{i+1}$ in $\rightarrow_L$ with $i \in 0 \mathrel{..} (k-1)$ for some $k \in \mathbb{N}$ such that $s_0 = s$ and $s_k = s'$. This notion naturally generalizes to a notion of reachability within an LTS L from any state s of L.

We are aiming at characterizing processes up to bisimulation, a semantic equivalence relation that is known to preserve properties expressed in most temporal logics, in particular those that can be expressed in the modal μ-calculus [1].

Definition 2 (*Bisimulation*) Let $L = (S_L, \Sigma_L, \rightarrow_L)$ be an LTS. A symmetric relation $R \subseteq (S_L \times S_L)$ is called a *bisimulation* if the following holds for all $(p, q) \in R$:

$$\forall (p \xrightarrow{a} p') \in \rightarrow_L .\ \exists (q \xrightarrow{a} q') \in \rightarrow_L .\ (p', q') \in R$$

Two states $s, s' \in S$ are called bisimilar in L, written as $s \sim_L s'$, iff there exists a bisimulation R with $(s, s') \in R$.[3]
Given a second LTS $L' = (S_{L'}, \Sigma_{L'}, \rightarrow_{L'})$, two processes $P_L(s)$ and $P_{L'}(s')$ are called bisimilar, written as $P_L(s) \sim P_{L'}(s')$, iff there exists a bisimulation in $L'' = (S_L \uplus S_{L'}, \Sigma_L \cup \Sigma_{L'}, \rightarrow_L \uplus \rightarrow_{L'})$ that contains (s, s').[4]

[2] Note that $\Sigma_L \subseteq \Sigma$ holds for any LTS L, because—as stated in the introduction—we only consider LTSs/processes from a given universe specified by Σ.

[3] Note that $\sim$, which is in fact the union of all bisimulation relations, is itself a bisimulation.

[4] The operator $\uplus$ stands for the disjoint union of two sets.

Our characteristic invariants are formulas specified in Hennessy–Milner logic (HML) [17] that extends propositional logic with a modal operator $\langle\cdot\rangle$ (diamond):

Definition 3 (*HML*) Given an alphabet Σ, Hennessy–Milner logic (HML) is defined by the following grammar in Backus–Naur form

$$\varphi ::= tt \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \langle\alpha\rangle\,\varphi$$

where α is a meta-variable for an arbitrary element of Σ. In addition, the following derived operators are frequently used:

$$\begin{array}{ll} ff =_{def} \neg tt & \varphi_1 \vee \varphi_2 =_{def} \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ \varphi_1 \implies \varphi_2 =_{def} \neg\varphi_1 \vee \varphi_2 & [\alpha]\,\varphi =_{def} \neg\langle\alpha\rangle\,\neg\varphi \end{array}$$

Definition 4 (*Semantics of HML*) The semantics of HML are defined relative to a given LTS $L = (S, \Sigma_L, \rightarrow)$ and formalized via the following satisfaction relation $\models_L \subseteq S \times \text{HML}$:

$$\begin{array}{lll} s \models_L tt & & \\ s \models_L \neg\varphi & \text{iff} & s \not\models_L \varphi \\ s \models_L \varphi_1 \wedge \varphi_2 & \text{iff} & s \models_L \varphi_1 \text{ and } s \models_L \varphi_2 \\ s \models_L \langle a\rangle\,\varphi & \text{iff} & \exists s' \in S.\ s \xrightarrow{a} s' \text{ and } s' \models_L \varphi \end{array}$$

where $s \in S, a \in \Sigma$ and $\varphi, \varphi_1, \varphi_2 \in$ HML. We omit the subscript L of $\models_L$ if it is unambiguous in a given context, and often abbreviate $p \models_L \varphi$ and $q \models_L \varphi$ by $p, q \models_L \varphi$. In cases were we want to emphasize the process perspective, we write $P_L(s) \models \varphi$ instead of $s \models_L \varphi$.

In the following, we will establish that HML is sufficient to define the invariants required for our characteristic formulas.

3 Generalized characteristic HML invariant

Our definition of a *generalized characteristic HML invariant* follows the "diamond-box-pattern" originally introduced in [13,15]:

Definition 5 (*Generalized characteristic HML invariant*) Let Σ be a global alphabet that contains all considered action labels. With a pair (Φ, Ψ) such that Φ is a finite set of HML formulas and $\Psi : \Phi \times \Sigma \rightarrow 2^{\Phi}$ a function we associate an HML formula $\chi_I(\Phi, \Psi)$ called *generalized characteristic invariant (GCI)* as follows:

$$\chi_I(\Phi, \Psi) =_{def} \bigwedge_{\varphi\in\Phi} \left(\varphi \implies \left(\bigwedge_{\substack{a\in\Sigma,\\ \psi\in\Psi(\varphi,a)}} \langle a\rangle\,\psi \;\wedge\; \bigwedge_{a\in\Sigma} [a] \left(\bigvee_{\psi\in\Psi(\varphi,a)} \psi \right) \right) \right)$$

Given an LTS $L = (S, \Sigma_L, \rightarrow)$, we write $L \models \mathbf{AG}(\varphi)$ iff $s \models_L \varphi$ holds for all $s \in S$ and say that $\chi_I(\Phi, \Psi)$ is a GCI for L iff $L \models \mathbf{AG}(\chi_I(\Phi, \Psi))$. We sometimes write χ_I instead of $\chi_I(\Phi, \Psi)$ iff the arguments are unambiguous in a given context.

In order to capture our notion of a GCI, we extend HML as follows.

Definition 6 (*Syntax of* HML$_{\text{AG}}$) Given an alphabet Σ, Hennessy–Milner logic with **AG** (HML$_{\text{AG}}$) is defined by the following grammar in Backus–Naur form

$$\varphi ::= tt \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \langle\alpha\rangle\,\varphi \mid \mathbf{AG}(\varphi)$$

where α is a meta-variable for an arbitrary element of Σ. The derived operators of HML are also used for $\mathrm{HML}_{\mathrm{AG}}$.

$\mathrm{HML}_{\mathrm{AG}}$ is equivalent to a logic called EF or UB^{-} in the literature [25] and is itself a fragment of CTL. In this paper, we are only interested in two patterns of $\mathrm{HML}_{\mathrm{AG}}$ formulas, namely $\mathbf{AG}(\chi_I)$ and $\varphi_0 \wedge \mathbf{AG}(\chi_I)$ where $\varphi_0, \chi_I \in \mathrm{HML}$. Definition 6 therefore specifies **AG** instead of an **EF** operator.

Definition 7 (*Semantics of* HML_{AG}) The semantics of $\mathrm{HML}_{\mathrm{AG}}$ are defined relative to a given LTS $L = (S, \Sigma_L, \rightarrow)$. Given an LTS L, the satisfaction relation $\models_L \subseteq S \times \mathrm{HML}_{\mathrm{AG}}$ extends Definition 4 with the following clause:

$$s \models_L \mathbf{AG}(\varphi) \quad \text{iff} \quad \forall s' \in S.\, s \rightarrow^*_L s' \ \text{implies} \ s' \models_L \varphi$$

where $s \in S$ and $\varphi \in \mathrm{HML}_{\mathrm{AG}}$. Shorthand notations are defined as in Definition 4.

Note the overloading of **AG**. In fact, we have:

$$\begin{aligned} L \models \mathbf{AG}(\varphi) \ &\text{iff} \ \forall s \in S.\, s \models_L \mathbf{AG}(\varphi) \\ &\text{iff} \ \forall s \in S.\, P_L(s) \models \mathbf{AG}(\varphi) \end{aligned}$$

Please recall that we only consider LTSs and processes from a universe that is specified by an alphabet Σ, i.e. the set of possible actions. The following lemma illustrates the power of the "diamond-box-pattern" [13,15] that underlies our definition of GCIs.

Lemma 1 *Let* $\chi_I(\Phi, \Psi)$ *be a GCI and* $L = (S, \Sigma_L, \rightarrow)$ *an LTS such that* $L \models \mathbf{AG}(\chi_I)$. *Then*

$$R = \{(p, q) \in S \times S \mid \exists \varphi \in \Phi.\, p, q \models_L \varphi\}$$

is a bisimulation.

Proof Let $\chi_I(\Phi, \Psi)$ be a GCI that is based on some finite set Φ of HML formulas and some function $\Psi \colon \Phi \times \Sigma \rightarrow 2^{\Phi}$. Furthermore, let L be any LTS such that $L \models \mathbf{AG}(\chi_I(\Phi, \Psi))$. As the empty relation is clearly a bisimulation, we can assume that R is not empty. Let us now consider an arbitrary pair $(p, q) \in R$. Then there exists a formula $\varphi \in \Phi$ with $p, q \models \varphi$. Therefore, we know due to $L \models \mathbf{AG}(\chi_I(\Phi, \Psi))$ that both p and q satisfy

$$\bigwedge_{\substack{a\in\Sigma,\\ \psi\in\Psi(\varphi,a)}} \langle a\rangle\, \psi \quad \wedge \quad \bigwedge_{a\in\Sigma} [a] \left(\bigvee_{\psi\in\Psi(\varphi,a)} \psi \right).$$

R is obviously symmetric. Therefore, it suffices to prove the defining invariance property of bisimulation (Definition 2). Let $p \xrightarrow{b} p'$ be an arbitrary transition starting in p. Looking at the subformula

$$[b] \left(\bigvee_{\psi\in\Psi(\varphi,b)} \psi \right)$$

that must be satisfied by p, we know that $p' \models \bigvee_{\psi\in\Psi(\varphi,b)} \psi$. Therefore, there exists a $\psi' \in \Psi(\varphi, b)$ that is satisfied by p'. The fact that q satisfies the subformula

$$\bigwedge_{a\in\Sigma,\psi\in\Psi(\varphi,a)} \langle a\rangle\, \psi$$

now yields that q satisfies $\langle b \rangle\, \psi'$, i.e., the existence of a transition $q \overset{b}{\to} q'$ such that q' satisfies ψ'. Thus, both p' and q' satisfy ψ' which guarantees $(p', q') \in R$. □

Of course, not every GCI fits each LTS L, e.g., the bisimulation guaranteed by the above characterization lemma may well be empty. GCIs develop their full characterizing power only in combination with an initial condition:

Lemma 2 *Let $\chi_I(\Phi, \Psi)$ be a GCI and $\varphi_0 \in \Phi$. Moreover, let $L = (S_L, \Sigma_L, \to_L)$, $L' = (S_{L'}, \Sigma_{L'}, \to_{L'})$ be two LTSs with states $s \in S_L$ and $s' \in S_{L'}$ such that $s \models_L \varphi_0$ and $s' \models_{L'} \varphi_0$, respectively. Then we have:*

1. *$L \models \boldsymbol{AG}(\chi_I)$ and $L' \models \boldsymbol{AG}(\chi_I)$ implies $P_L(s) \sim P_{L'}(s')$.*
2. *If all states of S_L are reachable from s in L and all states of $S_{L'}$ are reachable from s' in L', then $P_L(s) \sim P_{L'}(s')$ implies $L \models \boldsymbol{AG}(\chi_I)$ and $L' \models \boldsymbol{AG}(\chi_I)$.*

Proof Let χ_I be a GCI and L and L' two LTS with states $s \in S_L$ and $s' \in S_{L'}$ such that $s \models_L \varphi_0$ and $s' \models_{L'} \varphi_0$, respectively.

In order to prove the first part of Lemma 2, we assume that $L \models \mathbf{AG}(\chi_I)$ and $L' \models \mathbf{AG}(\chi_I)$ hold, and observe that

$$L'' = (S_L \uplus S_{L'}, \Sigma_L \cup \Sigma_{L'}, \to_L \uplus \to_{L'})$$

is a well-defined LTS which also satisfies $\mathbf{AG}(\chi_I)$ and in which both s and s' satisfy φ_0, i.e., $s, s' \models_{L''} \varphi_0$. Thus, Lemma 1 guarantees that

$$R = \{(p, q) \in (S_L \uplus S_{L'}) \times (S_L \uplus S_{L'}) \mid \exists \varphi \in \Phi.\ p, q \models_{L''} \varphi\}$$

is a bisimulation. As $s, s' \models_{L''} \varphi_0$ also implies $(s, s') \in R$, we can conclude that $P_L(s)$ and $P_{L'}(s')$ are bisimilar as desired.
Our semantics of **AG** for an entire LTS coincides with the (standard) semantics for processes in which all states are reachable. Thus, the proof of the second part is a consequence of the well-known fact that bisimulation preserves e.g. all CTL and modal μ-calculus formulas. □

The following theorem is a straightforward reformulation of Lemma 2 considering processes as models of the GCI instead of LTSs:

Theorem 1 (GCI-based characteristic formulas) *Let χ_I be a GCI based on a finite set Φ of HML formulas, $\varphi_0 \in \Phi$, and $L = (S_L, \Sigma_L, \to_L)$, $L' = (S_{L'}, \Sigma_{L'}, \to_{L'})$ be two LTSs with states $s \in S_L$ and $s' \in S_{L'}$. Then we have:*

$$(P_L(s) \models \varphi_0 \wedge \boldsymbol{AG}(\chi_I) \text{ and } P_{L'}(s') \models \varphi_0 \wedge \boldsymbol{AG}(\chi_I)) \quad \textit{iff} \quad P_L(s) \sim P_{L'}(s')$$

Whereas the implication from left to right is just a reformulation of Lemma 2(1), the converse implication exploits the fact that satisfaction of invariance properties of processes only concerns the reachable states.

The following section establishes a sufficient condition for sets of formulas Φ to serve as a basis for the definition of non-trivial GCIs for a finite-state LTS, i.e., GCIs that remain valid when initial conditions $\varphi \in \Phi$ are added, resulting in characteristic formulas for processes as shown in Theorem 1.

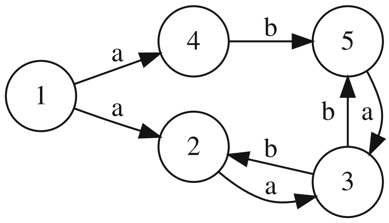

Fig. 1 Nondeterministic LTS

4 Class-distinguishing formulas

Our notion of class-distinguishing formulas is directly based on the concept of bisimulation:

Definition 8 (*Set of class-distinguishing formulas* (*SCDF*)) Given any LTS $L = (S, \Sigma_L, \rightarrow)$, a finite set $\Phi \subseteq$ HML with $\forall s \in S.\ \exists \varphi \in \Phi.\ s \models \varphi$ is called *set of class-distinguishing formulas* (SCDF) for L iff

$$R = \{(p, q) \in S \times S \mid \exists \varphi \in \Phi.\ p, q \models \varphi\}$$

is a bisimulation.

Example 1 (*Set of class-distinguishing formulas* (*SCDF*)) Consider the LTS L depicted in Fig. 1. The history tree in Fig. 2 identifies the set $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$ with

$$\varphi_1 = \langle a\rangle \langle a\rangle\, tt$$
$$\varphi_2 = \langle a\rangle\, tt \wedge [a]\,[a] ff$$
$$\varphi_3 = [a] ff$$

as an SCDF for L. The subscript i in φ_i correlates with the corresponding state identifier s_i such that $s_i \models \varphi_i$. Note that state 2 is bisimilar to state 5 and that state 3 is bisimilar to state 4.

Being HML formulas, the elements of an SCDF Φ for L can, in general, *not* fully characterize the behavior of states in L. They are only sufficient to identify bisimulation equivalence classes in the context of L. In the following, we will see how the "diamond-box-pattern" turns this property into a property that universally characterizes processes up to bisimulation, even in the context of infinite-state systems.

Definition 9 (*SCDF-based GCIs*) Let $L = (S, \Sigma_L, \rightarrow)$ be an LTS, Φ an SCDF for L, and $\Psi : \Phi \times \Sigma \rightarrow 2^{\Phi}$ defined by

$$\Psi(\varphi, a) = \{\psi \in \Phi \mid \exists p, q \in S.\ p \xrightarrow{a} q\ \wedge\ p \models \varphi\ \wedge\ q \models \psi\}.$$

Then $\chi_I(\Phi, \Psi)$ is called a *GCI for L based on* Φ.

A GCI χ_I based on an SCDF for L is sufficient to guarantee that $L \models \mathbf{AG}(\chi_I)$.

Lemma 3 *Let $L = (S, \Sigma_L, \rightarrow)$ be any LTS, Φ an SCDF for L, and χ_I a GCI for L based on Φ. Then we have $L \models \mathbf{AG}(\chi_I)$.*

Proof Let $L = (S, \Sigma_L, \rightarrow)$ be an LTS, Φ an SCDF for L, and $\chi_I(\Phi, \Psi)$ a GCI for L based on Φ. Then we have to prove that L satisfies the following formula:

$$\mathbf{AG}\left(\bigwedge_{\varphi\in\Phi}\left(\varphi \implies \left(\bigwedge_{\substack{a\in\Sigma,\\ \psi\in\Psi(\varphi,a)}} \langle a\rangle\,\psi \;\wedge\; \bigwedge_{a\in\Sigma}[a]\left(\bigvee_{\psi\in\Psi(\varphi,a)}\psi\right)\right)\right)\right)$$

where

$$\Psi(\varphi, a) = \{\psi \in \Phi \mid \exists p, q \in S.\ p \xrightarrow{a} q \;\wedge\; p \models \varphi \;\wedge\; q \models \psi\}.$$

According to the semantics of $L \models \mathbf{AG}(\varphi)$ for some $\varphi \in$ HML, it suffices to show that every state $s \in S$ satisfies the conjunction $\chi_I(\Phi, \Psi)$ that serves as the argument of the **AG** operator. Let $s \in S$ and $\varphi \in \Phi$ be arbitrary but fixed elements. We can assume that $s \models \varphi$, because otherwise the implication is trivially satisfied. Thus, it remains to be shown that

$$s \models \bigwedge_{\substack{a\in\Sigma,\\ \psi\in\Psi(\varphi,a)}} \langle a\rangle\,\psi \;\wedge\; \bigwedge_{a\in\Sigma}[a]\left(\bigvee_{\psi\in\Psi(\varphi,a)}\psi\right)$$

holds. Therefore, let φ_c be any of the conjuncts in that HML formula. In order to show its validity, we distinguish between two cases:

Case 1: $\varphi_c = \langle a\rangle\,\psi$ for some $a \in \Sigma$ with $\psi \in \Psi(\varphi, a)$.
Because of $\psi \in \Psi(\varphi, a)$, there exist states $p, q \in S$ and a transition $p \xrightarrow{a} q$ with $p \models \varphi$ and $q \models \psi$. Thus, we have $p \models \varphi_c$. In addition, $s, p \models \varphi$ implies $s \sim_L p$ according to the definition of an SCDF. As bisimilar states satisfy the same HML formulas [17], this yields $s \models \varphi_c$, which closes the first case.

Case 2: $\varphi_c = [a]\,\delta$ with $\delta = \bigvee_{\psi\in\Psi(\varphi,a)}\psi$ for some $a \in \Sigma$.
In this case, the following has to be shown:

$$\forall s' \in S.\ s \xrightarrow{a} s' \text{ implies } s' \models \delta$$

Therefore, let $s' \in S$ be any state such that $s \xrightarrow{a} s'$ and $\psi \in \Phi$ be a corresponding formula with $s' \models \psi$. Such a formula exists due to the definition of an SCDF. Now, $s \models \varphi$ implies $\psi \in \Psi(\varphi, a)$ by definition of Ψ which guarantees that ψ is a disjunct in δ and therefore that $s' \models \delta$ as desired. □

The following theorem follows straightforwardly from Lemma 3 and the definition of SCDFs.

Theorem 2 (Sufficiency for characterization) *Let $L = (S, \Sigma_L, \rightarrow)$ be any LTS with $s_0 \in S$, Φ an SCDF for L, and χ_I a GCI for L based on Φ. Then there exists a $\varphi_0 \in \Phi$ such that $s_0 \models \varphi_0$ and we have:*

$$P_L(s_0) \;\models\; \varphi_0 \wedge \boldsymbol{AG}(\chi_I)$$

The presented construction of a GCI for an LTS L depends on a corresponding SCDF (Definition 8). The next section introduces an approach that allows us to elegantly generate these sets of class-distinguishing formulas for any finite-state LTS.

5 Generation of characteristic invariants for finite-state LTSs

In this section, we present an algorithm that automatically generates a finite set of class-distinguishing formulas (SCDF) for any finite-state LTS L. Given this SCDF, a GCI for L can be generated according to Sect. 4.

After a brief sketch of partition refinement [16,19] and so called splitters, Sect. 5.2 introduces the *class-distinguishing functions* that we use to generate an SCDF. Afterwards, Sect. 5.3 presents our algorithm together with an accompanying example.

5.1 Partition refinement

Partition refinement serves as the underlying concept of many analysis and verification techniques [31].

Definition 10 (*Partition refinement*) Given a set S, $P \subseteq 2^S$ is a *partition* of S iff its elements, called classes, are non-empty, pairwise disjoint, and cover S when merged ($\bigcup_{X \in P} X = S$). Given two partitions P and Q of S, P *refines* Q, denoted by $P \preceq Q$, iff for each $X \in P$ there exists a $Y \in Q$ such that $X \subseteq Y$. We write $P \prec Q$ iff $P \preceq Q$ and $P \neq Q$.

Our algorithm that is presented in Sect. 5.3 relies on partition refinement: it extends the algorithm by Kanellakis and Smolka [21] for the minimization of non-deterministic systems up to bisimulation. Our extension computes HML formulas that allow to identify the individual classes of the partitions that arise during the refinement algorithm.

Partition refinement within this algorithm is based on witnesses, called splitters, which prove that one or more classes contain states that are not bisimilar.

Definition 11 (*Splitter*) Let $L = (S, \Sigma_L, \rightarrow)$ be an LTS and P a partition over S. Let $B, Y \in P$ and $a \in \Sigma_L$. Then the pair (B, a) is a *splitter* of Y iff $Y \cap a^{-1}B \neq \emptyset$ and $Y \backslash a^{-1}B \neq \emptyset$ (see Definition 1). We denote this by $(B, a) \mid Y$. Furthermore, we define the abbreviations $Y_B^a =_{def} Y \cap a^{-1}B$ and $Y_B^{\not a} =_{def} Y \backslash a^{-1}B$.

Note that (B, a) might be a splitter for multiple classes in P. A refinement based on (B, a) splits all those classes:

Definition 12 (*Splitter-based refinement*) Let P be a partition and (B, a) a splitter of some class $Y \in P$. Let $P' = \{Y \in P \mid (B, a) \text{ is a splitter of } Y\}$. Then the partition

$$P_B^a =_{def} (P \backslash P') \cup \{Y_B^a \mid Y \in P'\} \cup \{Y_B^{\not a} \mid Y \in P'\}$$

is called the *refinement of* P *based on* (B, a).

Kanellakis and Smolka [21] proved that exhaustive splitting while starting with the trivial partition inevitably results in the coarsest partition that defines a bisimulation. The next section shows how we obtain formulas that uniquely identify the classes of this partition by the corresponding satisfaction relation, a property that makes the set of those formulas an SCDF.

5.2 Class-distinguishing functions

The SCDFs that we construct depend on the concrete chain of refinements produced by the (typically non-deterministic) partition refinement algorithm. Thus, let us consider an arbitrary but fixed scenario in this subsection, i.e.:

Let $L = (S, \Sigma_L, \rightarrow)$ be a finite-state LTS, $P_0 = \{S\}$, $(B_0, a_0), \ldots, (B_{m-1}, a_{m-1})$ a sequence of m splitters, and $P_0, \ldots, P_m$ the corresponding sequence of $m + 1$ (refined) partitions such that for all $k \in 0 .. m - 1$, we have

1. (B_k, a_k) is a splitter of some $Y \in P_k$,
2. P_{k+1} is the refinement of P_k based on (B_k, a_k), and
3. P_m cannot be refined based on splitting.

This allows for the following inductive definition:

Definition 13 (*Class-distinguishing functions*) The sequence $\varphi_0, \ldots, \varphi_m$ with $\varphi_k \colon P_k \rightarrow \text{HML}$ for all $k \in 0 .. m$ inductively defined by $\varphi_0(S) = tt$ and

$$\varphi_{k+1}(X) =_{def} \begin{cases} \varphi_k(Y) \wedge \langle a_k \rangle\, \varphi_k(B_k) & \text{if } \exists Y \in P_k.\ (B_k, a_k) \mid Y \text{ and } X = Y^{a_k}_{B_k} \\ \varphi_k(Y) \wedge \neg\langle a_k \rangle\, \varphi_k(B_k) & \text{if } \exists Y \in P_k.\ (B_k, a_k) \mid Y \text{ and } X = Y^{\not a_k}_{B_k} \\ \varphi_k(X) & \text{otherwise.} \end{cases}$$

is called *sequence of class-distinguishing functions*.

Note that the three cases in the definition of $\varphi_{k+1}(X)$ are disjoint and that Y is unique in the first two cases. For this sequence of class-distinguishing functions, which is uniquely determined by the considered scenario, we have:

Theorem 3 (Class-distinguishing functions)

$$\forall k \in 0 .. m.\ \forall X \in P_k.\ \forall s \in S.\ \ s \in X \textit{ iff } s \models \varphi_k(X)$$

Proof We prove the validity of the statement

$$A(k) =_{def} \forall X \in P_k.\ \forall s \in S.\ \ s \in X \text{ iff } s \models \varphi_k(X)$$

by induction over $k \in 0 .. m$.

Base case ($k = 0$): Initially, we have $P_0 = \{S\}$ and $\varphi_0(S) = tt$ and therefore $\forall s \in S.\ s \models tt$ as required.

Induction hypothesis: Let $A(k)$ hold for an arbitrary but fixed $k \in 0 .. (m - 1)$.

Induction step ($k \rightarrow k + 1$): Let $X \in P_{k+1}$ and $s \in S$ be both arbitrary but fixed. We proceed by proving the two required implications depending on the cases within Definition 13 for the construction of $\varphi_{k+1}(X)$.

Case 1: $\exists Y \in P_k.\ (B_k, a_k) \mid Y$ and $X = Y^{a_k}_{B_k}$

(i) $s \in X$ implies $s \models \varphi_{k+1}(X)$:
We assume that $s \in X$. Because of $(B_k, a_k) \mid Y$, we know that $X \subset Y$ and therefore $s \in Y$. Due to the induction hypothesis, we have $s \models \varphi_k(Y)$. Based on $s \in Y^{a_k}_{B_k}$, we know that s has an a_k-successor in B_k (Definition 11). Let s' denote such a successor. By induction hypothesis, it follows that $s' \models \varphi_k(B_k)$. Due to $s \xrightarrow{a_k} s'$ and HML semantics, we know that $s \models \langle a_k \rangle\, \varphi_k(B_k)$. Thus, we have $s \models \varphi_k(Y) \wedge \langle a_k \rangle\, \varphi_k(B_k) = \varphi_{k+1}(X)$.

(ii) $s \models \varphi_{k+1}(X)$ implies $s \in X$:
We assume $s \models \varphi_{k+1}(X)$, which means that $s \models \varphi_k(Y) \wedge \langle a_k \rangle\, \varphi_k(B_k)$. Because of the induction hypothesis and the first conjunct, we know that $s \in Y$. Using HML semantics and the induction hypothesis, the second conjunct implies that there exists a transition $s \xrightarrow{a_k} s'$ such that $s' \in B_k$. Thus, we have $s \in (Y \cap {a_k}^{-1} B_k) = Y^{a_k}_{B_k} = X$.

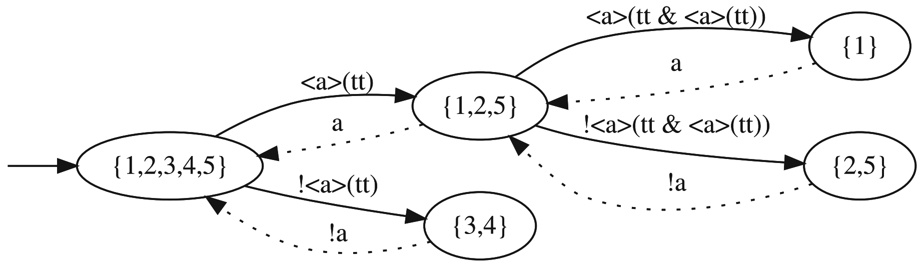

Fig. 2 Partition refinement history and corresponding class-distinguishing predicates based on the input-LTS from Fig. 1 as generated by a tool that implements Algorithm 1. Dotted arrows denote splitters

Case 2: $\exists Y \in P_k.\ (B_k, a_k) \mid Y$ and $X = Y_{B_k}^{\not a_k}$
Analogous to case 1.

Case 3: X is not split by (B_k, a_k)
Then $X \in P_k$ and $\varphi_{k+1}(X) = \varphi_k(X)$ and therefore by induction as desired:

$$s \in X \text{ iff } s \models \varphi_k(X)\ (= \varphi_{k+1}(X))$$

□

In summary, the class-distinguishing functions are defined such that they reflect information about each splitter in HML. This information can be summarized on the basis of a decision tree with node set

$$N = \bigcup_{k \in 0\,..\,m} P_k$$

and edges set

$$E = \bigcup_{i \in 0\,..\,m-1} \{(C, C') \in P_i \times P_{i+1} \mid C' \subset C\}$$

where for each $k \in 0\,..\,m-1$, all the nodes $C \in P_k$ that are split by (B_k, a_k) are annotated with the predicate $\langle a_k \rangle\, \varphi_k(B_k)$. If this predicate evaluates to true, then one continues with $C_{B_k}^{a_k}$, otherwise with $C_{B_k}^{\not a_k}$.

Figure 2 illustrates such a decision tree based on the LTS depicted in Fig. 1. As Fig. 2 is intended to also display the partition classes, the predicates were moved to the edges.

In the following, we present our algorithm that incorporates both the partition refinement and the corresponding labeling based on class-distinguishing functions.

5.3 Algorithm for generating an SCDF

We use Algorithm 1 to generate an SCDF (Definition 8). This algorithm is conceptually based on the “naive method” from [21]. As such, the derived partition P is actually the coarsest partition w.r.t. bisimulation. Additionally, our iteratively defined class-distinguishing function (Definition 13) is incorporated.

Algorithm 1 Algorithm for generating an SCDF

```
Input: A finite-state LTS $L = (S, \Sigma_L, \rightarrow)$
Output: An SCDF of $L$
1: function StateFormulas($S, \Sigma_L, \rightarrow$)
2:     $P \leftarrow \{S\}$
3:     $W \leftarrow P \times \Sigma_L$
4:     $\varphi[S] \leftarrow tt$
5:     while $W \neq \emptyset$ do
6:         $(B, \alpha) \leftarrow pop(W)$
7:         $splitterFormula \leftarrow \varphi[B]$
8:         for all $Y \in P$ which are split by $(B, \alpha)$ do
9:             $classFormula \leftarrow \varphi[Y]$
10:            replace $Y$ in $P$ by $Y^{\alpha}_{B}$ and $Y^{\not\alpha}_{B}$
11:            delete entry for $Y$ in $\varphi$
12:            $\varphi[Y^{\alpha}_{B}] \leftarrow classFormula \wedge \langle\alpha\rangle\, splitterFormula$
13:            $\varphi[Y^{\not\alpha}_{B}] \leftarrow classFormula \wedge \neg\langle\alpha\rangle\, splitterFormula$
14:            $W \leftarrow$ UpdateWorkset($W, Y, (B, \alpha), \Sigma_L$)
15:    return $\{\varphi[X] \mid X \in P\}$
```

Algorithm 2

```
1: function UpdateWorkset($W, Y, (B, \alpha), \Sigma_L$)
2:     for all $\beta \in \Sigma_L$ do
3:         if $(Y, \beta) \in W$ then remove $(Y, \beta)$ from $W$
4:         add $(Y^{\alpha}_{B}, \beta)$, $(Y^{\not\alpha}_{B}, \beta)$ to $W$
5:     return W
```

The worklist of Algorithm 1 contains potential splitters (Definition 11). Whenever the current partition P is refined and new classes are thereby added to it, these classes are combined with every possible transition label and added to the worklist (Algorithm 2). Every time that such a partition refinement occurs, the class-distinguishing function φ is updated according to Definition 13 in order to incorporate information about the most recent splitter (lines 11–13 of Algorithm 1).

Note that the order in which splitters are applied is not defined by Algorithm 1. In the following, we assume a fixed order of splitters. As stated in Sect. 5.2 and visible in the pseudo code (lines 12 and 13 of Algorithm 1), the definition of the class-distinguishing function throughout the algorithm's execution is based on that order of splitters. The sequence of splitters is exactly the sequence of those pairs received in line 6 of Algorithm 1 for which the inner for-all loop (line 8) executes at least one iteration.

Example 2 (Algorithm 1) Consider the LTS in Fig. 1 as input for Algorithm 1. Figure 2 illustrates the two refinements during an execution of Algorithm 1 for this input LTS. The actual class-distinguishing formulas generated by Algorithm 1 are as follows. In line with Sect. 5.2, we use a subscript index to refer to different refinements (Definition 12).

The algorithm's internal variables are initialized to $P_0 = \{S = \{1, 2, 3, 4, 5\}\}$ (line 2) and $\varphi_0[S] = tt$ (line 4).

The first splitter is (S, a) (line 6). It splits the class S into the sets $S^{a}_{S} = \{1, 2, 5\}$ and $S^{\not a}_{S} = \{3, 4\}$ (line 10). This split occurs because on the one hand, we have $1 \xrightarrow{a} 2, 2 \xrightarrow{a} 3$, $5 \xrightarrow{a} 3$, and $2, 3, 5 \in S$, and on the other hand, states 3 and 4 do not have outgoing transitions labeled a (cf. Fig. 1). The new partition is given by $P_1 = \{\{1, 2, 5\}, \{3, 4\}\}$ (line 10) and the formulas are $\varphi_1[\{1, 2, 5\}] = tt \wedge \langle a \rangle\, tt$ (line 12) and $\varphi_1[\{3, 4\}] = tt \wedge \neg\langle a \rangle\, tt$ (line 13).

The second splitter is $(\{1, 2, 5\}, a)$. Note that in the meantime, several other checks for possible refinements may have been performed that failed. For simplicity, we will only list actual splitters (Definition 11). The splitter $(\{1, 2, 5\}, a)$ separates the class $\{1, 2, 5\}$ into the sets $\{1\}$ and $\{2, 5\}$. Again, this happens because on the one hand, we have $1 \xrightarrow{a} 2$ with $2 \in \{1, 2, 5\}$, and on the other hand, states from $\{2, 5\}$ do not possess outgoing transitions labeled a that end in $\{1, 2, 5\}$. The new and final partition is given by $P = \{\{1\}, \{2, 5\}, \{3, 4\}\}$ and the formulas are:

$$\begin{aligned}
\varphi_2[\{1\}] &= tt \wedge \quad \langle a \rangle\, tt \wedge \quad \langle a \rangle\, (tt \wedge [a]\, tt) \\
\varphi_2[\{2, 5\}] &= tt \wedge \quad \langle a \rangle\, tt \wedge \neg\langle a \rangle\, (tt \wedge [a]\, tt) \\
\varphi_2[\{3, 4\}] &= tt \wedge \neg\langle a \rangle\, tt
\end{aligned}$$

Note that states 2 and 5 as well as 3 and 4 are bisimilar, respectively. The algorithm terminates (after potentially executing a few more checks for splitters) because the splitter potential is exhausted.

Algorithm 1 resembles the "naive" algorithm by Kanellakis and Smolka [21] which is known to run in cubic time and to terminate with the coarsest partition up to bisimulation. Thus, the following theorem is a consequence of Theorem 3 which guarantees the correct labeling of the partition classes.

Theorem 4 (Correctness of Algorithm 1) *Given an LTS L, Algorithm 1 terminates with an SCDF for L.*

The following theorem summarizes the main result of this paper. Please recall that by definition of an SCDF, every state $s \in S$ has a formula $\varphi \in \Phi$ with $s \models_L \varphi$:

Theorem 5 (Main theorem) *Let L be an LTS $L = (S, \Sigma_L, \rightarrow)$, Φ an SCDF for L generated by Algorithm 1, χ_I the GCI for L based on Φ, and $s \in S$, $\varphi \in \Phi$ with $s \models_L \varphi$. Then for any $P_{L'}(s')$ based on some LTS $L' = (S', \Sigma'_{L'}, \rightarrow')$ with $s' \in S'$, we have:*

$$P_{L'}(s') \models \varphi \wedge \boldsymbol{AG}(\chi_I) \;\; \textit{iff} \;\; P_L(s) \sim P_{L'}(s')$$

Proof Theorem 4 guarantees that Φ is an SCDF which, by Theorem 2, implies $P_L(s) \models \varphi \wedge \mathbf{AG}(\chi_I)$. Thus, applying Theorem 1 yields the desired statement. □

6 Discussion of related work

The coincidence theorem of Hennessy and Milner states that, given a finitely-branching LTS, two states are bisimilar if and only if they enjoy the same set of HML formulas (first stated 1980 in [17] and again in [18]). Thus, HML has distinguishing power: any two non-bisimilar finitely-branching LTSs can be distinguished by a formula in HML.

The idea of characteristic formulas goes beyond distinguishability: It requires that, given a finitely-branching LTS, there exists a (single) formula that distinguishes this LTS from any other non-bisimilar (finitely-branching) LTS. It is clear that this "swap of quantifiers" leads to a requirement that cannot be satisfied by HML as HML formulas are limited in their sensitivity to finite prefixes of computation trees.

In 1984, Graf and Sifakis published a scheme by which characteristic formulas for *finite CCS processes*[5] can be derived [13,15]. Such a scheme is usually referred to as *(characteristic) translation*. The pattern introduced by Graf and Sifakis, which we refer to as "diamond-box-pattern", is used in almost all following publications on characteristic formulas, including this paper (see Definition 5). Graf and Sifakis also extended the idea of Hennessy and Milner in the sense that they interpret the coincidence theorem as a strict requirement for sensible temporal logics [14]. Thereby such logics are "sufficiently powerful to distinguish non-congruent terms". Indeed, as later shown in various publications, many famous temporal logics such as HML, CTL, and μ-calculus satisfy this claim.

Browne, Clarke, and Grumberg were the first to take the step from finite to finite-state systems in 1987, again based on the "diamond-box-pattern" by Graf and Sifakis, but this time generalized using implication in order to deal with cycles [2]. The key idea, which is also used in this paper, consists of two steps. First, find a formula for each state that distinguishes it from every other state of the considered finite-state system. Second, show that the generalized "diamond-box-pattern" for connecting these distinguishing formulas is sufficient to define characteristic formulas for finite-state systems. In essence, they observed that non-bisimilar states of finite-state systems can be distinguished by a finite prefix of the corresponding computation tree[6] which allowed them to infer their distinguishing formulas simply as characteristic formulas up to a certain depth of bisimilarity.

The paper [28] introduced a technique for deriving characteristic formulas in the modal μ-calculus for finite-state processes (with divergence potential [26,27]). Key idea was to associate each state of the LTS with a fixed point variable and to construct a modal equational system on the basis of the "diamond-box-pattern" which can then be translated into the modal μ-calculus. As this translation leads to formulas of exponential size, [30] focused on modal equational systems as they provide characterizations that are linear in the size of a property, making bisimulation checking via model checking attractive.

Like [2], the approach presented in this paper is based on the combination of distinguishing formulas using the generalized "diamond-box-pattern". The main difference is the construction of the distinguishing SCDF based on Algorithm 1 which has a reduced form of linear size in terms of a recursion-free equational system. Key to this construction is an extension of Kanellakis and Smolka's partition refinement algorithm for bisimulation checking [21] in order to incrementally compute distinguishing formulas for the individual partition classes. By construction, this leads to distinguishing formulas with linearly many distinct subformulas. Therefore, their corresponding representation in terms of a directed acyclic graph or, equivalently, as recursion-free equational system, is guaranteed to be linear in the number of bisimulation equivalence classes. Thus, the characteristic formulas proposed in this paper can be regarded as very concise versions and therefore practical optimizations of the characteristic formulas of [2].

7 Conclusion and outlook

We have shown that HML, despite its structural limitations, is sufficiently expressive to specify an initial property φ_0 and a characteristic invariant χ_I for an arbitrary finite-state process P such that $\varphi_0 \wedge \mathbf{AG}(\chi_I)$ is a characteristic formula for P. This means, in particular,

[5] Such CCS processes can be represented as a labeled tree with finite depth.

[6] This is an immediate consequence of the fact that bisimilarity coincides with the limit of n-bisimilarity for finitely branching systems—a key property originally exploited in [17].

that it is sufficient to check an HML formula for each state of a *candidate* finite-state process to verify that it is bisimulation equivalent to P. In fact, our proofs do not require the candidate processes to be finite state. In particular, our theorem also holds for context-free [5] and push-down candidate systems [6].

The first model checking algorithm for context-free systems has been presented in 1992 in [5]. It was linear in the size of the context-free process representation and exponential in the size of the, in this case alternation-free, modal μ-calculus formula, or better, in the size of a corresponding modal equational system. Extensions covered pushdown-processes [6] and later sequential processes for the full modal μ-calculus [7]. Combining these algorithms, whose implementation in the fixed point analysis machine has been presented in CONCUR 1995 [29], with an algorithm that constructs characteristic formulas in terms of modal equational systems (e.g. [30]) immediately implies:

> Bisimulation checking of context-free processes with a finite-state process can be performed in exponential time via model checking of characteristic formulas.

Even though not explicitly formulated back then, there existed a corresponding implementation already in 1995 [29]. Related results concerning equivalence checking involving infinite-state systems can be found in [3,4,20,22–24].

Currently, we are re-implementing the algorithms for characteristic formula construction and context-free model checking, also with the goal to experimentally evaluate their practical efficiency. Our model checking algorithm uses shared BDDs[7] to efficiently represent the property transformers that are characteristic of our second-order treatment of context-free systems. It is not clear how the choice of characteristic formula construction interferes with the second-order fixed point computation, whether there are advantageous process patterns, or whether there are certain technological bottlenecks that can be overcome. An example for the latter category is the treatment of the identity function, which is intuitively very simple, but whose representation in terms of shared BDDs is bound to explode.

Acknowledgements Open Access funding provided by Projekt DEAL.

References

1. Bradfield, J.C., Stirling, C.: Modal mu-calculi. In: Blackburn, P., van Benthem, J.F.A.K., Wolter, F. (eds.) Handbook of Modal Logic, Studies in logic and practical reasoning, vol. 3, pp. 721–756. North-Holland (2007). https://doi.org/10.1016/s1570-2464(07)80015-2
2. Browne, M.C., Clarke, E.M., Grumberg, O.: Characterizing Kripke structures in temporal logic. In: Ehrig, H., Kowalski, R.A., Levi, G., Montanari, U. (eds.) TAPSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Pisa, Italy, March 23–27, 1987, Volume 1: Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming (CAAP'87), Lecture Notes in Computer Science, vol. 249, pp. 256–270. Springer (1987). https://doi.org/10.1007/3-540-17660-8_60

[7] A shared BDD is an aggregation of multiple BDDs into one BDD with multiple start nodes.

3. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification on infinite structures. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 545–623. Elsevier Science, Amsterdam (2001). https://doi.org/10.1016/B978-044482830-9/50027-8. http://www.sciencedirect.com/science/article/pii/B9780444828309500278
4. Burkart, O., Caucal, D., Steffen, B.: Bisimulation collapse and the process taxonomy. In: Montanari, U., Sassone, V. (eds.) CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26–29, 1996, Proceedings, Lecture Notes in Computer Science, vol. 1119, pp. 247–262. Springer (1996). https://doi.org/10.1007/3-540-61604-7_59
5. Burkart, O., Steffen, B.: Model checking for context-free processes. In: Cleaveland, W. (ed.) CONCUR '92, pp. 123–137. Springer, Berlin Heidelberg, Berlin, Heidelberg (1992)
6. Burkart, O., Steffen, B.: Pushdown Processes: Parallel Composition and Model Checking, pp. 98–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1994). https://doi.org/10.1007/BFb0015001
7. Burkart, O., Steffen, B.: Model checking the full modal mu-calculus for infinite sequential processes. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) Automata, Languages and Programming, pp. 419–429. Springer, Berlin Heidelberg, Berlin, Heidelberg (1997)
8. Fischer, N., van Glabbeek, R.: Axiomatising infinitary probabilistic weak bisimilarity of finite-state behaviours. J. Logical Algebraic Methods Program. **102**, 64–102 (2019). https://doi.org/10.1016/j.jlamp.2018.09.006. http://www.sciencedirect.com/science/article/pii/S2352220817302201
9. Floyd, R.W.: Assigning Meanings to Programs, pp. 65–81. Springer Netherlands, Dordrecht (1993). https://doi.org/10.1007/978-94-011-1793-7_4
10. Fokkink, W., van Glabbeek, R., de Wind, P.: Compositionality of Hennessy–Milner logic through structural operational semantics. In: Lingas, A., Nilsson, B.J. (eds.) Fundamentals of Computation Theory, pp. 412–422. Springer, Berlin Heidelberg, Berlin, Heidelberg (2003)
11. Fokkink, W., van Glabbeek, R., de Wind, P.: Compositionality of Hennessy–Milner logic by structural operational semantics. Theor. Comput. Sci. **354**(3), 421–440 (2006). https://doi.org/10.1016/j.tcs.2005.11.035. http://www.sciencedirect.com/science/article/pii/S0304397505008819
12. Fokkink, W., van Glabbeek, R., de Wind, P.: Divide and congruence: from decomposition of modalities to preservation of branching bisimulation. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) Formal Methods for Components and Objects, pp. 195–218. Springer, Berlin Heidelberg, Berlin, Heidelberg (2006)
13. Graf, S., Sifakis, J.: A modal characterization of observational congruence on finite terms of CCS. In: Paredaens, J. (ed.) Automata, Languages and Programming, 11th Colloquium, Antwerp, Belgium, July 16–20, 1984, Proceedings, Lecture Notes in Computer Science, vol. 172, pp. 222–234. Springer (1984). https://doi.org/10.1007/3-540-13345-3_20
14. Graf, S., Sifakis, J.: A logic for the description of non-deterministic programs and their properties. Inf. Control **68**(1–3), 254–270 (1986). https://doi.org/10.1016/S0019-9958(86)80038-9
15. Graf, S., Sifakis, J.: A modal characterization of observational congruence on finite terms of CCS. Inf. Control **68**(1–3), 125–145 (1986). https://doi.org/10.1016/S0019-9958(86)80031-6
16. Habib, M., Paul, C., Viennot, L.: Partition refinement techniques: an interesting algorithmic tool kit. Int. J. Found. Comput. Sci. **10**(2), 147–170 (1999). https://doi.org/10.1142/S0129054199000125
17. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. In: de Bakker, J.W., van Leeuwen, J. (eds.) Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14–18, 1980, Proceedings, Lecture Notes in Computer Science, vol. 85, pp. 299–309. Springer (1980). https://doi.org/10.1007/3-540-10003-2_79
18. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM **32**(1), 137–161 (1985). https://doi.org/10.1145/2455.2460
19. Hopcroft, J.: An n log n algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations, pp. 189–196. Elsevier (1971). https://doi.org/10.1016/B978-0-12-417750-5.50022-1. http://www.sciencedirect.com/science/article/pii/B9780124177505500221
20. Jancar, P., Kucera, A., Mayr, R.: Deciding bisimulation-like equivalences with finite-state processes. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13–17, 1998, Proceedings, Lecture Notes in Computer Science, vol. 1443, pp. 200–211. Springer (1998). https://doi.org/10.1007/BFb0055053
21. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. In: Probert, R.L., Lynch, N.A., Santoro, N. (eds.) Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17–19, 1983, pp. 228–240. ACM (1983). https://doi.org/10.1145/800221.806724
22. Kucera, A., Jancar, P.: Equivalence-checking on infinite-state systems: techniques and results. TPLP **6**(3), 227–264 (2006). https://doi.org/10.1017/S1471068406002651

23. Kucera, A., Mayr, R.: On the complexity of semantic equivalences for pushdown automata and BPA. In: Diks, K., Rytter, W. (eds.) Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26–30, 2002, Proceedings, Lecture Notes in Computer Science, vol. 2420, pp. 433–445. Springer (2002). https://doi.org/10.1007/3-540-45687-2_36
24. Kucera, A., Schnoebelen, P.: A general approach to comparing infinite-state systems with their finite-state specifications. Theor. Comput. Sci. **358**(2–3), 315–333 (2006). https://doi.org/10.1016/j.tcs.2006.01.021
25. Mayr, R.: Decidability of model checking with the temporal logic EF. Theor. Comput. Sci. **256**(1), 31–62 (2001). https://doi.org/10.1016/S0304-3975(00)00101-8. http://www.sciencedirect.com/science/article/pii/S0304397500001018. ISS
26. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3
27. Milner, R.: Communication and Concurrency. PHI Series in Computer Science. Prentice Hall, Upper Saddle River (1989)
28. Steffen, B.: Characteristic formulae. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11–15, 1989, Proceedings, Lecture Notes in Computer Science, vol. 372, pp. 723–732. Springer (1989). https://doi.org/10.1007/BFb0035794
29. Steffen, B., Claßen, A., Klein, M., Knoop, J., Margaria, T.: The fixpoint-analysis machine. In: Lee, I., Smolka, S.A. (eds.) CONCUR '95: Concurrency Theory, pp. 72–87. Springer, Berlin Heidelberg, Berlin, Heidelberg (1995)
30. Steffen, B., Ingólfsdóttir, A.: Characteristic formulas for processes with divergence. Inf. Comput. **110**(1), 149–163 (1994). https://doi.org/10.1006/inco.1994.1028
31. Steffen, B., Isberner, M., Jasper, M.: Playing with abstraction and representation. In: Probst, C.W., Hankin, C., Hansen, R.R. (eds.) Semantics, Logics, and Calculi, pp. 191–213. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-27810-0_10
32. van Glabbeek, R.: A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In: Borzyszkowski, A.M., Sokołowski, S. (eds.) Mathematical Foundations of Computer Science 1993, pp. 473–484. Springer, Berlin Heidelberg, Berlin, Heidelberg (1993)
33. van Glabbeek, R.: The linear time—branching time spectrum ii. In: Best, E. (ed.) CONCUR'93, pp. 66–81. Springer, Berlin Heidelberg, Berlin, Heidelberg (1993)
34. van Glabbeek, R.: What is branching time semantics and why to use it? In: Nielsen, M. (ed.) The Concurrency Column, Bulletin of the EATCS 53, pp. 190–198 (1994)
35. van Glabbeek, R.: The linear time—branching time spectrum i. The semantics of concrete, sequential processes. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 3–99. Elsevier Science, Amsterdam (2001). https://doi.org/10.1016/B978-044482830-9/50019-9. http://www.sciencedirect.com/science/article/pii/B9780444828309500199
36. van Glabbeek, R.: A Characterisation of Weak Bisimulation Congruence, pp. 26–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11601548_4
37. van Glabbeek, R.: On cool congruence formats for weak bisimulations. Theor. Comput. Sci. **412**(28), 3283–3302 (2011). https://doi.org/10.1016/j.tcs.2011.02.036. http://www.sciencedirect.com/science/article/pii/S0304397511001605
38. van Glabbeek, R., Luttik, B., Trčka, N.: Branching bisimilarity with explicit divergence. Fundamenta Informaticae **93**(4), 371–392 (2009)
39. van Glabbeek, R., Smolka, S., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Inf Comput **121**(1), 59–80 (1995). https://doi.org/10.1006/inco.1995.1123. http://www.sciencedirect.com/science/article/pii/S0890540185711236
40. van Glabbeek, R., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. ACM **43**(3), 555–600 (1996). https://doi.org/10.1145/233551.233556