

BNClassifier: Classifying Boolean Models by Dynamic Properties^{*}

Nikola Beneš¹, Luboš Brim¹, Ondřej Huvar¹,
Samuel Pastva², and David Šafránek¹✉

¹ Faculty of Informatics, Masaryk University, Brno, Czech Republic
`{xbenes3,brim,xhuvar,safranek}@fi.muni.cz`

² Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria
`samuel.pastva@ist.ac.at`

Abstract. The ability to automatically classify models according to their properties is critical in model-based disciplines where the number of models to be dealt with is extremely large. Here, we consider discrete models resulting from possible interpretations of unknown interactions in Boolean networks, an essential modelling framework in systems biology. The classification criteria are expressed in a hybrid extension of CTL. We present our novel tool BNCLASSIFIER, its architecture, symbolic implementation, and its use in computational systems biology. We also report on the evaluation of the effectiveness and practicality of the tool.

1 Introduction

Model-based computer systems development allows designers to create a more abstract and high-level view of the system, which can help them to understand the system’s requirements and design. Model checking is a traditional formal method used in model-based development to ensure that a model of a computer system satisfies the given property. Model measuring [19, 20] tries to quantify how “far” a model is from satisfying the given property. *Model classifying* is used to divide a set of models into classes satisfying the same properties.

In this tool paper, we propose the tool BNCLASSIFIER capable of classifying models by dynamic properties expressed in the hybrid extension of the branching-time logic CTL (HCTL) [22, 14, 15]. The motivation for developing the tool comes from the field of systems biology. It is well-known that systems biology is particularly amenable to applying formal computer science methods. Systems biology is also a model-based discipline. It uses models to study complex interactions in biological systems to better understand the processes that happen in such systems, as well as to grasp the emergent properties of such a system as a whole. As advocated by Fisher and Henzinger [12], the *discrete state-transition*

^{*} The work has been supported by the Czech Science Foundation grant No. GA22-10845S. This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 101034413.

based computational models may explain the mechanisms behind a biological system more intuitively than traditionally used mathematical models. Furthermore, such models form a natural basis for the application of formal mathematical approaches to modelling and analysis that has been developed for distributed computer systems [4].

In modelling biological reality, modellers consider many different abstractions resulting in many particular models. *Boolean Networks (BNs)* [21, 32] represent an essential qualitative modelling framework in computational systems biology [33] and in systems pharmacology [9]. BNs capture the interactions among system’s components.

Typically, in biological modelling the interactions are often not completely known. It may be known that some of the components interact but the exact mechanism is not established yet. To deal with such uncertainty, the modeller might consider a set of principally possible interpretations of unknown parts leading to a set of particular models. For Boolean models, we consider the so-called partially specified Boolean networks (PSBNs), where this set of models can be represented as a single edge-coloured transition system.

For a model-based understanding of complex reality, it is crucial to sort out or *classify* the models by diverse dynamic properties. The resulting classes, in fact “define” the abstract concepts essential for the given model. The dynamic properties of interest typically reflect long-term behaviour, such as reaching a final steady state (or branching to several different steady states or oscillatory regimes). To express such emergent dynamic phenomena occurring in complex systems dynamics, we utilise the branching-time temporal logic (CTL) extended with hybrid operators [23]. The importance of hybrid operators in the context of Boolean Networks has been discussed in [2].

Contribution. We present the tool BNCLASSIFIER that classifies Boolean networks with respect to the properties expressed as formulae of HCTL. The set of networks to be classified results from possible interpretations of unknown logical relations in the network and is given as a partially specified BN. The properties are checked using a novel, fully symbolic BDD-based HCTL model checker. A distinguished feature of the checker is that it handles the entire set of networks at once. BNCLASSIFIER is available at <https://github.com/sybila/biodivine-bn-classifier>. For the visualisation part, we use the concept of decision trees that is used to determine model classes satisfying the same sets of HCTL formulae. The tool’s main features are demonstrated in a real case of an important signalling network controlling the cell life-cycle and growth.

Related Tools. Though BNCLASSIFIER is focused on PSBNs, due to its architecture, it can be also considered as the very first entirely symbolic model checker supporting full HCTL over general Kripke structures. Historically, the first HCTL model checker [31] has been explicit and it utilised a fragment of HCTL over acyclic structures (not general KSs). Another model checker HLMC [16] has implemented the labelling algorithms [13], allowing to handle full HCTL. Antelope [2] performs HCTL model checking on KSs encoding operational semantics of asynchronous BNs and also works—to some extent—

with PSBNs. However, there are two critical limitations: (i) the model checking algorithm verifies the formula universally for all interpretations, i.e., it is not possible to compute only the satisfying interpretations, (ii) while the CTL part of the algorithm is symbolic, the computation regarding the hybrid operators is enumerative: the formulae employing the state variable quantifiers are solved by explicitly calling the labelling algorithm once for each possible state. This causes a considerable decrease in efficiency for larger models or when hybrid operators are nested.

In [5], we have presented the tool AEON for the symbolic identification of attractors in PSBNs. Subsequently, AEON has been enriched with decision trees [6] to ease the interpretation of produced results. BNCLASSIFIER is a completely novel tool that implements, for the first time, coloured HCTL model checking over PSBNs. However, in the architecture of BNCLASSIFIER we employ our general library for manipulating BDDs (`lib.bdd`) that is also used in AEON. For the GUI, we adopt the AEON-based concept of decision trees that is substantially re-implemented in BNCLASSIFIER to determine model classes satisfying the same sets of HCTL formulae. The HCTL properties decision trees visual explorer has a completely new architecture based on Tauri framework [1].

2 Tool Workflow

We start with a motivating example on which we intuitively introduce the core concepts. The formal definitions are given in the subsequent sections. The example presents a simple partially specified Boolean network, its dynamical behaviour, several biologically motivated constraints, and a classification tree constructed by our tool.

Consider the biological phenomenon of a cell cycle: a periodic process consisting of several phases, responsible for cell growth and division. A malfunction in the cell cycle can have widespread consequences, such as cancer. We create a simple model of a three-phase cell cycle. The model consists of three Boolean variables x_1, x_2, x_3 . The variable x_i is set to true whenever the cell cycle phase i is active. The cell cycle phases, hence the variables, influence each other. In biology, such influences are typically called *regulations*.

How a variable is influenced by its regulators is given in the form of a so-called (Boolean) *update function*. In typical modelling scenarios, the update functions are not completely known. We represent such uncertainty in terms of *function symbols*. The particular interpretations of the function symbols determine the concrete behaviour of the system. In other words, the model containing function symbols stands for a set of concrete models. In our example, the model is described by the update functions F_1, F_2, F_3 containing function symbols \mathbf{f}, \mathbf{g} .

In addition to the update functions, the models also capture information about the character of particular regulations. In our example, some of the regulations are positive (*activation*; green colour) and some negative (*inhibition*; red colour). A positive regulation $x_i \rightarrow x_j$ means that an increase in x_i (from 0 to 1) cannot

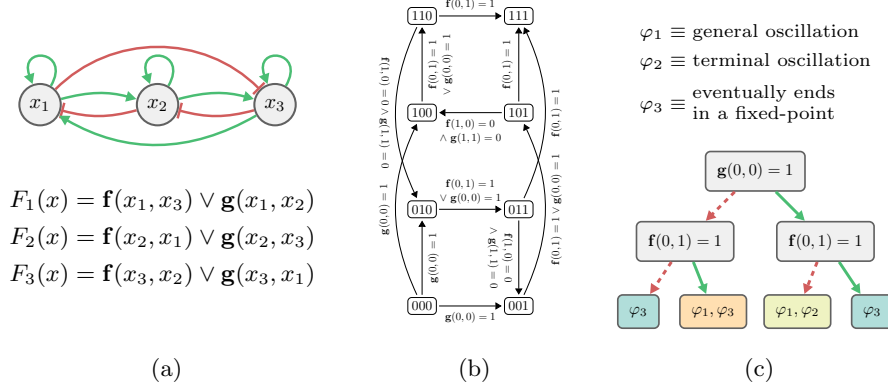


Fig. 1. (a) The partially specified Boolean network of the toy cell cycle model (bottom) and its influence graph (top). (b) The possible dynamics of the partially specified model. Each edge is annotated with a condition that is sufficient assuming all F_i must also agree with the presented influence graph. (c) The temporal classification constraints (top) and the decision tree generated based on these constraints (bottom).

lead to a decrease of x_j ; symmetrically for the case of negative regulation. In our example, this information is displayed in the influence graph shown in Fig. 1 (a).

We restrict ourselves to the interpretations of \mathbf{f} and \mathbf{g} that respect the influence graph. This leaves 10 valid interpretations instead of 256 (any pair of 16 possible binary functions). In the terminology of coloured model checking, we call these interpretations *colours*.

The semantics of a partially-specified BN is given as an *edge-coloured* state-transition graph; see Fig. 1 (b). The *states* are valuations of model variables, i.e., bit vectors of length n . Transitions are determined by the update functions. In Fig. 1 (b) the transitions are labelled with expressions restricting the possible interpretations of \mathbf{f} and \mathbf{g} that enable the given transition. By fixing a particular interpretation, we obtain a classical (uncoloured) state-transition graph on which temporal properties can be evaluated.

To study the behaviour of our partially specified model, Fig. 1 (c) informally introduces three temporal properties (their formal definitions are given in Section 3). Here, φ_1 signifies that there is a cycle (i.e. an *oscillation*) somewhere in the state space of the model, while φ_2 strengthens this property and requires that this cycle cannot be escaped (i.e. it is *terminal*). Finally, φ_3 says that the system eventually (under any fair path) reaches a fixed-point state. As such, note that φ_2 implies φ_1 , while φ_3 and φ_2 are mutually exclusive (a path that reaches a terminal cycle cannot reach a fixed-point). In the terminology of dynamical systems, the terminal cycles and fixed point states represent examples of *attractors* [25].

These properties are of biological interest: the presence of oscillation is a crucial property of the cell cycle, while various fixed-points can correspond to different failure modes of this simple system. However, note that these are not *requirements*: their goal is not to directly specify a single “correct” model. Instead,

their validity guides the exploratory part of the modelling process, during which the modeller selects additional real-world experiments to further refine the model. In other words, the properties do not specify what the model *must* do, but rather what the model *could* do. It is up to the modeller to determine which of these hypotheses should be further explored in practice.

To guide the modeller, once the property validity is determined by our tool, we provide an interactive decision tree which explores the relationship between the colours (interpretations) and the individual properties. A possible version of such decision tree is shown in Fig. 1 (c). Here, we see which conditions lead to the satisfaction of the properties. In particular, we have three possible outcomes: the model always directly reaches a fixed-point (only φ_3); the model can cycle, but eventually reaches a fixed-point (φ_1, φ_3); or the model can reach a terminal cycle (φ_1, φ_2). These three outcomes partition the set of possible interpretations into three *classes*. The first outcome (only φ_3) corresponds to those interpretations under which $\mathbf{f}(0, 1) = \mathbf{g}(0, 0)$; the second outcome (φ_1, φ_3) to those under which $\mathbf{f}(0, 1) = 1$ and $\mathbf{g}(0, 0) = 0$; and the third outcome (φ_1, φ_2) to those under which $\mathbf{f}(0, 1) = 0$ and $\mathbf{g}(0, 0) = 1$.

3 Model Classification Problem

Let us now formally define the classification problem. We start by briefly describing the inputs: HCTL formulae and partially specified Boolean networks. HCTL extends the standard CTL temporal logic with hybrid operators that quantify and reference model states using state variables. This gives the logic the power to express biologically interesting behaviour, such as the attractor structure of the model, that would otherwise be impossible in (pure) CTL.

Hybrid CTL. Let AP be a set of atomic propositions. We say that \mathcal{K} is a *Kripke structure* (KS) over AP iff $\mathcal{K} = (S, R, L)$, where $S \neq \emptyset$ is a finite set of states, $R \subseteq S \times S$ is a total transition relation, and $L : AP \rightarrow 2^S$ is a labelling assigning to each atomic proposition the set of states where the proposition holds.

To describe properties of Kripke structures, we consider the hybrid extension of the branching-time temporal logic CTL (HCTL) [22, 14, 15]. Let Var be a set of *state variables*. The syntax for HCTL formulae is given by the grammar (here p ranges over AP and x over Var):

$$\varphi ::= p \mid x \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{EX} \varphi \mid \mathbf{E}[\varphi \mathbf{U} \varphi] \mid \mathbf{A}[\varphi \mathbf{U} \varphi] \mid @_x.\varphi \mid \downarrow x.\varphi \mid \exists x.\varphi.$$

The temporal operators **EX**, **EU**, **AU** have their usual meaning. The intuition behind the hybrid operators is: the *at operator* $@_x.\varphi$ means “continue evaluating φ at the state x ”, the *down-arrow binder* $\downarrow x.\varphi$ stands for “name the current state of evaluation x and then proceed to checking whether φ holds under this assumption”. We define the derived universal quantification operator as $\forall x.\varphi \equiv \neg \exists x.\neg\varphi$. We also make use of the usual propositional abbreviations for \vee , \Rightarrow , \Leftrightarrow as well as the temporal ones **EF**, **AF**, **EG**, **AG**, **AX**. The operators $\downarrow x.\varphi$ and $\exists x.\varphi$ act as variable binders; all occurrences of x in φ are *bound* and not free. We say that an HCTL formula is *closed* if it contains no free variables.

In the following, we use $\mathcal{K} \models \varphi$ to mean that all states of \mathcal{K} satisfy the HCTL formula φ . For more details and a formal definition of the semantics, see e.g. [22].

Consider the three properties stated informally in Fig. 1 (c). The first property can be formulated in HCTL as $\varphi_1 = \exists x. @x. \mathbf{EX} \mathbf{EF} x \wedge \neg \mathbf{EX} x$, i.e. “there exists a state that has a path to itself, but does not have a self-loop”. The second property can be formulated as $\varphi_2 = \exists x. @x. \mathbf{AX} \mathbf{AF} x \wedge \neg \mathbf{EX} x$, i.e. “there exists a state from which all paths lead back to itself, but does not have a self loop”. Finally, the third property is formulated as $\varphi_3 = \forall x. (@x. \mathbf{AG} \mathbf{EF} x) \Rightarrow (@x. \mathbf{AX} x)$, i.e. “if a state lies in an attractor, it is a fixed point”.

Partially Specified Boolean Networks. A *Boolean network* \mathcal{B} assumes n Boolean variables $Var_n = \{v_1, \dots, v_n\}$ and consists of n Boolean update functions $\{F_1, \dots, F_n\}$ (one for each variable), such that $F_i : \{0, 1\}^n \rightarrow \{0, 1\}$. Each Boolean-valued vector of $\{0, 1\}^n$ represents an assignment of Boolean values to the variables of Var_n . We call the set $\{0, 1\}^n$ the *state space* of \mathcal{B} , and the members of $\{0, 1\}^n$ its *states*.

We utilise the widely used *asynchronous semantics* assuming that every network variable can be updated independently of the others. Under this assumption, we construct a directed (asynchronous) state-transition graph $\text{STG}(\mathcal{B}) = (S, T)$, where $S = \{0, 1\}^n$ is the network’s state space, and the transition relation $T \subseteq S \times S$ is defined as $(s, t) \in T \Leftrightarrow s \neq t \wedge \exists i. t = s[i \mapsto F_i(s)]$.

Notice that in the resulting graph, every transition “updates” exactly one network variable. For technical reasons, without the loss of generality, we include a self-loop $(s, s) \in T$ whenever $\forall i. s_i = F_i(s)$. To enable interpretation of HCTL formulae over the dynamics of a BN \mathcal{B} , we take $AP = \{p_1, \dots, p_n\}$ (one proposition per variable) and transform the state-transition graph $\text{STG}(\mathcal{B}) = (S, T)$ directly into a Kripke structure $\mathcal{K}_{\mathcal{B}} = (S, T, L)$ where L is defined by $L(p_i) = \{x \in \{0, 1\}^n \mid x_i = 1\}$.

In systems modelling practice, it is often impossible to know all the update functions exactly. To that end, we adopt the notion of *partially specified Boolean network* (PSBN) [8]. In a PSBN, the modeller can use *function symbols* within each update function definition to describe a model that is only partially known. Formally, let us assume a set \mathcal{F} of function symbols, each with a specified arity. A *partially specified Boolean network* \mathcal{B}_E again assumes n variables Var_n and consists of n *partially specified Boolean expressions* $\{E_1, \dots, E_n\}$, where each E_i is defined by the following grammar:

$$E ::= 0 \mid 1 \mid v \mid \neg E \mid E \wedge E \mid \mathbf{f}^{(a)}(E, \dots, E)$$

where v ranges over Var_n and $\mathbf{f}^{(a)}$ ranges over \mathcal{F} (a denotes the arity of the symbol). An *interpretation* of \mathcal{F} is a function I that assigns to each symbol $\mathbf{f}^{(a)} \in \mathcal{F}$ a Boolean function $f^{(a)} : \{0, 1\}^a \rightarrow \{0, 1\}$. By fixing a particular interpretation I for a given PSBN \mathcal{B}_E , we substitute a concrete Boolean function for each function symbol in its partially specified Boolean expressions, and thus obtain a standard BN, which we denote by $\mathcal{B}_{E(I)}$. We call these BNs the *instances* of PSBNs.

Given a PSBN \mathcal{B}_E and a finite set \mathcal{I} of interpretations, the family of Kripke structures $\{\mathcal{K}_{\mathcal{B}_E(I)} \mid I \in \mathcal{I}\}$ represents the (state-transition) dynamics of \mathcal{B}_E . These Kripke structures share the set of states, but differ in their transition relations. To represent such a family compactly, we consider an extension of Kripke structures, called *coloured Kripke structures*, that allows usage of labels on transitions. The abstract notion of *colours* is used to label the particular “instances” of the transition relation [3]. In case of PSBNs, the set of colours corresponds with the set of function symbol interpretations.

Definition 1 (Coloured Kripke structure (CKS)). *We say that \mathcal{K} is a coloured Kripke structure over AP iff $\mathcal{K} = (\mathcal{C}, S, T, L)$, where \mathcal{C} is a finite set of colours, $T = \{T_c \mid c \in \mathcal{C}, T_c \subseteq S \times S\}$, and for each $c \in \mathcal{C}$ the tuple $\mathcal{K}_c = (S, T_c, L)$ is a Kripke structure.*

Model Classification Problem. Given a PSBN \mathcal{B}_E , a set of function symbol interpretations \mathcal{I} , a set Ψ of closed HCTL formulae representing *required properties* and a (non-empty) set Φ of closed HCTL formulae representing *classified properties*, compute the *classification* mapping $class : 2^\Phi \rightarrow 2^\mathcal{I}$ satisfying for any $X \subseteq \Phi$ the following property:

$$\forall I \in class(X). \mathcal{K}_{\mathcal{B}_E(I)} \models \bigwedge_{x \in X} x \wedge \bigwedge_{y \in \Phi \setminus X} \neg y \wedge \bigwedge_{z \in \Psi} z. \quad (1)$$

It follows from the definition that the classification mapping induces an equivalence relation on \mathcal{I} that associates all interpretations that satisfy the same subset of classification properties. The resulting quotient set of \mathcal{I} thus clusters BN models based on Φ . Since every class is unique for some particular set of formulae $X \subseteq \Phi$, we use X to denote the corresponding class.

It is apparent that the number of classes is exponential in $|\Phi|$. On the other hand, the number of possible interpretations can be enormous and simultaneously high-dimensional (tens or even hundreds of function symbols). To make the classification practical, we utilise the notion of decision trees [7] that allow us to visualise custom scenarios showing the user how the particular settings of individual function symbols lead to desired classes. The tree consists of two types of nodes. A *leaf node* represents interpretations exhibiting the same properties combination (a particular class $X \subseteq \Phi$). A *decision node* represents a choice on the concrete interpretation of a particular function symbol. It has an outgoing positive and negative edges, which lead to the sub-trees where the corresponding interpretations evaluate that particular function to true or false, respectively (an example of this structure is given in Section 6).

4 Model Classification Algorithm

The operation of the tool consists of two phases. In the first phase, we run the model classification algorithm, described in this section, to obtain the classification mapping as explained above. In the second phase, we build the decision tree in

Table 1. Extended state space operations. ($V \subseteq S \times C \times S^{Var_\varphi}$)

existential quantification of the state	$\exists_{st}(V)$	$\{(s', c, \rho) \mid s' \in S \wedge \exists s : (s, c, \rho) \in V\}$
existential quantification of a state variable	$\exists_x(V)$	$\{(s, c, \rho') \mid \rho' \in S^{Var_\varphi} \wedge \exists \rho : (s, c, \rho) \in V$ $\wedge \forall y \in Var_\varphi \setminus \{x\} : \rho'(y) = \rho(y)\}$
equaliser of the state and a state variable	$EQ(x)$	$\{(\rho(x), c, \rho) \mid c \in C \wedge \rho \in S^{Var_\varphi}\}$
coloured pre-image	$PRE(V)$	$\{(s, c, \rho) \mid \exists t : (t, c, \rho) \in V \wedge (s, t) \in T_c\}$

an interactive manner, allowing the user to choose which tree nodes to expand. The procedure behind this second phase is explained in more detail in [7].

The model classification algorithm proceeds by repeatedly calling a subroutine which we call the *coloured HCTL model checking*. This subroutine gets a symbolic description of a CKS \mathcal{K} and a closed HCTL formula φ and returns the set of colours c such that $\mathcal{K}_c \models \varphi$.

In the first run, the CKS is given by the semantics of the input PSBN and the formula is a conjunction of all the *required* properties. We call the resulting set of colours the *permissible colours*. After this first run, we create a new CKS by restricting the original CKS only to the permissible colours. We then run the subroutine in this new CKS several times, once for each *classified* property. Once we get all the resulting sets of colours, we combine them using intersections and complements to obtain the classification mapping.

The rest of this section is used to explain the coloured HCTL model checking subroutine. In the following, we assume a fixed CKS $\mathcal{K} = (C, S, T, L)$ and a fixed closed HCTL formula φ . We use Var_φ to denote the (finite) set of state variables that appear in the formula φ . We use the notation S^{Var_φ} for the set of all functions $Var_\varphi \rightarrow S$, i.e. the set of all assignments of states to the variables of Var_φ . Because we need to reason about both the colours and the valuation of the state variables, we work with the set $S \times C \times S^{Var_\varphi}$, which can also be seen as a relation of arity $(|Var_\varphi| + 2)$. We call this set the *extended state space* and its elements *extended states*. We assume that this set and its arbitrary subsets can be represented symbolically. In the algorithm we use standard set operations (union, intersection) together with specific operations on sets of extended states that are described in Table 1. A note about how the symbolic representation and the operations are implemented using BDDs is provided below.

Intuitively, the role of the *existential quantification* operations is to “forget” certain information about the current subset of the extended state space: \exists_{st} forgets the current state while \exists_x forgets the valuation of the state variable x . The *equaliser* of x , $EQ(x)$ is then a subset that holds a single piece of information, namely that the valuation of the state variable x should be equal to the current state. Finally, the most important operation is the *coloured pre-image*, $PRE(V)$. Given a subset of the extended state space, this operation computes the set of all predecessors respecting the colours of the transitions while keeping the valuation of the state variables intact.

The pseudocode of the algorithm is given as Algorithm 1. As in standard symbolic CTL model checking, we assume that the formula is given in a normal form that only uses the temporal operators **EX**, **EG**, and **EU**.

The algorithm begins by computing the set Var_φ of variables that appear in the given formula. Once the set Var_φ is computed, it is considered a global constant during the rest of the computation, together with the input CKS. The rest of the algorithm proceeds by recursively calling the **CHECK** function. This function accepts a sub-formula of the main formula as its input and produces a set of extended states (s, c, ρ) such that the state s satisfies the given sub-formula in the Kripke structure \mathcal{K}_c with ρ as the state variable valuation. For efficient computation, the function is supposed to be memoized, i.e. its results should be cached and used whenever the same sub-formula is to be processed.

The computation of the **CHECK** function is mostly straightforward. The algorithm for **EU** comes from the fact that $\mathbf{E}[\psi_1 \mathbf{U} \psi_2] \equiv \psi_2 \vee (\psi_1 \wedge \mathbf{EXE}[\psi_1 \mathbf{U} \psi_2])$ and that the **EU** operator is the least fixed point of this equation. Similarly, the algorithm for **EG** comes from the fact that $\mathbf{EG} \psi \equiv \psi \wedge \mathbf{EXEG} \psi$ and that the **EG** operator is the greatest fixed point of this equation. The proof of these claims in HCTL is the same as for standard CTL.

As for the hybrid extension, to evaluate the sub-formula x where x is a state variable, we simply take the equaliser of x , as this sub-formula is true in exactly all the extended states where the valuation of x is the current state. To evaluate the *jump* operator $@_x.\psi$, we first compute all the extended states where ψ holds. Out of them, we filter only those extended states where the valuation of x is the current state. And finally, we “forget” the current state as the satisfaction of $@_x.\psi$ is independent of it. To evaluate the existential quantification $\exists x.\psi$, we again first compute **CHECK**(ψ). Then, we simply “forget” the valuation of x as x is no longer a free variable in $\exists x.\psi$. Finally, to evaluate the *bind* operation $\downarrow x.\psi$, we do a computation similar to the previous case, but we filter only those states where the valuation of x is the current state.

Once the **CHECK** function computes the set of extended states for the whole original formula φ , the algorithm discards the state variable valuation part of the set and returns the result. Note that as the whole formula is closed, the valuation plays no role in the final result.

To symbolically represent the CKS, we use Reduced Ordered Binary Decision Diagrams (ROBDDs, or simply BDDs) [10], which can concisely encode Boolean functions or relations of Boolean vectors. As part of [8], the authors describe how the coloured dynamics of every PSBN can be described using only Boolean variables. To verify HCTL formulae, we also need to incorporate the possible assignments of state variables in our BDD representation (subsets of S^{Var_φ}). To encode this component, we employ $n \cdot |Var_\varphi|$ Boolean variables, such that for each $x \in Var_\varphi$, a sequence of variables x_1, \dots, x_n encodes mappings for this particular state variable in the BDD. If we combine this encoding with the encoding of partially specified Boolean networks from [8], we can represent subsets of the $S \times \mathcal{C} \times S^{Var_\varphi}$ relation that are used throughout Algorithm 1. Finally, to implement the extended state space operations defined in Table 1,

Algorithm 1: Coloured symbolic model checking for HCTL

Input: $\mathcal{K} = (\mathcal{C}, S, T, L)$ is a coloured Kripke structure; φ is a HCTL formula
Output: the set $\{c \in \mathcal{C} \mid \mathcal{K}_c \models \varphi\}$

- 1 $Var_\varphi \leftarrow \{x \in Var \mid x \text{ appears in } \varphi\}$
- 2 $V \leftarrow \text{CHECK}(\varphi)$
- 3 **return** $\{c \mid \exists \rho : \forall s : (s, c, \rho) \in V\}$

4 **Function** $\text{CHECK}(\psi)$ (*memoized*)

- 5 **switch** ψ **do**
- 6 **case** $p \in AP$ **do return** $\{(s, c, \rho) \in S \times \mathcal{C} \times S^{Var_\varphi} \mid s \in L(p)\}$
- 7 **case** $\neg\psi'$ **do return** $S \times \mathcal{C} \times S^{Var_\varphi} \setminus \text{CHECK}(\psi')$
- 8 **case** $\psi'_1 \wedge \psi'_2$ **do return** $\text{CHECK}(\psi'_1) \cap \text{CHECK}(\psi'_2)$
- 9 **case** $\text{EX } \psi'$ **do return** $\text{PRE}(\text{CHECK}(\psi'))$
- 10 **case** $\text{E}[\psi'_1 \text{ U } \psi'_2]$ **do**
- 11 $B \leftarrow \text{CHECK}(\psi'_2)$; $Set_{\psi'_1} \leftarrow \text{CHECK}(\psi'_1)$
- 12 **repeat** $B \leftarrow B \cup (Set_{\psi'_1} \cap \text{PRE}(B))$ **until fixpoint**
- 13 **return** B
- 14 **case** $\text{EG } \psi'$ **do**
- 15 $B \leftarrow \text{CHECK}(\psi')$
- 16 **repeat** $B \leftarrow B \cap \text{PRE}(B)$ **until fixpoint**
- 17 **return** B
- 18 **case** $x \in Var_\varphi$ **do return** $\text{EQ}(x)$
- 19 **case** $\downarrow x.\psi'$ **do return** $\exists_x(\text{EQ}(x) \cap \text{CHECK}(\psi'))$
- 20 **case** $@x.\psi'$ **do return** $\exists_{st}(\text{EQ}(x) \cap \text{CHECK}(\psi'))$
- 21 **case** $\exists x.\psi'$ **do return** $\exists_x(\text{CHECK}(\psi'))$

consider the following: (i) The \exists_{st} operation can be implemented as existential quantification (in BDDs sometimes also called projection) over the variables s_1, \dots, s_n which describe the states of the Boolean network. (ii) Similarly, the \exists_x operation can be implemented as a projection over the variables x_1, \dots, x_n which correspond to the state variable x in our encoding. (iii) The $\text{EQ}(x)$ set is described by a formula $(s_1 \Leftrightarrow x_1) \wedge \dots \wedge (s_n \Leftrightarrow x_n)$, which equates the BN variables with the desired state variable x .

5 Tool components and implementation

BNCLASSIFIER consists of two main components (see Fig. 2): The model checking and *classification engine* (available as a Rust or a Python library) and an interactive *decision tree explorer* (a multiplatform desktop application).

All methods concerned with symbolic computation are written in Rust to ensure safety and efficiency. These core symbolic algorithms are then also available as a Python library (*classification engine*), or through a graphical user interface (*decision tree explorer*). This separation aids in interoperability and enables the user to run the main computation remotely on high-performance hardware.

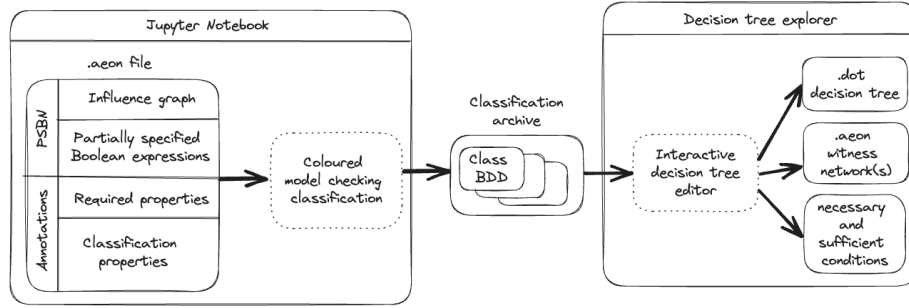


Fig. 2. Visual overview of the BNCLASSIFIER components and functionality. The left part represents the *classification engine*, typically used through its Python API in Jupyter notebooks. Right part represents the *decision tree explorer* which is used to interactively explore the classification result.

Classification engine. The functionality of the classification engine is summarised by the following categories:

- Input specification: The classification problem is specified in the `.aeon` file format, originally introduced to compactly describe PSBNs [5]. Later, the format was extended with general purpose annotations, which we use to specify the required and classification HCTL properties. The actual HCTL properties are written using a custom textual representation.
- Coloured model checking: The *classification engine* implements the coloured model checking algorithm based on Section 4, extended with several optimisations: First, every HCTL formula is transformed into a canonical form, so that the computation of any equivalent sub-formulae can be cached and reused. Second, the *saturation* technique [11] for asynchronous systems is used to speed up the **EU** and **EF** fixed-point computation. Finally, the engine also recognises certain common formula patterns which are replaced with equivalent results of faster, domain specific symbolic algorithms.
- Output specification: The result of the classification (the *class map*) consists of several BDDs, each representing the set of interpretations for one of the non-empty classes (combinations of valid properties). Together with the annotated `.aeon` file and a textual summary of the results, these are exported into a single archive that is then consumed by the *decision tree explorer*.
- Python bindings: The functionality of the *classification engine* is available both as a stand-alone Rust library as well as a Python library that can be integrated with Jupyter notebooks and other similar environments. This ensures interoperability within the larger ecosystem of systems biology tools, such as the ones maintained by the CoLoMoTo consortium [26].

Decision tree explorer. Architecturally, the *decision tree explorer* consists of a Rust backend and a HTML/JS frontend that are packaged together as a single multi-platform desktop application using the framework Tauri [1]. This

architecture maintains the flexibility and interoperability of a typical web application, but also allows us to perform any non-trivial computations locally within the application, without requiring a server or other distributed infrastructure.

The main element of the application is an interactive decision tree editor. Here, the user can partition the classification map based on various attributes of the PSBN interpretations into a decision tree. Such decision tree helps to explain the relationships between classes and enables user-guided search for necessary and sufficient conditions leading to the emergence of a particular class.

The explorer can also automatically generate a decision tree which is optimal with respect to the *information gain*, a standard metric in machine learning [30]. The user can then export the resulting tree for use in other materials as a standard `.dot` graph. Finally, the user can randomly sample the fully specified witness networks (as `.aeon` files) for any fully expanded tree branch representing a set of conditions sufficient to achieve a particular behaviour class.

6 Evaluation

Finally, this section demonstrates the applicability and performance of BNCLASSIFIER. First, we present a biological case study using a PSBN model of cancer cell fate to demonstrate the complete workflow. Then, we present an extensive evaluation of the *classification engine* on a large dataset of real-world Boolean models. Finally, we compare the performance of the coloured model checking with the naïve “parameter scan” method. The technical details and reproducibility instructions for each experiment are available in the tool artefact.

6.1 Analysis of MAPK model

To demonstrate the tool’s applicability, we consider a well-known model [17] of one of the most important signalling mechanisms in human cells – the MAPK pathway. The model investigates the influence of several extracellular stimuli on cancer cell fate decisions. It consists of 18 variables. Four of these variables have update functions characterized by function symbols with zero arity. For simplicity, we refer to such variables as *inputs*. They represent the extracellular stimuli (`TGFBR_stimulus`, `EGFR_stimulus`, `FGFR3_stimulus`, and `DNA_damage`). All possible settings of these inputs result in 16 different BN instances. Another three variables represent possible modes of the cell behaviour: `Apoptosis` (programmed cell death), `Growth_Arrest` (temporary stop of growth), and `Proliferation` (cell reproduction).

We are especially interested in the relationship among the individual settings of the inputs and the long-term behaviour of the model (attractors). Moreover, we investigate the possible presence of biologically unrealistic behaviour to assess the overall soundness of the model. With these goals in mind, we define the set of required properties $\Psi = \{\psi\}$ and the set of classification properties $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$:

$$\psi = \exists x. @_x ((\mathbf{AG\ EF} \ x) \wedge (\mathbf{Apoptosis} \vee \mathbf{Growth_Arrest} \vee \mathbf{Proliferation}))$$

There exists an attractor state where some cell mode is active.

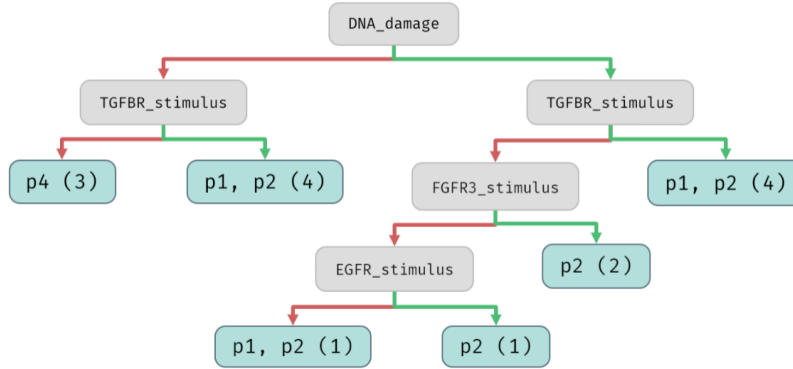


Fig. 3. Decision tree of the MAPK model produced by BNCLASSIFIER. It shows the effect of input variables on the classification properties. Each decision node contains the name of a variable used for the decision. Leaf node labels show the number of instances represented by the node (in brackets), plus the list of properties satisfied by these interpretations (with $p_i = \varphi_i$).

$$\varphi_1 = \forall x. @_x ((\mathbf{AG} \mathbf{EF} x) \Rightarrow (\mathbf{Apoptosis} \wedge \mathbf{AX} x))$$

Every attractor state is a fixed point with **Apoptosis** active (the system always converges to programmed cell death; *healthy* behaviour).

$$\varphi_2 = \exists x. @_x (\mathbf{AG} \mathbf{Growth_Arrest}), \varphi_3 = \exists x. @_x (\mathbf{AG} \mathbf{Proliferation})$$

Growth_Arrest (φ_2) or **Proliferation** (φ_3) remain active in some attractor (programmed cell death may not be achieved; possibly *problematic* behaviour).

$$\varphi_4 = \exists x. @_x ((\mathbf{AG} \mathbf{EF} x) \wedge \mathbf{Apoptosis} \wedge \mathbf{Proliferation})$$

Both **Proliferation** and **Apoptosis** are simultaneously active in an attractor state (biologically *unrealistic* behaviour).

We obtained classification results in less than two minutes, revealing several interesting model characteristics: (i) One instance (with all stimuli turned off) does not satisfy the required property ψ . In this case, the model does not manifest any known biological cell mode. (ii) Nine instances satisfy φ_1 , i.e. the cell is guaranteed to undergo proper programmed death. Furthermore, all these instances also satisfy φ_2 (impeded cell growth). (iii) There is no instance where φ_3 holds (**Proliferation** cannot remain active indefinitely). However, (iv) three instances admit intermittent simultaneous activity of both **Apoptosis** and **Proliferation** (prop. φ_4).

Although we found that φ_4 holds for some instances, it does not necessarily make the model unsound. Biologically unrealistic conditions can be admissible, but it is important to *understand* when and how they occur to interpret the model correctly. To do that, we construct a decision tree (Fig. 3) where we explore the conditions leading to the validity of individual properties. In particular, we see that the biologically unrealistic property φ_4 occurs if and only if both **DNA_damage** and **TGFBR_stimulus** are inactive. The decision tree also guides us towards conditions that ensure proper programmed cell death. For example, we see that the presence of **TGFBR_stimulus** is a sufficient condition for φ_1 .

6.2 Performance evaluation

To demonstrate that our tool can handle realistic BN models, we evaluate it on a set of 230 biological models available in the BBM repository [29]. This benchmark set includes models available through the GINSim [27], Cell Collective [18], and Biomodels [24] databases, as well as the COVID-19 disease map project [28]. The largest number of variables in a single model is 341, and the maximum number of regulations is 1100.

In this section, we consider only fully specified models (i.e. without function symbols). In Section 6.3, we further discuss the benefits of our approach with partially specified models. We have selected four biologically motivated properties and encoded them as HCTL formulae. They cover relevant long-term behaviour patterns, such as various modes of stability or oscillations. These patterns play an important role in analysing biological models [2, 5].

$$\varphi_1 = \exists x. \forall y. (@_y. \mathbf{EF}(x \wedge (\downarrow z. \mathbf{AX} z)))$$

There exists a steady state reachable from all states.

$$\varphi_2 = \exists x. \exists y. (@_x. (\neg y \wedge \mathbf{AX} x)) \wedge (@_y. \mathbf{AX} y)$$

The model admits multiple terminal steady states.

$$\varphi_3 = \exists x. \exists y. @_x. (\mathbf{EF}(y \wedge (\downarrow z. \mathbf{AX} z)) \wedge \mathbf{EF}(\neg y \wedge (\downarrow z. \mathbf{AX} z)))$$

There are states from which multiple steady states are reachable.

$$\varphi_4 = \exists x. (\mathbf{AX}(\neg x \wedge \mathbf{AF} x))$$

The model admits periodically visited “checkpoint” states (states without a self-loop, with all outgoing paths inevitably leading back to the state itself).

The crucial part of our computation engine is the model-checking sub-procedure. Therefore, in addition to evaluating the overall classification process, we also focus on evaluating the model-checking procedure. For each model, we run one classification experiment (with the set of classification properties $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$), and four model-checking experiments (one for each formula). This allows us to individually measure the times for each model-checking computation and assess it better.

We performed all experiments on a workstation with a 32-core AMD Threadripper 2990WX CPU and 64GB of RAM. Every experiment was executed in a separate process, and its runtime was measured using the standard UNIX time utility. Due to the high number of considered experiments, we typically executed up to 16 benchmarks in parallel to speed up the evaluation. This is because, in our experience, the memory bandwidth of this particular CPU does not scale sufficiently to all 32 cores for the considered workloads (symbolic algorithms typically perform a lot of sequential random memory accesses, resulting in contention among cores). For each benchmark experiment, we consider a timeout of one hour.

Fig. 4 displays an overall cumulative graph of benchmarks for each experiment completed before a certain time. We were able to individually model check φ_1 and φ_3 on 217 models, φ_1 on 203 models, and φ_4 on 141 models. Note that we were able to evaluate $\varphi_1, \varphi_2, \varphi_3$ on more than 150 models in less than a second. Evaluating φ_4 proved most demanding due to its subformula $\mathbf{AF} x$, where we must (symbolically) evaluate \mathbf{AF} for the whole state space. When running the entire classification process, we obtained results for 140 models.

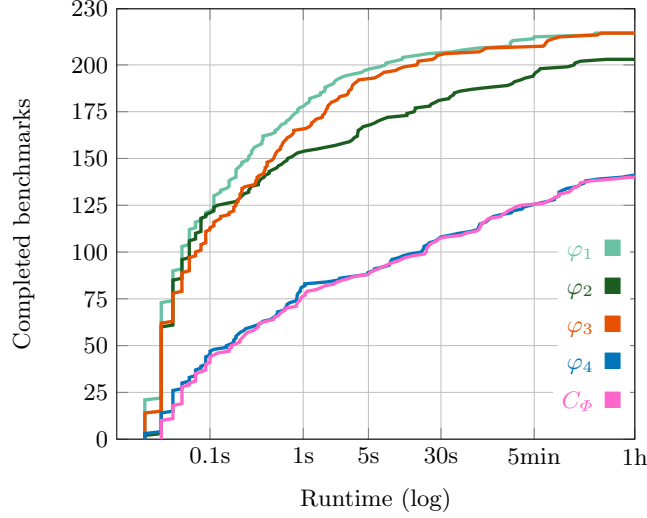


Fig. 4. The number of benchmarks (y-axis) completed for each set of experiments before a specific time (x-axis). Results of the overall classification experiments are shown as C_Φ , and model-checking experiments for individual formulae as φ_i . Note the times are logarithmic.

6.3 Comparison with Parameter Scanning

One of the advantages of our approach is that we symbolically represent the dynamics of all possible instances of a given PSBN. This allows us to perform a single symbolic computation for all instances at once. The other option would be to generate all the individual instances and evaluate them one by one, the so-called *parameter scan*. This section demonstrates that our approach provides a significant speed-up in comparison with the parameter scan.

We have selected seven PSBN models with at least 50 variables from the BBM repository [29]. The only uncertainty in all these models is in their inputs, which is a typical situation in systems biology. All the considered models admit more than 10 inputs (i.e., more than 2^{10} interpretations). We discuss other models with higher-arity function symbols at the end of this section. Within the repository, each model is assigned a canonical numeric identifier. We refer to particular benchmark models using these identifiers. The models are listed in Table 2. Among this benchmark set, the largest number of non-input variables is 129, and the maximum number of inputs is 28. We only focus on evaluating the model-checking procedure. In each experiment, we measure the time to model check the formula φ_2 (introduced in Section 6.2).

Since we were not able to complete a full parameter scan for each model (due to the high number of inputs), we randomly sampled $2^{10} = 1024$ unique instances for each model. We ran the model checking for these instances one by one, and then, based on the observed values, we made a conservative estimate

for the runtime of the full parameter scan. For our estimate, we consider the runtime of the fastest encountered instance and multiply it by the total number of model instances. Consequently, the runtime of a full parameter scan would most likely have been significantly longer because not all instances can be processed in this optimistic time frame. Also note that the runtime estimate involves a little overhead (due to, e.g., input loading), which is usually negligible compared with the model-checking computation times.

ID	Name	n	m	CMC	Scan
018	EGFR-ERBB Signalling	76	28	1513.10	3.76e9
019	IL-6 Signalling	71	15	12.06	3.90e5
056	IGVH Mutations in Leukemia	66	25	1081.15	3.04e9
077	Butanol Production Pathway	53	13	0.22	983.04
132	Virus Replication Cycle	129	19	61.79	5.15e6
137	Signalling in Liver Cancer	71	11	163.18	5.80e4
207	Breast Cancer Tumour	85	18	228.63	5.75e7

Table 2. Benchmark results comparing the symbolic approach with the parameter scan. Each experimented model is described by its ID, name, number of non-input variables (n), and inputs (m). The column ‘CMC’ shows the runtime (in seconds) with all instances processed using the coloured model-checking approach. The column ‘Scan’ shows, in seconds, the conservative estimate of the runtime for the entire parameter scan (i.e., the runtime of the fastest instance multiplied by the total number of instances).

Due to the high number of experiments, we only show a summary in Table 2. In every case, the coloured model-checking approach is at least several orders of magnitude better than our most conservative estimate of the parameter scan. The speed-up is between 4400 times (model 077) and more than 2.4×10^6 times (model 018). In general, the speed-up increases with the number of model inputs (since the number of instances is exponential to the number of inputs). For models 018 and 056, the full parameter scan is (conservatively) estimated to run for approximately 100 years.

Finally, we have also experimented with PSBN models that involve update functions with function symbols of higher arity. To that end, we chose five models from the BBM database with number of instances ranging from 2^{50} to 2^{100} . We were able to run the classification with the set of all four properties $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ (introduced in Section 6.2) on all these models in a matter of minutes. Evaluating such benchmarks through the parameter scan would clearly be infeasible.

References

1. Tauri apps. <https://tauri.app/>, accessed: 2023-09-30
2. Arellano, G., Argil, J., Azpeitia, E., Benítez, M., Carrillo, M., Góngora, P., Rosenblueth, D.A., Alvarez-Buylla, E.R.: “Antelope”: a hybrid-logic model checker for branching-time boolean grn analysis. *BMC Bioinformatics* **12**(1), 490 (2011)

3. Barnat, J., Brim, L., Krejci, A., Streck, A., Safranek, D., Vejnar, M., Vejpustek, T.: On parameter synthesis by parallel model checking. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **9**(3), 693–705 (2012)
4. Bartocci, E., Lio, P.: Computational modeling, formal analysis, and tools for systems biology. *PLOS Computational Biology* **12**(1), 1–22 (01 2016)
5. Beneš, N., Brim, L., Pastva, S., Šafránek, D.: AEON: Attractor bifurcation analysis of parametrised boolean networks. In: *Computer Aided Verification - 32nd International Conference, CAV 2020. Lecture Notes in Computer Science*, vol. 12224. Springer International Publishing, Cham (2020)
6. Beneš, N., Brim, L., Pastva, S., Šafránek, D.: Aeon 2021: Bifurcation decision trees in boolean networks. In: *Computational Methods in Systems Biology*. pp. 230–237. Springer (2021)
7. Beneš, N., Brim, L., Pastva, S., Poláček, J., Šafránek, D.: Formal analysis of qualitative long-term behaviour in parametrised boolean networks. In: *Formal Methods and Software Engineering (ICFEM 2019). Lecture Notes in Computer Science*, vol. 11852, pp. 353–369. Springer (2019)
8. Beneš, N., Brim, L., Pastva, S., Šafránek, D.: BDD-based algorithm for SCC decomposition of edge-coloured graphs. *Log. Methods Comput. Sci.* **18**(1) (2022). [https://doi.org/10.46298/lmcs-18\(1:38\)2022](https://doi.org/10.46298/lmcs-18(1:38)2022)
9. Bloomingdale, P., Nguyen, V.A., Niu, J., Mager, D.E.: Boolean network modeling in systems pharmacology. *Journal of pharmacokinetics and pharmacodynamics* **45**(1), 159–180 (2018)
10. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* **35**(8), 677–691 (1986)
11. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Saturation: An efficient iteration strategy for symbolic state—space generation. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001). Lecture Notes in Computer Science*, vol. 2031, pp. 328–342. Springer (2001)
12. Fisher, J., Henzinger, T.A.: Executable cell biology. *Nature Biotechnology* **25**(11), 1239–1249 (2007)
13. Franceschet, M., de Rijke, M.: Model checking hybrid logics (with an application to semistructured data). *Journal of Applied Logic* **4**(3), 279–304 (2006)
14. Goranko, V.: Temporal logic with reference pointers. In: *Temporal Logic, First International Conference, ICTL '94, Bonn, Germany, July 11-14, 1994, Proceedings. Lecture Notes in Computer Science*, vol. 827, pp. 133–148. Springer (1994)
15. Goranko, V.: Temporal logics with reference pointers and computation tree logics. *Journal of Applied Non-Classical Logics* **10**(3-4), 221–242 (2000)
16. Goron, A., Chesñévar, C.I.: [hlmc: A hybrid logic tool for model checking in verification of administrative processes]. In: *Proceedings of the 9th International Conference on Theory and Practice of Electronic Governance. p. 376–377. ICEGOV '15-16, Association for Computing Machinery (2016)*. <https://doi.org/10.1145/2910019.2910046>
17. Grieco, L., Calzone, L., Bernard-Pierrot, I., Radvanyi, F., Kahn-Perlès, B., Thieffry, D.: Integrative modelling of the influence of mapk network on cancer cell fate decision. *PLOS Computational Biology* **9**(10), 1–15 (10 2013). <https://doi.org/10.1371/journal.pcbi.1003286>
18. Helikar, T., et al.: The cell collective: Toward an open and collaborative approach to systems biology. *BMC Systems Biology* **6**(1), 96 (2012)
19. Henzinger, T.A., Otop, J.: From model checking to model measuring. In: D’Argenio, P.R., Melgratti, H. (eds.) *CONCUR 2013 – Concurrency Theory*. pp. 273–287. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

20. Henzinger, T.A., Otop, J.: Model measuring for discrete and hybrid systems. *Nonlinear Analysis: Hybrid Systems* **23**, 166–190 (2017)
21. Kauffman, S.: Homeostasis and differentiation in random genetic control networks. *Nature* **224**(5215), 177–178 (1969)
22. Kernberger, D., Lange, M.: Model checking for hybrid branching-time logics. *Journal of Logical and Algebraic Methods in Programming* **110**, 100427 (2020)
23. Kernberger, D., Lange, M.: On the expressive power of hybrid branching-time logics. *Theoretical Computer Science* **813**, 362–374 (2020)
24. Malik-Sheriff, R.S., et al.: BioModels — 15 years of sharing computational models in life science. *Nucleic Acids Research* **48**(D1), 407–415 (2019)
25. Mori, T., Akutsu, T.: Attractor detection and enumeration algorithms for boolean networks. *Computational and Structural Biotechnology Journal* **20**, 2512–2520 (2022)
26. Naldi, A., Hernandez, C., Levy, N., Stoll, G., Monteiro, P.T., Chaouiya, C., Hellikar, T., Zinovyev, A., Calzone, L., Cohen-Boulakia, S., Thieffry, D., Paulevé, L.: The colomoto interactive notebook: Accessible and reproducible computational analyses for qualitative biological networks. *Frontiers in Physiology* **9** (2018). <https://doi.org/10.3389/fphys.2018.00680>
27. Naldi, A., et al.: Logical modelling of regulatory networks with ginsim 2.3. *Biosystems* **97**(2), 134–139 (2009)
28. Ostaszewski, M., et al.: COVID19 disease map, a computational knowledge repository of virus–host interaction mechanisms. *Molecular Systems Biology* **17**(10) (2021)
29. Pastva, S., Šafránek, D., Beneš, N., Brim, L., Henzinger, T.: Repository of logically consistent real-world boolean network models. *bioRxiv* (2023). <https://doi.org/10.1101/2023.06.12.544361>
30. Quinlan, J.R.: Induction of decision trees. *Machine learning* **1**, 81–106 (1986)
31. Søgaaard, A., Kristiansen, S.L.: Using hybrid logic for querying dependency treebanks. *Linguistic Issues in Language Technology* **7** (2012). <https://doi.org/10.33011/lilt.v7i.1267>
32. Thomas, R.: Boolean formalization of genetic control circuits. *Journal of Theoretical Biology* **42**(3), 563–585 (1973)
33. Wang, R.S., Saadatpour, A., Albert, R.: Boolean modeling in systems biology: an overview of methodology and applications. *Physical biology* **9**(5), 055001 (2012)