From the Archives of the Formal Methods and Tools Lab
Axiomatising and Contextualising ACTL

Stefania Gnesi(✉)[0000−0002−0139−0421] and
Maurice H. ter Beek[0000−0002−2930−6367]

Formal Methods and Tools Lab
ISTI–CNR, Pisa, Italy
{stefania.gnesi,maurice.terbeek}@isti.cnr.it

Abstract. We present a sound and complete axiomatisation of ACTL, an action-based version of the well-known branching-time temporal logic CTL, and place it into a historical context. ACTL was originally introduced by Rocco De Nicola together with Frits Vaandrager 30 years ago, and it has played a major role in shaping the activity of our Formal Methods and Tools Lab from the nineties to this very day.

Keywords: Temporal logic · ACTL · Axiomatisation.

1 Introduction

To appreciate the contribution of this paper, we first provide some necessary context through a brief recollection of memories from the last 40 years.

1.1 Rocco & Stefania

Rocco and Stefania were fellow students in Computer Science at the University of Pisa. They followed the same classes and had the same thesis supervisor. In fact, both were advised by Ugo Montanari and both graduated in 1978. After that, they followed a different road for some time. Their paths crossed again in 1984 when Stefania started working at the CNR, in the *Istituto di Elaborazione dell'Informazione* (later incorporated in what is nowadays called the *Istituto di Scienza e Tecnologie dell'Informazione* (ISTI)), where Rocco had been employed a couple of years earlier. Those were the years of the birth of formal verification techniques and tools. Temporal logics [15,11,16,12,47,27,28,26,53] were very much *à la mode* at that time and the first automatic tools for the verification of concurrent systems, mostly model checkers [15,48,17,18,33], were being realised.

The late eighties was the time during which Rocco and Frits Vaandrager worked on the definition of the so-called *action-based* branching-time temporal logics, namely ACTL and its extended version ACTL* [23,24,25]. Such temporal logics are highly suitable to express properties of concurrent systems specified by means of process algebras.

1.2 Action-Based Temporal Logics

Process algebras [43,44,1,51,2] are generally recognised as being a convenient means for describing concurrent systems at different levels of abstraction. Their basic operational semantics is usually defined in terms of Labelled Transition Systems (LTSs), which are then quotiented by means of observational equivalences and allow the behaviour of a system to be analysed in relation to the *actions* the specified system may perform.

Specific logics for process algebras were proposed (cf., e.g., [42,53]), typically interpreted on LTSs, and ACTL and ACTL* were defined in this framework as the action-based counterparts of CTL and CTL* [15,16,27,28]. In another Festschrift contribution [6], we provided a more detailed historical account of temporal logics for reasoning on state-based as well as action-based properties and their interpretation structures, typically variants of the Doubly-Labelled Transition Systems (L^{2}TS) introduced by Rocco and Frits Vaandrager in [24,25].

1.3 Model Checking Action-Based Temporal Logics

Model-checking techniques [17,18] were defined to verify system properties, expressed as temporal logic formulae, on finite-state models of the behaviour of systems. Once a model of a system has been generated, the properties are automatically verified by model-checking tools.

An efficient model checker, called AMC, was defined for ACTL to verify the satisfaction of ACTL formulae over states in an LTSs as a collaboration between Stefania, Rocco and other colleagues from Pisa and was first presented at CAV'91 [20,21]. The model checker AMC was later integrated in the JACK verification environment [14], whose extended version also contains a symbolic model checker for ACTL, called SAM [36], and they were successfully used to verify properties expressed as ACTL formulae on several concurrent systems, among which some interesting industrial case studies [19,13,36,38].

Following this initial experience and to better deal with the so-called *state-space explosion problem* which is typical of explicit-state model checkers, the on-the-fly model checker FMC [41] was developed by Franco Mazzanti, another Formal Methods and Tools Lab member. This tool formed the basis on which a family of model checkers, named KandISTI, has been developed at ISTI–CNR for over two decades now; that family now includes besides FMC, the UML model checker UMC [4], the model checker CMC for verifying specifications in the Calculus for Orchestration of Web Services (COWS) [35] and—the most recent member of the family—the variability model checker VMC [9,10]. Each tool allows for the efficient verification, by means of explicit-state on-the-fly model checking of a family of logics based on ACTL. The KandISTI model checkers, available online at http://fmt.isti.cnr.it/kandisti/, allow for model checking with a complexity that is linear with respect to the size of the model and the size of the formula, when ignoring the fixed point operators and the parametric aspects of the logics (in which cases the complexity depends on the number of nested fixed point operators and the number of instantiations of parametric subformulae).

In yet another Festschrift contribution [7], we described the development of the KandISTI family of model checkers from its origins.

1.4 The ERCIM Workshop on Theory and Practice in Verification

The eighties and nineties of the last century saw the birth of numerous events concerning the formal verification of systems and protocols. The IFIP WG6.1 established the series of symposia on Protocol Specification, Testing and Verification (PSTV) and conferences on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE). In 1989, exactly 30 years ago, the conference series on Computer Aided Verification (CAV) began as a workshop on Automatic Verification Methods for Finite State Systems, including a contribution by Rocco and some of his colleagues from Pisa [22]. Subsequently, a steering committee composed of Edmund Clarke, Robert Kurshan, Amir Pnueli and Joseph Sifakis decided that CAV was to be organised annually. Started as a workshop, nowadays it is the premier international conference on formal verification. Together with colleagues from Pisa, Rocco and Stefania contributed to CAV'91 with a paper that shows how to model check ACTL formulae [20].

A year later, in 1992, Stefania and colleagues from Pisa organised a workshop on the Theory and Practice in Verification at the CNR, in the context of the European Research Consortium for Informatics and Mathematics (ERCIM), founded in 1989 to foster collaborative work within the European ICT research community and to increase co-operation with European industry, which CNR joined in 1992. This workshop was the first 'European' meeting in the context of formal verification and Rocco was among the participants, together with many of the main actors in verification in the context of process algebras (cf. Figs. 1–2).

1.5 The Formal Methods and Tools Lab of ISTI–CNR

From that moment, formal verification became one of the main research fields for us and we may say that this particular event, as well as the collaborations with Rocco on ACTL, have played a major role in shaping the activity of the Formal Methods and Tools Lab of ISTI–CNR from the nineties to this very day.

Since Maurice joined the lab in the beginning of this century, he has worked together with Rocco and Stefania in a number of European and national projects, most notably the FP6-IP-IST-016004 project SENSORIA (Software Engineering for Service-Oriented Overlay Computers) and the FP7-FET-ICT-600708 project QUANTICOL (A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours), as well as the MIUR–PRIN 2010LHT4KM project CINA (Compositionality, Interaction, Negotiation, Autonomicity for the future ICT society) and the most recently approved MIUR–PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems), both coordinated by Rocco. During this period, he has become acquainted with ACTL and he has helped to promote the model checkers from the KandISTI family [8,9,10,5,7], which are characterised by their logics based on ACTL that allow for the verification of action-based as well as state-based properties [3,4,6].

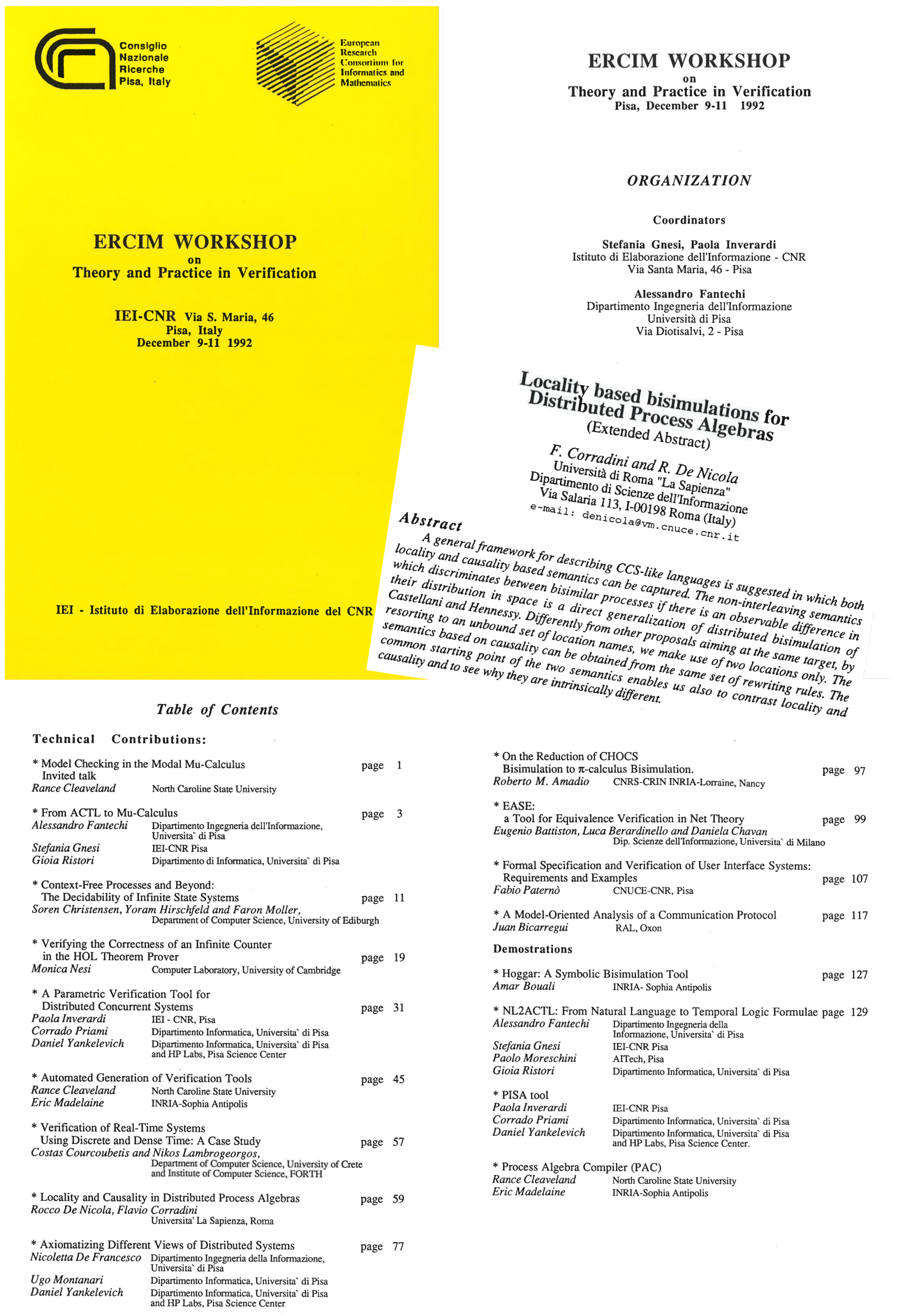
1.6 Contribution: An Axiomatisation of ACTL

An alternative to the model-theoretic approach to verification is the proof-theoretic one, according to which a system is modelled in terms of set-theoretic structures on which deduction rules are defined and theorems can be proved [49]. A proof assistant or theorem prover is an (interactive) tool which assists the user in the development of formal proofs of properties of finite- as well as infinite-state specifications, a known advantage over model checking. Furthermore, deductive proofs can *certify* or *justify* the validity of a model-checking result [45,46]. However, there is no automatic procedure that can always determine whether there exists a derivation of a given formula in a given logic setting, which is the reason for which theorem proving typically involves interaction with a trained user.

The technical contribution of this paper is to present a set of axioms and inference rules for ACTL, which provide a sound and complete axiom system for ACTL, and which may thus form the basis for realising a proof-theoretic approach to the verification of ACTL formulae. It complements the sound and complete axiomatisations of CTL and CTL* first presented in [29] and [50], respectively.[1]

[1] A preliminary version of the axiom system was presented in [40]; here we provide a more succinct set of axioms, based on the fact that the *eventually* and *always* operators F and G can be expressed in terms of the Until operator U, cf. Section 3.

Fig. 1. Rocco, Stefania and several other contributors, as well as ¾ of the editors of this Festschrift, at the 1992 ERCIM Workshop on Theory and Practice in Verification.



Consiglio Nazionale Ricerche Pisa, Italy

European Research Consortium for Informatics and Mathematics

ERCIM WORKSHOP
on
Theory and Practice in Verification

IEI-CNR Via S. Maria, 46
Pisa, Italy
December 9-11 1992

IEI - Istituto di Elaborazione dell'Informazione del CNR

ERCIM WORKSHOP
on
Theory and Practice in Verification
Pisa, December 9-11 1992

ORGANIZATION

Coordinators

Stefania Gnesi, Paola Inverardi
Istituto di Elaborazione dell'Informazione - CNR
Via Santa Maria, 46 - Pisa

Alessandro Fantechi
Dipartimento Ingegneria dell'Informazione
Università di Pisa
Via Diotisalvi, 2 - Pisa

Locality based bisimulations for Distributed Process Algebras
(Extended Abstract)

F. Corradini and R. De Nicola
Università di Roma "La Sapienza"
Dipartimento di Scienze dell'Informazione
Via Salaria 113, I-00198 Roma (Italy)
`e-mail: denicola@vm.cnuce.cnr.it`

Abstract

A general framework for describing CCS-like languages is suggested in which both locality and causality based semantics can be captured. The non-interleaving semantics which discriminates between bisimilar processes if there is an observable difference in their distribution in space is a direct generalization of distributed bisimulation of Castellani and Hennessy. Differently from other proposals aiming at the same target, by resorting to an unbound set of location names, we make use of two locations only. The semantics based on causality can be obtained from the same set of rewriting rules. The common starting point of the two semantics enables us also to contrast locality and causality and to see why they are intrinsically different.

Table of Contents

Fig. 2. Contents of the 1992 ERCIM Workshop on Theory and Practice in Verification and the abstract of Rocco's contribution (including his likely first-ever email address).

Outline

This paper is organised as follows. Section 2 provides some relevant preliminary definitions. Section 3 contains the definition of the sound and complete axiomatisation of ACTL. Section 4 concludes the paper.

2 Basic Definitions

The semantic models for the action-based branching-time temporal logic ACTL are Labelled Transition Systems (LTSs).

Definition 1. *A* Labelled Transition System *(LTS) is a triple*

$$L = (Q, \longrightarrow, A \cup \{\tau\}),$$

where

- *Q is a set of* states*, and $u, v, w, s, t, \ldots$ range over Q.*
- *A is a* finite and non-empty *set of* visible actions*, and $a, b, c, \ldots$ range over A; τ is the* silent action*, which is not in A. We let $A_\tau = A \cup \{\tau\} = \{\ell_1, \ell_2, \ldots\}$.*
- *$\longrightarrow \subseteq Q \times A_\tau \times Q$ is the* state transition relation*. Instead of $(s, \ell, t) \in \longrightarrow$, we also write $s \xrightarrow{\ell} t$ and we call such transition an ℓ-transition.*

Remark 1. Hereafter, when we write that a state t is a *successor* of s, we intend that $\exists\, \ell \in A_\tau$ such that $s \xrightarrow{\ell} t$; if $s = t$, such transition is called a loop (on ℓ).

Definition 2. *Let $L = (Q, \longrightarrow, A \cup \{\tau\})$ be an LTS and let*

$$\longrightarrow^n = \underbrace{\longrightarrow \times \longrightarrow \times \cdots \times \longrightarrow}_{n \text{ times}} \quad \textit{and} \quad \longrightarrow^\infty = \underbrace{\longrightarrow \times \longrightarrow \times \cdots}_{\infty \text{ times}}$$

be Cartesian products of the state transition relation $\longrightarrow$. Then:

- *an infinite sequence σ of ordered triples of the form*

  $$\sigma = (s_0, \ell_0, s_1)\,(s_1, \ell_1, s_2)\,(s_2, \ell_2, s_3) \cdots \in \longrightarrow^\infty$$

  is called a path *beginning in s_0, and σ, π, δ range over paths.*
- *a finite sequence σ of ordered triples of the form*

  $$\sigma = (s_0, \ell_0, s_1)\,(s_1, \ell_1, s_2) \cdots (s_{k-1}, \ell_{k-1}, s_k) \in \longrightarrow^k$$

  is called a (finite) path *from s_0 to s_k (of* length *k).*
- *a path σ that cannot be extended, i.e. σ is infinite or ends in a state without outgoing transitions, is called a* full path.
- *$\sigma(0)$ is the* starting state *of the path σ, also denoted by first(σ).*
- *$\sigma(n)$, for some $n \geq 0$, is the n^{th} state of the path σ.*
- *if σ is a finite path, last(σ) denotes its* last state*.*
- *the n^{th}* suffix *of σ, denoted by σ^n, with $n \leq k$ for finite paths of length k, is the sequence that contains all the states of σ* starting from *$\sigma(n)$, which is thus included. Thus $\sigma^0 = \sigma$.*
- *if σ is a finite path and δ is a path such that last(σ) = first(δ), the path $\pi = \sigma\delta$ is called a* concatenation *of σ and δ (and δ is a suffix of π).*

3 The Temporal Logic ACTL

The *branching-time* temporal logic ACTL [23] is the *action-based* version of CTL [15,16] and its semantic models are LTSs. ACTL is suitable for describing the behaviour of systems that perform actions during their execution. In fact, ACTL embeds the idea of "evolution over time by actions" and is suitable for describing the various possible temporal sequences of actions that characterise a system. The original definition of ACTL includes an action calculus to improve the expressiveness of its operators.

In this section, we consider an LTS $L = (Q, \longrightarrow, A \cup \{\tau\})$ as defined above.

Definition 3. *Let $a \in A$. Then* action formulae *f, g are defined by the grammar:*

$$f, g \ ::= \ a \ \mid \ \neg f \ \mid \ f \vee g$$

Let A_{for} be the set of action formulae over A.

Intuitively, the action formulae are Boolean expressions over (visible) actions.

Next we define the satisfaction of an action formula f by a single action a, and we denote this satisfaction by $a \models f$.

Definition 4. *Let $a \in A$ and let $f, g \in A_{for}$. Then:*

$a \models a$	*always holds*
$a \models \neg b$	*holds for each $b \in A$ such that $a \neq b$*
$a \models \neg f$	*iff $a \not\models f$*
$a \models f \vee g$	*iff $a \models f$ or $a \models g$*

It is common to let tt denote a formula that is always satisfied in a calculus and to let ff correspond to $\neg tt$. In our action calculus A_{for}, we define the formula tt by choosing an $a \in A$ and letting $tt = a \vee \neg a$ (i.e. all actions of A are permitted).

Given $f \in A_{for}$, the set of actions satisfying f is defined as $[\![f]\!] = \{\, a \mid a \models f \,\}$.

Well-formed ACTL formulae are defined by the state formulae generated by the following grammar.

Definition 5. Well-formed formulae *ϕ, ψ of ACTL are defined by the grammar:*

$$\begin{aligned} \phi, \psi \ &::= \ tt \ \mid \ \phi \wedge \phi \ \mid \ \neg \phi \ \mid \ \forall \pi \ \mid \ \exists \pi \\ \pi \ &::= \ X_\tau \, \phi \ \mid \ X_f \, \phi \ \mid \ X \, \phi \ \mid \ \phi \, {}_f U \, \psi \ \mid \ \phi \, {}_f U_g \, \psi \end{aligned}$$

where $f, g \in A_{for}$ are action formulae.

Here, $\forall$ and $\exists$ are universal and existential *path quantifiers*, while X and U are (action-based) *neXt(time)* and *Until* operators (first introduced in [23]).

3.1 Models for ACTL

Let L be a *total* LTS (i.e. each state has a successor) and let R_L be a non-empty and suffix-closed set of paths on L (i.e. $\sigma \in R_L$ implies $\sigma^i \in R_L$ for all $i \geq 0$). The tuple (L, R_L) is called an *extended LTS* and it is a model for ACTL formulae. We consider only total LTSs to simplify the ACTL axiom system presented next. Note that this is not a limitation, since there is a simple way to transform any finite path of an LTS into an infinite one: it suffices to add a loop on τ in its final state. First, we define the satisfaction relation for ACTL formulae.

Definition 6. *Let M be an extended LTS, let $s \in Q$ be a state of M, and let σ be a path of M. Then the* satisfaction relation $\models$ *for well-formed ACTL formulae ϕ, ψ is inductively defined as follows:*

$$\begin{array}{ll}
M, s \models \mathit{tt} & \textit{always holds} \\
M, s \models \neg\phi & \textit{iff } M, s \not\models \phi \\
M, s \models \phi \wedge \psi & \textit{iff } M, s \models \phi \textit{ and } M, s \models \psi \\
M, s \models \forall\pi & \textit{iff } \forall\, \sigma \textit{ such that } \sigma(0) = s : M, \sigma \models \pi \\
M, s \models \exists\pi & \textit{iff } \exists\, \sigma \textit{ such that } \sigma(0) = s \textit{ and } M, \sigma \models \pi \\
M, \sigma \models X\phi & \textit{iff } M, \sigma(1) \models \phi \\
M, \sigma \models X_\tau\, \phi & \textit{iff } M, \sigma(1) \models \phi \textit{ and } \sigma(0) \xrightarrow{\tau} \sigma(1) \\
M, \sigma \models X_f\, \phi & \textit{iff } M, \sigma(1) \models \phi \textit{ and } \sigma(0) \xrightarrow{\ell} \sigma(1) \textit{ such that } \ell \neq \tau \textit{ and } \ell \models f \\
M, \sigma \models \phi\, {}_fU\, \psi & \textit{iff } \exists\, k \geq 0 \textit{ such that } M, \sigma(k) \models \psi \textit{ and } \forall\, 0 \leq j < k : \\
 & \qquad M, \sigma(j) \models \phi \textit{ and } (\sigma(j) \xrightarrow{\ell} \sigma(j+1)) \to (\ell = \tau \textit{ or } \ell \models f) \\
M, \sigma \models \phi\, {}_fU_g\, \psi & \textit{iff } \exists\, k \geq 0 \textit{ such that } M, \sigma(k+1) \models \psi,\ M, \sigma(k) \models \phi, \\
 & \qquad (\sigma(k) \xrightarrow{\ell} \sigma(k+1)) \to (\ell \models g) \textit{ and } \forall\, 0 \leq j < k : \\
 & \qquad (M, \sigma(j) \models \phi \textit{ and } (\sigma(j) \xrightarrow{\ell} \sigma(j+1)) \to (\ell = \tau \textit{ or } \ell \models f))
\end{array}$$

The meaning of the propositional operators and the CTL path quantifiers is standard. Intuitively, the neXt operator says that in the next state of the path (reached by the silent action τ or by an action satisfying f) the formula ϕ holds; the Until operator says that ψ holds at some future state of the path (reached by an action satisfying g), while ϕ holds from the current state until that state is reached and all actions executed meanwhile along the path satisfy f.

As usual, numerous modalities can be derived starting from these basic ones. In particular, we may write ff for $\neg\mathit{tt}$ and $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$. Furthermore, we define the following derived operators:

- $\exists F\, \phi$ stands for $\exists[\mathit{tt}\ {}_{\mathit{tt}}U\ \phi]$
- $\forall G\, \phi$ stands for $\neg\exists F\, \neg\phi$
- $\langle\tau\rangle\, \phi$ stands for $\exists[\mathit{tt}\ {}_{\mathit{ff}}U\ \phi]$
- $\langle a\rangle\, \phi$ stands for $\exists[\mathit{tt}\ {}_{\mathit{ff}}U_a\ \phi]$

The meaning of $\exists F\,\phi$ is that ϕ must *eventually* be true in a possible Future, while $\forall G\,\phi$ means that ϕ must *always* be true in all possible futures (Globally). The meaning of $\langle\tau\rangle\,\phi$ is that ϕ must be true in some future state reached by zero or more τ-transitions. The meaning of $\langle a\rangle\,\phi$ is that ϕ must be true in some future state reached by zero or more τ-transitions followed by an a-transition; this resembles the *diamond* modality (*possibly*) of Hennessy–Milner logic [42], which however does not require ϕ to be true immediately in the state reached by the a-transition, but allows another zero or more τ-transitions also after the a-transition before reaching the state in which ϕ is true (i.e. $\langle a\rangle\,\langle\tau\rangle\,\phi$ in ACTL). More details on the variants of Hennessy–Milner logic introduced in [42,52,24] and their relation to ACTL can be found in [37]. Finally, the dual *box* modalities (*necessarily*) of Hennessy–Milner logic, denoted by $[\cdot]\,\phi$, are defined by $\neg\langle\cdot\rangle\,\neg\phi$.

ACTL can thus be used to define the well-known properties of *liveness* ("something good eventually happens") and *safety* ("nothing bad can happen").

Definition 7. *Let M be an extended LTS and let s be a state of M. If $M, s \models \phi$, then we say that M is a model for ϕ in state s and that state formula ϕ is* satisfiable*. Analogously for path formulae. We say that ϕ is* valid*, denoted by $\models \phi$, if ϕ is satisfiable for all models and all its states (for a state formula) and similarly for a path formula.*

Note that a formula is satisfiable iff its negation is not valid.

Notation 1. Neither $\forall X_{f\vee\tau}\,\phi$ nor $\exists X_{f\vee\tau}\,\phi$ is a well-formed ACTL formula. Therefore, we define the following shorthands to be used in the rest of the paper:

$$\begin{aligned}\exists X_{f\vee\tau}\,\phi &\overset{\mathsf{def}}{=} \exists X_f\,\phi \vee \exists X_\tau\,\phi\\ \forall X_{f\vee\tau}\,\phi &\overset{\mathsf{def}}{=} \neg\exists X_{tt}\,\neg\phi \wedge \neg\exists X_\tau\,\neg\phi \wedge \neg\exists X_{\neg f}\,tt\end{aligned}$$

We are now ready to present the main (technical) contribution of this paper.

3.2 An Axiom System for ACTL

We define an axiom system for ACTL, after which we present the main result of this paper: the set of axioms and inference rules provides a sound and complete axiomatisation of ACTL.

The axiom system for ACTL is shown in Fig. 3. We now provide some explanations of this axiomatisation, discussing first the axioms and then the rules.

A0 represents any set of axioms that characterises the propositional tautologies. A possible choice could be the following:

$$\begin{aligned}&(\text{A0/1}) && (\phi \vee \phi) \to \phi\\ &(\text{A0/2}) && \phi \to (\phi \vee \psi)\\ &(\text{A0/3}) && (\phi \vee \psi) \to (\psi \vee \phi)\\ &(\text{A0/4}) && (\phi \to \psi) \to ((\phi \vee \gamma) \to (\psi \vee \gamma))\end{aligned}$$

Together with the MP rule, this is a consistent and complete axiomatisation of the calculus of the sentences.

ACTL axiom system

Axioms:

(A0) All tautology instances

(A1) $\exists X_f\, \phi \leftrightarrow \bigvee_{a \in [\![f]\!]} \exists X_a\, \phi$

(A2) $\exists X_f\, (\phi \vee \psi) \leftrightarrow (\exists X_f\, \phi \vee \exists X_f\, \psi)$

(A3) $\exists X_\tau\, (\phi \vee \psi) \leftrightarrow (\exists X_\tau\, \phi \vee \exists X_\tau\, \psi)$

(A4) $\neg\forall X_f\, tt \leftrightarrow \exists X_\tau\, tt \vee \exists X_{\neg f}\, tt$

(A5) $\neg\forall X_\tau tt \leftrightarrow \exists X_{tt}\, tt$

(A6) $\forall X_f\, \phi \leftrightarrow \forall X\, \phi \wedge \forall X_f\, tt$

(A7) $\forall X_\tau\, \phi \leftrightarrow \forall X\, \phi \wedge \forall X_\tau\, tt$

(A8) $\exists X\, \phi \leftrightarrow \exists X_{tt}\, \phi \vee \exists X_\tau\, \phi$

(A9) $\forall X\, \phi \leftrightarrow \neg\exists X\, \neg\phi$

(A10) $\forall X\, (\phi \rightarrow \psi) \rightarrow (\exists X_f\, \phi \rightarrow \exists X_f\, \psi)$

(A11) $\forall X\, (\phi \rightarrow \psi) \rightarrow (\exists X_\tau\, \phi \rightarrow \exists X_\tau\, \psi)$

(A12) $\exists X\, tt \wedge \forall X\, tt$

(A13) $\exists(\phi\, {}_fU\, \psi) \leftrightarrow \psi \vee (\phi \wedge \exists X_{f\vee\tau}\, \exists(\phi\, {}_fU\, \psi))$

(A14) $\forall(\phi\, {}_fU\, \psi) \leftrightarrow \psi \vee (\phi \wedge \forall X_{f\vee\tau}\, \forall(\phi\, {}_fU\, \psi))$

(A15) $\exists(\phi\, {}_fU_g\, \psi) \leftrightarrow \phi \wedge (\exists X_g\, \psi \vee \exists X_{f\vee\tau}\, \exists(\phi\, {}_fU_g\, \psi))$

(A16) $\forall(\phi\, {}_fU_g\, \psi) \leftrightarrow (\phi \wedge \forall X_{f\vee g\vee\tau}\, tt \wedge \neg\exists X_{\neg f\wedge g}\, \neg\psi \wedge$
$\neg\exists X_{f\wedge g}\, (\neg\forall(\phi\, {}_fU_g\, \psi) \wedge \neg\psi) \wedge \neg\exists X_{(f\wedge\neg g)\vee\tau}\, (\neg\forall(\phi\, {}_fU_g\, \psi)))$

(A17) $\forall G\, (\gamma \rightarrow (\neg\psi \wedge \neg\exists X_{f\vee\tau}\, (\exists(\phi\, {}_fU\, \psi) \wedge \neg\gamma))) \rightarrow (\gamma \rightarrow \neg\exists(\phi\, {}_fU\, \psi))$

(A18) $\forall G\, (\gamma \rightarrow (\neg(\phi \wedge \exists X_g\, \psi) \wedge \neg\exists X_{f\vee\tau}\, (\exists(\phi\, {}_fU_g\, \psi) \wedge \neg\gamma))) \rightarrow$
$(\gamma \rightarrow \neg\exists(\phi\, {}_fU_g\, \psi))$

(A19) $\forall G\, (\gamma \rightarrow (\neg\psi \wedge \exists X\, \gamma)) \rightarrow (\gamma \rightarrow \neg\forall(\phi\, {}_fU\, \psi))$

(A20) $\forall G\, (\gamma \rightarrow (\exists X_{(f\wedge\neg g)\vee\tau}\, \gamma \vee \exists X_{f\wedge g}\, (\neg\psi \wedge \gamma))) \rightarrow (\gamma \rightarrow \neg\forall(\phi\, {}_fU_g\, \psi))$

(A21) $\forall G\, (\gamma \rightarrow (\neg\psi \wedge \exists X\, \gamma)) \rightarrow (\gamma \rightarrow \neg\forall F\, \psi)$

(A22) $\forall G\, (\gamma \rightarrow (\neg\psi \wedge \forall X\, \gamma)) \rightarrow (\gamma \rightarrow \neg\exists F\, \psi)$

Rules:

(R∀X) $\dfrac{\vdash \phi}{\vdash \forall X\, \phi}$ (MP) $\dfrac{\vdash \phi \rightarrow \psi \qquad \vdash \phi}{\vdash \psi}$ (R∀G) $\dfrac{\vdash \phi}{\vdash \forall G\, \phi}$

Fig. 3. The axiom system of ACTL.

A1 defines the $\exists X_f$ operator in terms of single action $\exists X_a$ operators.

A2–A12 concern the quantified neXt operators. More precisely:

- **A2** distribution law related to visible actions satisfying f.
- **A3** distribution law related to the silent action τ.
- **A4** defines the relation between the universal and existential next operators. It says that if not all the states that are successors of a state s are reachable from s by satisfying f, then at least one of them is reachable either by the silent action τ or by an action that does not satisfy f; the reverse holds too.
- **A5** defines the separation between the visible actions and the silent action. It says that if not all the states that are successors of a state s are reachable from s by the silent action τ, then there is one such successor state that is reachable by a visible action satisfying f, and vice versa.
- **A6** defines the $\forall X_f$ operator. It says that if a state s satisfies $\forall X_f\, \phi$, then all the successors of s satisfy ϕ ($\forall X \phi$) and, moreover, they are all reachable by actions that satisfy f ($\forall X_f\, tt$).
- **A7** defines the $\forall X_\tau$ operator in a way that is analogous to A6.
- **A8** defines the $\exists X$ operator.
- **A9** defines the $\forall X$ operator as the dual of the $\exists X$ operator.
- **A10** distribution law.
- **A11** distribution law.
- **A12** guarantees that each model for ACTL formulae must be *total*.

A13–A16 show the inductive way by which $\exists(\phi\, {}_fU\, \psi)$, $\forall(\phi\, {}_fU\, \psi)$, $\exists(\phi\, {}_fU_g\, \psi)$ and $\forall(\phi\, {}_fU_g\, \psi)$ (in a slightly different way, cf. [40] for details) propagate themselves along the paths of models. Note that A13–A16 do not forbid the infinite unfolding of the Until operators, which is handled by A17–A20.

A17–A20 avoid the infinite unfolding of the Until operators. The trick is to use a *placeholder* γ to characterise the case of infinite unfolding of an operator O that should actually have a *finite* unfolding; we then say that if γ holds in a state (i.e. such a state is the initial one for an infinite unfolding of O), then in such a state O cannot hold.

A21–A22 avoid the infinite unfolding of the eventually operators F in a way similar to the way this is done for the Until operators in A17–A20.

MP the usual Modus Ponens.

R∀X ensures that the theorems of the inference systems are closed under the most general universal neXt operator.

R∀G ensures that the theorems of the inference systems are closed under the always operator G.

We say that a formula ϕ can be inferred from an axiom system, denoted by $\vdash \phi$, if there exists a finite sequence of formulae, ending with ϕ, such that each formula is an instance of one of the axioms or follows from previous formulae by applying one of the rules.

Finally, the next theorem ensures the soundness and completeness of the axiom system for ACTL.

Theorem 1 (Soundness and Completeness). *Each well-formed ACTL formula ϕ is valid if and only if it can be inferred from the ACTL axiom system, i.e.*

$$\vdash \phi \quad \leftrightarrow \quad \models \phi$$

Proof sketch. ($\vdash \phi \rightarrow \models \phi$) The soundness proof is a rather standard proof by induction on the structure of the derivation of ϕ.

($\models \phi \rightarrow \vdash \phi$) The completeness proof is quite long and tedious; therefore, we only provide an outline. It uses a technique from [30,26] based on a decision algorithm for the satisfiability of CTL formulae. This technique is a variant of the tableau approach, which was applied to the branching-time logics considered in [12,34].

A formula ψ is *consistent* if $\neg\psi$ cannot be inferred from the axiom system. To show that any valid ACTL formula can be inferred from the axiom system, it thus suffices to show that any consistent ACTL formula is satisfiable.

Let ϕ be a consistent ACTL formula. Then we need to define a procedure to characterise a model for ϕ in a structural way, i.e. in a way that allows us to automatically build and manipulate an LTS to achieve a model for ϕ. To do so, we define a Fischer–Ladner *finite* closure set for ϕ (cf. [39]) and a particular class of LTSs, so-called Hintikka Structures (HS), as in [12,30]. HS have the property that each of their states is labelled by a subset of the Fischer–Ladner closure of ϕ and we say that an HS is an HS *for* ϕ if one of its states contains the formula ϕ. The following property holds: each model for ϕ is an HS for ϕ, and each HS for ϕ is extendible to a model for ϕ without changing the number of its states. Hence, if we have an algorithm that returns an HS for ϕ, then we know that ϕ is satisfiable.

In order to write a procedure that takes ϕ, calculates its Fischer–Ladner closure and tries to build an HS for ϕ, we must ensure that such a procedure will terminate, i.e. that it is possible to build a *finite* HS for ϕ if ϕ is satisfiable. To achieve this, we prove that ϕ is satisfiable if and only if it is possible to build a *finite* HS that satisfies ϕ.

We conclude this proof sketch with an outline of the above mentioned decision procedure:

1. Calculate the Fischer–Ladner closure of ϕ.
2. Let M be the set of all maximal subsets of this closure. Build an LTS L that satisfies a minimal subset of conditions among those defining an HS and whose states are elements of M. This ensures that whenever a finite HS for ϕ exists, it is contained in L.

3. Purge all states of L that do not match the definition of HS. We prove that (i) if ϕ is consistent, then there exists a consistent element S of M that contains ϕ, and (ii) if a state is purged, then it was not consistent. Hence, S cannot be purged by the procedure, and S will be a state of the resulting HS that is calculated by the procedure. But S contains ϕ, so we obtain an HS for ϕ and hence ϕ is satisfiable. □

4 Conclusion

In this paper, we have revisited De Nicola & Vaandrager's action-based logic ACTL. We have sketched the context in which it was introduced 30 years ago and the impact it has had on our research and that of many of our colleagues of the Formal Methods and Tools Lab. Furthermore, we have revamped an axiom system for ACTL that has originally been published in the proceedings of a national conference [40], by providing a more concise sound and complete axiom system for ACTL.

Axiomatisation of a logic is often said to offer a better understanding of the logic. Moreover, the ACTL axiom system may form the basis for developing a theorem prover for the verification of ACTL formulae. In [40], a preliminary proof assistant for ACTL implemented in HOL (http://hol-theorem-prover.org) was described. Other directions for future work include the consideration of infinite-state systems, to overcome limitations of model checking, and to investigate the use of ACTL theorem proving to certify or justify the validity of an ACTL model-checking result.

Finally, it would be interesting to develop an axiom system also for ACTL*. This logic, as is the case for CTL*, includes both linear- and branching-time operators, and it is well known that the model-checking algorithms for this class of logics are PSPACE-complete. A proof-theoretic approach for ACTL* formulae might ease verification for at least some classes of properties. However, it is known from [26] and [31,32] that the complexity of checking satisfiability of CTL and CTL* is EXPTIME-complete and 2-EXPTIME-complete, respectively, in the length of the formula.

Acknowledgements Stefania wishes to thank Salvatore Larosa, who worked on the ACTL axiomatisation; Alessandro Fantechi, Franco Mazzanti, and Monica Nesi, for interesting discussions on the preliminary version of the ACTL axiomatisation; and Maurizio La Bella, who developed the ACTL proof assistant. And, *last but not least*, Stefania would like to thank Rocco, for having initiated this line of research that has led to so many interesting papers, projects, and collaborations with many different people, and which in hindsight has made it worthwhile to remain at the CNR.

Maurice also would like to thank Alessandro and Franco, for numerous pleasant collaborations on, among others, ACTL-like logics and the KandISTI family. And, *of course*, also Rocco, for quality time spent together during a number of projects, not limited to research.

References

1. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge Tracts in Theoretical Computer Science, vol. 18. Cambridge University Press (1990). doi: 10.1017/CBO9780511624193
2. Baeten, J.C.M., Basten, T., Reniers, M.A.: Process Algebra: Equational Theories of Communicating Processes. Cambridge Tracts in Theoretical Computer Science, vol. 50. Cambridge University Press (2010). doi: 10.1017/CBO9781139195003
3. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In: *Revised Selected Papers of the 12th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'07)*. LNCS, vol. 4916, 133–148. Springer (2008). doi: 10.1007/978-3-540-79707-4_11
4. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A state/event-based model-checking approach for the analysis of abstract system properties. *Science of Computer Programming* **76**(2), pp. 119–135 (2011). doi: 10.1016/j.scico.2010.07.002
5. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: Using FMC for family-based analysis of software product lines. In: *Proceedings of the 19th International Software Product Line Conference (SPLC'15)*, pp. 432–439. ACM (2015). doi: 10.1145/2791060.2791118
6. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: States and Events in KandISTI: A Retrospective. In: *Models, Mindsets, Meta: The What, the How, and the Why Not?* LNCS, vol. 11200, pp. 110–128. Springer (2018).
7. ter Beek, M.H., Gnesi, S., Mazzanti, F.: From EU projects to a family of model checkers. In: *Software, Services and Systems*. LNCS, vol. 8950, pp. 312–328. Springer (2015). doi: 10.1007/978-3-319-15545-6_20
8. ter Beek, M.H., Mazzanti, F., Gnesi, S.: CMC–UMC: A framework for the verification of abstract service-oriented properties. In: *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC'09)*, pp. 2111–2117. ACM (2009). doi: 10.1145/1529282.1529751
9. ter Beek, M.H., Mazzanti, F., Sulova, A.: VMC: A tool for product variability analysis. In: *Proceedings of the 18th International Symposium on Formal Methods (FM'12)*. LNCS, vol. 7436, pp. 450–454. Springer (2012). doi: 10.1007/978-3-642-32759-9_36
10. ter Beek, M.H., Mazzanti, F.: VMC: Recent Advances and Challenges Ahead. In: *Proceedings of the 18th International Software Product Line Conference (SPLC'14)*, pp. 70–77. ACM (2014). doi: 10.1145/2647908.2655969
11. Ben–Ari, M., Pnueli, A., Manna, Z.: The temporal logic of branching time. In: *Proceedings of the 8th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages (POPL'81)*, pp. 164–176. ACM (1981). doi: 10.1145/567532.567551
12. Ben–Ari, M., Pnueli, A., Manna, Z.: The temporal logic of branching time. *Acta Informatica* **20**(3), pp. 207–226 (1983). doi: 10.1007/BF01257083
13. Bernardeschi, C., Fantechi, A., Gnesi, S., Larosa, S., Mongardi, G., Romano, D.: A Formal Verification Environment for Railway Signaling System Design. *Formal Methods in System Design* **12**(2), pp. 139–161 (1998). doi: 10.1023/A:1008645826258
14. Bouali, A., Gnesi, S., Larosa, S.: JACK: Just Another Concurrency Kit – The integration project. *Bulletin of the EATCS* **54**, pp. 207–223 (1994).

15. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: *Proceedings of the Workshop on Logics of Programs*. LNCS, vol. 131, pp. 52–71. Springer (1981). doi: 10.1007/BFb0025774
16. Clarke, E.M., Emerson, E.A.: Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming* **2**(3), pp. 241–266 (1982). doi: 10.1016/0167-6423(83)90017-5
17. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In: *Proceedings of the 10th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages (POPL'83)*, pp. 117–126. ACM (1983). doi: 10.1145/567067.567080
18. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* **8**(2), pp. 244–263 (1986). doi: 10.1145/5397.5399
19. De Nicola, R., Fantechi, A., Gnesi, S., Larosa, S., Ristori, G.: Verifying hardware components with JACK. In: *Proceedings of the 8th Advanced Research Working Conference on Correct HARdware Design MEthodologies (CHARME'95)*. LNCS, vol. 987, pp. 246–260. Springer (1995). doi: 10.1007/3-540-60385-9_15
20. De Nicola, R., Fantechi, A., Gnesi, S., Ristori, G.: An action based framework for verifying logical and behavioural properties of concurrent systems. In: *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*. LNCS, vol. 575, pp. 37–47. Springer (1991). doi: 10.1007/3-540-55179-4_5
21. De Nicola, R., Fantechi, A., Gnesi, S., Ristori, G.: An action-based framework for verifying logical and behavioural properties of concurrent systems. *Computer Networks and ISDN Systems* **25**(7), pp. 761–778 (1993). doi: 10.1016/0169-7552(93)90047-8
22. De Nicola, R., Inverardi, P., Nesi, M.: Using the axiomatic presentation of behavioural equivalences for manipulating CCS specifications. In: *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. LNCS, vol. 407, pp. 54–67. Springer (1989). doi: 10.1007/3-540-52148-8_5
23. De Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In: *Semantics of Systems of Concurrent Processes: Proceedings of the LITP Spring School on Theoretical Computer Science*. LNCS, vol. 469, pp. 407–419. Springer (1990). doi: 10.1007/3-540-53479-2_17
24. De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation (extended abstract). In: *Proceedings of the 5th Annual Symposium on Logic in Computer Science (LICS'90)*, pp. 118–129. IEEE (1990). doi: 10.1109/LICS.1990.113739
25. De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation. *Journal of the ACM* **42**(2), pp. 458–487 (1995). doi: 10.1145/201019.201032
26. Emerson, E.A.: Temporal and Modal Logic. In: *Handbook of Theoretical Computer Science*, vol. B: Formal Models and Semantics, pp. 995–1072. Elsevier (1990). doi: 10.1016/B978-0-444-88074-1.50021-4
27. Emerson E.A., Halpern, J.Y.: "Sometimes" and "not never" revisited: On branching versus linear time (preliminary report). In: *Proceedings of the 10th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages (POPL'83)*, pp. 127–140. ACM (1983). doi: 10.1145/567067.567081
28. Emerson E.A., Halpern, J.Y.: "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM* **33**(1) (1986), pp. 151–178. doi: 10.1145/4904.4999

29. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. In: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82)*, pp. 169–180. ACM (1982). doi: 10.1145/800070.802190
30. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences* **30**(1), pp. 1–24 (1985). doi: 10.1016/0022-0000(85)90001-7
31. Emerson, E.A., Jutla, C.S.: The Complexity of Tree Automata and Logics of Programs (Extended Abstract). In: *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pp. 328–337. IEEE (1988). doi: 10.1109/SFCS.1988.21949
32. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. *SIAM Journal on Computing* **29**(1), pp. 132–158 (1999). doi: 10.1137/S0097539793304741
33. Emerson, E.A., Lei, C.-L.: Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract). In: *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, pp. 267–278. IEEE (1986).
34. Emerson, E.A., Sistla, A.P.: Deciding full branching time logic. *Information and Control* **61**(3), pp. 175–201 (1984). doi: 10.1016/S0019-9958(84)80047-9
35. Fantechi, A., Gnesi, S., Lapadula, A., Mazzanti, F., Pugliese, R., Tiezzi, F.: A logical verification methodology for service-oriented computing. *ACM Transactions on Software Engineering and Methodology* **21**(3), pp. 16:1–16:46 (2012). doi: 10.1145/2211616.2211619
36. Fantechi, A., Gnesi, S., Mazzanti, F., Pugliese, R., Tronci, E.: A Symbolic Model Checker for ACTL. In: *Proceedings of the International Workshop on Current Trends in Applied Formal Methods (FM-Trends'98)*, LNCS, vol. 1641, pp. 228–242. Springer (1999). doi: 10.1007/3-540-48257-1_14
37. Fantechi, A., Gnesi, S., Ristori, G.: Model Checking for Action-Based Logics. *Formal Methods in System Design* **4**(2), pp. 187–203 (1994). doi: 10.1007/BF01384084
38. Fantechi, A., Gnesi, S., Semini, L.: Formal Description and Validation for an Integrity Policy Supporting Multiple Levels of Criticality. In: *Dependable Computing and Fault-Tolerant Systems, vol. 12: Proceedings of the 7th IFIP international Conference on Dependable Computing for Critical Applications (DCCA-7)*, pp. 129–146. IEEE (1999). doi: 10.1109/DCFTS.1999.814293
39. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences* **18**(2), pp. 194–211 (1979). doi: 10.1016/0022-0000(79)90046-1
40. Gnesi, S., Larosa, S.: A sound and complete axiom system for the logic ACTL. In: *Proceedings of the 5th Italian Conference on Theoretical Computer Science (ICTCS'95)*, pp. 343–358. World Scientific (1996). doi: 10.1142/9789814531184
41. Gnesi, S., Mazzanti, F.: On the fly verification of networks of automata In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pp. 1040–1046. CSREA Press (1999).
42. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM* **32**(1), pp. 137–161 (1985). doi: 10.1145/2455.2460
43. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985).
44. Milner, R.: Communication and Concurrency. Prentice Hall (1989).
45. Namjoshi, K.S.: Certifying Model Checkers. In: *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*. LNCS, vol. 2102, pp. 2–13. Springer (2001). doi: 10.1007/3-540-44585-4_2

46. Peled, D., Pnueli, A., Zuck, L.: From Falsification to Verification. In: *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS'01)*. LNCS, vol. 2245, pp. 292–304. Springer (2001). doi: 10.1007/3-540-45294-X_25
47. Pnueli, A.: Linear and Branching Structures in the Semantics and Logics of Reactive Systems. In: *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming (ICALP'85)*. LNCS, vol. 194, pp. 15–32. Springer (1985). doi: 10.1007/BFb001572
48. Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: *Proceedings of the 5th International Symposium on Programming (Programming'82)*. LNCS, vol. 137, pp. 337–351. Springer (1982). doi: 10.1007/3-540-11494-7_22
49. Ray, S.: Scalable Techniques for Formal Verification. Springer (2010). doi: 10.1007/978-1-4419-5998-0
50. Reynolds, M.: An Axiomatization of Full Computation Tree Logic. *The Journal of Symbolic Logic* **66**(3), pp. 1011–1057 (2001). doi: 10.2307/2695091
51. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall (1997).
52. Stirling, C.: An Introduction to Modal and Temporal Logics for CCS. In: *Proceedings of the UK/Japan Workshop on Concurrency: Theory, Language, and Architecture (CONCURRENCY'89)*. LNCS, vol. 491, pp. 2–20. Springer (1991). doi: 10.1007/3-540-53932-8_41
53. Stirling, C.: Modal and temporal logics. In: *Handbook of Logic in Computer Science*, vol. 2: Background: Computational Structures, pp. 477–563. Oxford University Press (1993).