Tense & temporality: Computing and the logic of time

Troy Kaighin Astarte

This chapter explores the role of time in logic, from ancient history to modern computing. It provides an outline and primer on the development of tense logic, and explores how these ideas found their way into computer science. The chapter examines how certain philosophical problems in this space saw new light as technical questions, and explores the role of logic in computer science.

Keywords: history of computing, logic, concurrency, history of science, philosophy of science

1 Introduction

The consideration of time in logic is both ancient and thoroughly modern. From Greek philosophy to computer science, investigation into tensed logical statements showed they increase the expressiveness of a logic and its connection with the real world, but also its complexity. While the classic Aristotelian view was that a statement could change its truth value with time, this became theologically concerning when examining statements about determinism and the omniscience of God. Debate on the logic of time saw detailed and thorough treatment only in the 1950s when A. N. Prior developed 'tense logic', a form of modal logic[1] in which notions of time and contingency are expressed as modalities. Although Prior saw an immediate application to computing, the ideas broke into this field only in the 1970s, and via alternate means.

1 This branch of logic concerns predicates which express possibility or certainty, typically through 'possibly' and 'necessarily' operators. Such logics are often evaluated by means of a 'possible world' semantics which allows the linking of modal propositions. Neither the logic itself nor its history is discussed in detail here; the interest reader is referred to Hughes and Cresswell (1968) as a standard reference or Goldblatt (2006) for its history.

In the late 1960s Manna was working on systems for proving termination of programs. Burstall took some of Manna's ideas and gave them a different notation in a 1974 paper—and made the connection to modal and temporal operators. Meanwhile, Pnueli, who had worked alongside Manna and Francez on cyclical programs, took Burstall's ideas and applied them to concurrent behaviour, naming this 'The temporal logic of programs'. Concurrent systems are those in which different components act simultaneously while sharing certain resources; they are mandated by the presence of hardware which operates at different speeds, and concurrent behaviour may be deliberately introduced into programs for performance benefits or to model real-world parallel systems. Handling the behaviour of such programs has been recognised as a difficult task since at least the 1950s,[2] and the 70s saw the emergence of various theoretical frameworks—of which temporal logic is one—for modelling and reasoning about concurrent behaviour.

The case of temporal logic demonstrates an example of ancient ideas considered purely philosophical ending up with serious practical applications. One interesting facet of this is the rediscovery of many old problems in new contexts. Computer scientists working with temporal logic came against and argued about issues such as whether time should be considered branching or linear; the interpretation of future tensed propositions in the present; and the creation of appropriate models for logical systems. In most cases, these computer scientists considered these problems entirely unaware of the history behind them.

This chapter explores the 'logic' background of temporal logic and investigates how the ideas came into computing. It considers the early field of temporal logic and shows examples of old arguments recurring in new contexts. It begins with a brief overview of some pre-modern views of logic in time, before considering some areas of particular interest that re-emerge in the computing era. Prior's tense logic is introduced, and then the routes by which these ideas made their way into computer science literature is examined. Temporal logic for concurrency is then discussed and the earlier problems are reintroduced; the chapter concludes by considering what this story says about the role of logic in computer science. The case is made that temporal logic, like much of theoretical computer science, sits neither in logic or mathematics, nor within the practical toolset of the everyday programmer, but somewhere else entirely.

A large part of the material in this chapter on the pre-computing era draws from the analysis by Øhrstrøm and Hasle (2007) who provide both contextually accurate and

2 The major challenge is preventing harmful interference which can arise when different concurrent agents access a resource simultaneously. See Astarte (2023) for discussion of the historical emergence of this problem and some of the ways to address it.

modern rephrasing of historical logic systems. A final note on terminology: in this chapter, 'tense logic' is taken to refer to the logic of Prior and others, working outside a computing context; 'temporal logic' will denote such logic ideas applied to computing.

2 Old problems, old logics

An early problem, discussed in the work of Aristotle, was how to cope with the inherent unknowability of the future (Øhrstrøm and Hasle 2007, §1.1). In *On Interpretation* (Chapter IX), the example is presented of potential fight happening at sea tomorrow. We use Fs to denote the statement "There will be a sea-fight tomorrow", for reasons which will become apparent later. The question is: how can the truth of Fs be interpreted today?

If there are two potential outcomes which are opposite—a fight, or no fight—and neither of these is contingent on anything today, but that everything which happens (i.e., every true statement) is necessitated; then there is no way to determine right now whether Fs or its negation is true. Aristotle recognised the modality of this situation: if tomorrow there is indeed a fight, making Fs true yesterday, did that make it also necessary? And if tomorrow there is no fight, was Fs not possible yesterday? Aristotle's perspective was that the past and present are deterministic, but the future is not. All true statements about the past and present are necessary, but true statements about the future might only be possible. This, however, created a disquieting asymmetry between past and future.

One way to interpret Aristotle's position was reasoned out by Richard of Lavenham (c. 1380) starting with the idea that it might simply be impossible to interpret statements like Fs (Øhrstrøm 1983). To many religious logicians, like Lavenham, this was problematic due to God's omniscience; and the future being deterministic was equally unappealing. Lavenham's solution was to reject the necessity of the past. An alternative approach, favoured by Jan Łukasiewicz (1873–1956), was to reject the law of the excluded middle, and introduce a third truth value, 'undetermined', which could be the result of evaluating statements like Fs (Øhrstrøm and Hasle 2020).

More relevant for the current chapter, another interpretation of Aristotle was put forward by William of Ockham, and later Peirce, which allowed both the excluded middle for future propositions, and non-determinism of the future. These were reasoned out into full modern logical systems by Prior (1967). The idea was to reject the classical view of time as a single line.

The alternative was to view the future as branching out from the present like a tree, in a series of different possibilities. A simple illustration of the sea-fight branch can be

seen in Figure 1. The present can even be considered as one of a set of potential presents along various parallel lines of time, no longer accessible due to previous forks. As time progresses, one of the possible futures becomes the present, and then the past, leaving an asymmetric tree with the past as a line.

[Fig. 1] Aristotle's sea fight example

Using this branched notion of time allowed Prior to develop a few ways to address the knowability of future events. One is that all future paths are true until they reach the present and are evaluated; though Prior didn't care for that one; after Ockham, the most religiously satisfying view is that only one future is true but is as yet unknowable to mere humans; finally, after Peirce, another interpretation of future propositions is that they are true only when true in all possible futures. Branching approaches like these, especially the second, allowed logicians to solve the omniscience problem: while to a human the various potential paths might be unknown and therefore subject to free will, God is able to identify which future is the real one.

Prior's contributions to the logic of time are not confined to the branching structure thereof; the next section examines his work and sources of inspiration.

3 Prior & before

Arthur Norman Prior (1916–1969) was born in New Zealand and brought up a devout Christian, even intending to follow a career as a Presbyterian minister while he studied philosophy at the University of Otago (Copeland 2020). Prior's early interests lay in ethical philosophy, and the relationship between free will and omniscience. He was introduced to logic by one of his tutors, John Findlay. Prior favoured realism over idealism and was opposed to purely formal logic in philosophy, writing that logic was useless unless it helps describe the world (Jakobsen 2020). After a little work on ethical logic, Prior became interested in the history of logic. Likely inspired by Findlay, who had sketched some propositions for a calculus of time, Prior began to study, among others, Peirce, Łukasziewicz, and Boole.

Boole had written about time in the context of probability, assigning 'simple' propositions a numerical value between 0 and 1, combining them with algebraic laws to create more complex events (Øhrstrøm and Hasle 2007, §2.1). However, he introduced an interesting "surreptitious" shift in which these values represented not the probability of the events occurring, but rather the truth of statements that those events occurred (Durand-Richard 2023). This allowed him to sidestep the philosophically-challenging problem of ascertaining the true independence of events. Prior shared Boole's belief in

the importance of finding a role for time in symbolic algebra, though he felt Boole's logic, lacking specific operators for tensed statements, was insufficiently rich (Prior 1957, Appx. A).

Like the mediaeval scholars he had studied, Prior believed that since human speech naturally contains many tensed statements, any system of logic which purports to describe the world as experienced by humans must be equally rich in its ability to express temporal notions. Critically, Prior held that sentences like 'Socrates is sitting' must be viewed as complete without needing extra components such as 'at time t' to become interpretable (Øhrstrøm and Hasle 2007, §2.5).

Prior's approach here came from his historical knowledge, especially Peirce and Łukasiewicz. Peirce too had studied mediaeval logic and used concepts from this in his semiotics, though he believed that the early 20th Century was not the right time to bring temporal aspects back into logic (ibid., §2.2). Peirce related modality with time by connecting 'actuality' with the past and present, and 'possibility' and 'necessity' with the future, which enabled his view of the future as a set of branching possibilities, and his proposal for rationalising the apparently competing doctrines of human consciousness, omniscience, and determinism.

Łukasiewicz's work in the 1920s and 30s had established a new 'Polish' school of logic, cementing the logical paradigm in which Prior later worked. One relevant facet of that approach was the translation of historical logical systems into symbolic ones, thereby allowing their properties to be analysed alongside modern logic. Just as Prior would, Łukasiewicz used concepts from ancient and mediaeval logic in his original work (Simons 2021). This was present in his interpretation of Aristotle as describing a contingent future view which rejected bivalence in favour of a third, 'possible', truth value. Though Prior did not himself favour trivalence, he was full of praise for Łukasiewicz in many other ways, and made extensive use of his Polish notation.[3]

Armed with this inspiration, Prior developed a number of systems he called 'tense logic' based on the ordering of events in time. He took the universe of possible worlds from modal logic into the temporal domain as instants in time; and the accessibility relation between these instants became sequentiality. In this context, the modal 'necessity' operator because a temporal 'always', and 'possibility' became 'at some point'. Prior presented his work in the John Locke series of lectures in Oxford 1955–6, and published a book the next year (Prior 1957). By 1967, the ideas had caused

3 This concise system for writing propositions uses only the Roman alphabet but is deeply expressive and has the benefit of unambiguous order of interpretation without requiring brackets. For more on the way this writing tool shaped logic, see Dunning (2020).

significant waves in the logic community (Copeland 2020) and Prior had written a sequel, in which he presented a variety of logical systems and their subsequent properties (Prior 1967).

In the present chapter we will consider the basic aspects of Prior's tense logic and some specific points, rather than attempting to cover every single system.[4] Tensed statements are expressed as logical propositions combining untensed terms, usually lowercase Roman letters, with uppercase Roman letter operators capturing the tense. P and F indicate that something happened at some point in the past or future (respectively) and are outlined in Figure 2.a. H and G capture events that have happened, or are going to happen, continuously; they are presented in Figure 2.b. These statements are interpreted with reference to a privileged instant which is 'now'.

[Fig. 2] (2.a) Expressing tensed notions with indeterminate operators, (2.b) Determinate tense logic operators

This system also does not allow the expression of duration. In order to study this, we first need to examine another critical problem in the logic of time: the basic units of the logical system.

Though Prior (1967) discussed this idea in some depth, examining the consequences of various choices, it has a longer pedigree, much like the question of linear or branching time. As far back as Aristotle, again, the question arose of whether to consider time as a dynamic continuum, or a series of static instants. It was given a particularly thorough examination in the work of John McTaggart Ellis McTaggart,[5] who used this in his provocative argument for the unreality of time (McTaggart 1908).

McTaggart classified logical systems about time into two types: A and B. The A-theory view of time is that presented above; it is the logic of tenses and is operator-first, with statements made relative to a privileged 'now'. Operators are defined axiomatically (e.g., $Gp \stackrel{\text{def}}{=} \neg F \neg p$) and they are able to translate temporal statements lacking measurements into a formal setting. See Figure 3.a for definitions.

By contrast, B-theory systems start by defining the notion of a comparable instant. Logical propositions are associated with instants using an operator which lets us judge the truth of a proposition at a given instant. B-theory relies on concepts of simultaneity, before, and after as its main notions; critically, it does not afford privilege to

4 The interested reader is referred to Øhrstrøm and Hasle (2007) for extensive discussions and many fine points.

5 The unusual duplicity in his name is due to being named 'John McTaggart Ellis' at birth, after his great uncle John McTaggart. When that latter died, he left his estate to the Ellis family on the condition that they changed their family name to McTaggart. They complied by adding it to the end of every family member's name (McDaniel 2020).

any particular instant, not even 'now'. The same temporal operators can be defined as in A-theory, but are now given in relation to the series of instants. Definitions, following Prior (1967) and Øhrstrøm and Hasle (2007), are given in Figure 3.b.

Operators from A-theory can be defined with relation to the time series of B-theory, as seen in Figure 3.b. Øhrstrøm and Hasle (2007, §3.2) detail an argument that B-theory should be seen as the 'basic' system, with the premise that B-theory concepts cannot be expressed in A-theory since it has no way to express the ordering of events. That the B-theory is seen as the standard is reflected in its primacy in the Stanford Encyclopedia of Philosophy entry on temporal logic (Goranko and Rumberg 2022). However, Prior was opposed to that view, in part motivated by his own realist philosophy which privileged human-intuitive models, including (in his view) A-theory. Øhrstrøm and Hasle (2007, §3.2) agree, and provide a mechanism through which A- and B-theory concepts can be freely inter-translated.

It is notable that many modern logics were developed by people familiar with complex mathematics (e.g. Russell, Kripke, Scott). By contrast, Prior did not train in maths until later in life, and tense logic was developed in an almost purely philosophical setting. This created an emphasis, at least initially, on using tense logic for conceptual investigations, hence the strong connection with statements in natural language. Tense logic did, however, still attract the intellectual cachet of formalism, though its form differed: A-theory lent itself to an axiomatic presentation and B-theory to a semantic model (ibid., §2.8). This is how they are presented in Figures 3.a and 3.b.

[Fig. 3] (3.a) Prior's A system of tense logic, (3.b) Prior's B system of tense logic.

The distinction is relevant for the application of tense logic to computing, for which the potential utility of discrete-time B-systems was recognised by Prior (1967, 67). He wrote they "are applicable in limited fields of discourse in which we are concerned only with what happens next in a sequence of discrete states, e.g. in the working of a digital computer." Despite this observation in 1967, the impact of tense logic on computing did not come until well into the next decade. This history is traced in the next section.

4 Time, for computers

One early glimpse of a logic of time in a computing setting came from McCarthy and Hayes (1969), writing a review of ideas from philosophy that could be exploited for their AI work at Stanford. The authors propose a 'fluent' function which allows statements such as 'given one situation, a certain other will eventually result'. The authors explicitly made the connection between this situation calculus and Prior's tense logic,

presenting inter-translation of the relevant operators. They also propose a 'formal literature' which is "like a formal language with a history: we imagine that up to a certain time a certain sequence of sentences have been said. The literature then determines what sentences may be said next" (ibid., 32). As well as an interesting anticipation of 2010s natural language processing AI, this notion has a reflection in later models of temporal logic which use sequences of (abstract) computer states as models for history, and which determine the future properties of the system or its allowable actions.

While this paper shows glimpses of how the ideas of tense logic could be used in computing, it seems to have made little impact on the early development of temporal logic in computer science. This could be because of its focus on the AI domain, quite different to the applications for which early temporal logic work were developed. (It has a large number of citations in AI literature). Instead, the line of work taking tense logic to concurrency goes via another Stanford connection, the work of Zohar Manna on termination.

The task of verifying the correctness of programs is difficult and has a long history (Jones 2003). An early proponent was Bob Floyd (1967), who drew flowcharts of programs with logical assertions expressing properties at particular points. Floyd then conducted semi-formal proofs using axioms about the program behaviour and input to trace through its intended execution, demonstrating that particular conditions about the program output would be true. In this way, Floyd, and others who followed in his style such as Hoare (1969) argued that a program could be verified correct.[6]

However, should there be some error in program or input that prevents it terminating (for example, a loop which is supposed to count down to a target value instead counts upward indefinitely), the program's correctness cannot be shown. Since most programs in the 1960s were 'functional' (i.e., intended to produce an answer), proving termination was a desirable feature. Due to the halting problem (Davis 1965), an algorithm to automatically determine whether an arbitrary program terminates is impossible; but for some programs it is possible to construct a proof of termination. Floyd used his annotations to show that every step of the program decreased some quantity towards a limit (e.g., a loop counter decrementing to zero). The inherent future tense in the question "Will this program terminate?" provides a hint to why this line of work eventually involved tense logic.

---

6 Later proponents of program verification would come to talk (more precisely) about a program 'meeting its specification' and detractors argued about the applicability of proofs made about abstract representations to the physical working of programs—but these concerns are out of scope for the current chapter and the interested reader is directed to MacKenzie (2001, Ch. 6).

We see in the work of Floyd and Hoare some early attempts to bring programs into the domain of mathematical proofs. There is yet to be any explicit connection with *logic* (though later Hoare's work would be termed 'Hoare logic') and neither author provides citations to literature outside the computing field. Instead we can observe that their use of logico-mathematical constructs like propositions and predicates assumes the audience has familiarity with these concepts already. From the framing, the authors position themselves within the tradition of formal semantics for programming languages, one of the earliest academic spheres of theoretical computer science (Astarte 2022). By the late 1960s, the precedent for these approaches was well-established, though much-criticised; indeed, this new strand of program verification was in some respects an answer to the difficulties of formal semantics (ibid.). The application of techniques explicitly influenced by logic is yet to appear, and, as we will see, was largely reinvented.

The thread begins properly with Floyd's student Zohar Manna (1939–2018), who studied mathematics at the Technion in Haifa, Israel, before embarking on a PhD at Carnegie Mellon. His dissertation 'Termination of algorithms' (Manna 1968) took Floyd's ideas further and included significantly more logical weight. The idea was to translate an algorithm (an already abstract representation of a program's functionality, not dissimilar to Floyd's flowcharts although less pictorial) into a series of statements of first-order predicate calculus. Through a careful manipulation of these predicates, now in the space of logic, a particular statement could be formulated, the (logical) satisfiability of which would indicate the termination of the original algorithm.[7] Through a series of papers, Manna (1969a, 1969b) showed that the translation into logical predicates could be achieved for more concrete programs. This decision to move away from merely adding logical statements to programs (as Floyd had done) and instead to translate wholesale into predicate calculus represented a new way of thinking about programs in a logical way: forefronting the *logic* as the space for reasoning.

Manna (1970) next became interested in non-deterministic programs—those which, rather than being functional, have a number of equally correct possible terminations. This work led ultimately towards his collaboration with another Stanford postdoc, Ed Ashcroft (Ashcroft and Manna 1971), which addressed (inherently non-deterministic) concurrent programs. However, this approach to concurrency was ultimately discarded by Manna and did not inform his further work; for more on this, see Jones (2023).

7 Satisfiability is a property of formulas in mathematical logic; a formula is satisfiable iff there exists an value for each of its variables which results in the overall formula being true. For example, $i < 0$ is satisfiable when i can range over the integers, but not when i is confined to the natural numbers.

Instead, Manna (and Pnueli) came to favour a modal-logic based approach, which appeared a paper by Rod Burstall (1974). Burstall had encountered Manna's approach to program correctness, but found his wholesale translation into predicate calculus overly complicated for problems which Burstall felt were relatively simple. Instead, he favoured a more informal proof of program correctness and termination achieved by 'stepping through' a pen-and-paper simulation of a program's execution. This proof was built using predicate assertions attached to points in the program.

The key distinction came from Burstall using two kinds of assertion attachment. In the standard Floyd approach, as used by Manna, an assertion must *always* be true every time the control flow reaches that point in the program. Burstall, however, introduced another kind of assertion which stated that execution of the program would *eventually* reach that point with a particular assertion true (ibid., 308). To illustrate the distinction, a typical Floydian assertion within a loop whose termination criterion is $i > n$ might state $i \leq n$; whereas Burstall's assertion would state $i > n$. The distinction might seem rather trivial at this point, and, indeed, Burstall shows that there is not much expressive difference in his approach and Floyd's, since in functional programs one would expect control to pass through every point in a (well-written, terminating) program. Instead, some proofs come out easier with Floyd's and some with Burstall's. The real value would come later, with an application to non-deterministic and concurrent programs, where the multiple valid outcomes meant that control would not be guaranteed to flow through a whole program.

Burstall made the important observation in the paper's conclusion that what he wrote represented a simple form of modal logic. His 'eventually' notion was like the modal possibility operator and a standard Floyd-like assertion was like the necessity operator. The possible world semantics of modal logic establishes an interpretation for modal statements: a proposition of the form 'possibly p' is true if a world in which *p* is true is somehow accessible from the actual world. Burstall's formulation used "possible states" and their accessibility was shown through the path of program execution. Clearly Burstall was deeply familiar with logic literature since he could precisely state to which system of Hughes and Cresswell (1968) this corresponded (S5)—however, despite his accessibility notion connecting states of computation through time, he did not make the connection to Prior's tense logic. The fact that this section comes at the conclusion, and is not used as a motivator throughout the text, suggests that Burstall did not consider this the major contribution of his work—perhaps seeing it instead as something of a curiosity rather than a significant tool for the programmer to use.

Burstall's ideas got back to Manna, and provided the inspiration for a new direction in his work, reported on in a joint paper with Waldinger (1976). Now the concept of temporality appears at last in a computer science context, with the title reading 'Is "sometime" sometimes better than "always"? Intermittent assertions in proving program correctness'. Despite this there are no references to the logic literature—the authors do not follow Burstall's lead. Instead the paper is situated firmly in the program correctness paradigm. 'Intermittent assertions', which term was coined in this paper as a way to describe Burstall's idea, and opposed to the Floyd-style assertions which they called 'invariant', are used for proving the correctness and termination of programs in a single proof. This is seen as the major reason to prefer intermittent assertions, though the authors do note in the conclusion that while termination is an interesting property, many useful classes of program do not terminate, and for those, intermittent assertions might also turn out to be appropriate. This line of work was followed by another Israeli, Amir Pnueli.

Pnueli (1941–2009), according to his longtime collaborator Harel (2010), was a quiet and gentle man, generous; his habit of running late with work did not prevent his contributions being recognised for the 1996 Turing Award (Zuck 2019). Pnueli's had a background working on applied mathematics before moving to Stanford as a postdoc in 1967 (ibid.). There, he worked with Manna and knew his early ideas; their work together concerned moving problems from computing and programming into first order predicate logic, but there was yet to be any sign of modal or tense logic.

By the mid-1970s, Pnueli had returned to Israel, setting up a new computer science department at Tel Aviv University, where he became interested in non-terminating and cyclic programs, such as operating systems and—critically—concurrent programs (Francez and Pnueli 1978). Viewing such programs as functional transformations of input into output, worked rather poorly. Pnueli did not wish to give up on logic, but came to believe that a dynamic approach was needed. He expressed this in a speech in 2000, after receiving the Israel Prize (quoted in Zuck 2019):

> In mathematics, logic is static. It deals with connections among entities that exist in the same time frame. When one designs a dynamic computer system that has to react to ever changing conditions, … one cannot design the system based on a static view. It is necessary to characterize and describe dynamic behaviors that connect entities, events, and reactions at different time points. Temporal Logic deals therefore with a dynamic view of the world that evolves over time.

The early work of Francez and Pnueli did not yet use the language of temporality; nor did they cite Manna and Waldinger (1976), suggesting their work happened somewhat independently. Francez and Pnueli analysed the ongoing behaviour of a program

as its key property, and wanted to formalise how a program reacted appropriately to stimuli. This lent itself to statements of the form 'if input x happens, then at some point the program will do y in response'. By introducing an explicit time variable to propositions about their computational statements, these inherently temporal statements could be expressed. Over these discrete time, ordered, sequences of states, an operator Ev [eventually] was used. This is a clear example of a B-theory system of tense logic, though it was reinvented in this paper. While Pnueli and Francez knew of Burstall's work and followed a similar strategy for their proofs, they did not take up on his suggestion for using modal logic.

One computer scientist who did present his work with a strong grounding in logic was Fred Kröger, who was working at Technische Universität München in the mid-1970s. Inspired by Burstall, but unaware of Manna and Pnueli, his program logic is likely the earliest example of tense logic in computing. Kröger (1975) argued that since algorithms are dynamic, a logical system is needed to cope with propositions that become true or false over time. Kröger introduced a language for program logic that had specialised operators to represent specific program components such as branching and looping, which he formalised using Kripke semantics, a standard method for modal logic.

A few years later, Kröger (1978) became aware of Manna and Waldinger (1976) and showed how their approach connected to the literature of logic. Here, he presented a complete logical system with operators $\underline{nex}$ and $\underline{som}$. While we have seen the latter, an F equivalent, emerge many times (and implicitly seen G appear in the guise of Floyd-style invariant assertions), this is the first discussion of a 'next' operator in computing. Not typically included in Prior's tense logic, a chapter on 'Non-standard Logics' in his later book considered this operator (Prior 1967, § IV.3). This operator, which Prior called T (for 'tomorrow', and its complement Y for 'yesterday'), was due to Dana Scott, who had some widely-cited work in this area that was never published (Copeland 2020). Only applicable to discrete-time systems, this operator indicates a particular proposition should hold in the state directly succeeding the one in which the term is evaluated.

In response to Kröger's presentation, Pnueli mentioned that he was working on a similar system for concurrency and non-determinism using only the 'sometimes' and 'always' modalities. In the years since his paper with Francez, Pnueli had encountered these modal notions. While on sabbatical at the University of Pennsylvania, Pnueli was working on problems stemming from his partnership with Manna, according to Øhrstrøm and Hasle (2007, 344) who had personal communication with Pnueli. At Penn, Saul Gorn showed Pnueli *Logic of Commands* (Rescher 1966), which was not directly useful,

but provided a reference to a book by Rescher and Urquart (1971) called *Temporal Logic* which *did* inform his work. Pnueli said this was "late 1975 or early 1976"; given that it was not mentioned in the paper with Francez, later seems more likely. With this material in hand, Amir Pnueli (1977) was able to make a proper link to the logical literature. Here, he synthesised ideas from Burstall, Manna, and Kröger with some firm modal logic, calling the result 'temporal logic'—noting also that McCarthy and Hayes (1969) had made some suggestions along these lines. Intermittent assertions could now be formalised using 'temporal reasoning' and applied to concurrency. In a later publication expanding the idea (Pnueli 1979) uses a state sequence semantics and includes a Kröger style 'next' operator X alongside F and G, now explicitly invoking the connection with Prior.

Here we can see a new phase in the application of logic to computing. While Burstall, Manna, and Pnueli (and their collaborators) had taken ideas here and there from logic, much of their work in the 1970s created new ways of thinking about programming largely ignorant of the related literature in logic. By the end of the decade, the links were discovered, and a new basis for the computing logics could be presented, as part of a distinguished lineage. In contrast, Kröger explicitly evoked much of the logic canon in his work, but perhaps because of its lack of immediate connection to the program proving paradigm, it was somewhat overlooked—though all the same components were present as in Pnueli's paper a few years later, it is Pnueli that received the Turing Award. This will be discussed further in the current chapter's conclusion, but first, let us example the manner in which Pnueli's temporal logic was presented with its new logic grounding.

5 Temporal logic canon

By 1980, Manna held positions at both Weizmann and Stanford and Pnueli was at Weizmann; the two build a network of researchers into temporal logic located around Israel. Many of these can be seen as collaborators during this time and in the acknowledgements of a series of joint works by Manna and Pnueli (1981a, 1981b, 1982, 1983). These papers lay out a canonical form for temporal logic, specifically positioned as a method for abstract reasoning about, and verification of, concurrent programs.

By now, Manna and Pnueli had become versed in modal and temporal logic, and reinterpret their previous work in this grounding—making reference to Prior (1967) and Rescher and Urquart (1971). Indeed they build their own logical system by presenting modal logic as a well-known natural starting point. They provide the justification for employing modal logic as a basis for temporal logic by explaining that it fixes one variable and allows variation over others. While standard predicates and quantifiers allow

expression of properties of a particular program state, temporal operators indicate relationships between states. Using modal logic is not wholly necessary for a temporal logic system, since Pnueli had already shown that time could simply appear as a parameter in predicate logic system; but by making time the major variable its importance as the factor linking states is more clearly emphasised.

[Fig. 4] Temporal operators defined in Manna/Pnueli style

The logic system uses a discrete sequence of ordered states and operators are derived from this—in other words, a B-system. The technical details are represented in Figure 4; note the use of pictorial operators borrowed from modal logic rather than the uppercase letters Prior used. The system explicitly includes an axiom (no. 39) of 'forwards linearity': there is no branching future here.

Having built a temporal logic system which is so far entirely abstract—applicable to any kind of tensed statement about events—Manna and Pnueli work in the parts specific to programs. By deriving axioms from particular programs and adding them to the overall temporal logic, expressions of properties of those programs can be made. The proof principles and decision procedure of the logical system can then be used in order to prove them. Various proof strategies are presented, which build on the intermittent assertions approach of Manna and Waldinger (1978) as well as Manna's early work on termination. Pnueli and Manna provide an intriguing advantage for using this latter system: less temporal reasoning is required. The implication here is that the additional complexity of reasoning using the logical system might be off-putting to potential users.

The presentation of temporal logic in this series of papers, which forms the basis of textbooks written by Kröger (1987) and later Manna and Pnueli themselves (1992), paints a different picture to that which is outlined in the current chapter. This temporal logic system which was built up a variety of authors over a series of years started with systems that worked very closely with programs—and which might conceivably form part of an advanced programmer's workflow—is presented here firmly as a variety of the well-established modal logic. This reframing of their work as part of a time-honoured abstract logical world mirrors the "reflective closure" of the formalisation programming language semantics, a related field of computer science (Astarte 2022), and is typical of the way computers were reinvented as logic machines in the 1980s (Priestley 2011).

The four Manna/Pnueli papers discussed in this section indicate the maturing of temporal logic into a central canon, with the duo at the centre and support from their growing research environment. The topic started to be taught, e.g., by Kröger at TUM from 1983, out of which course he wrote a textbook combining the core ideas from Manna and Pnueli with some own aspects of his own earlier logic (Kröger 1987).

Work throughout the 1980s on temporal logic continued in two veins: experimentation with the core logical system—adding operators, testing expressiveness, decision procedures, and so on—this activity often coming from the Israeli group; and applying the ideas from temporal logic to practical tools and approaches, often externally to Manna and Pnueli's team. Those applications are explored in the following section.

6 Problems of the past, logics of the future

With an established canon of temporal logic to examine, we can now return to some of the old logical questions considered previously in this chapter: the fundamental basis for a system, and the branching or linear nature of time. As explored earlier, McTaggart identified two basic conceptions for thinking about tensed statements, A- and B-theory, which Prior subsequently considered in some detail. Every computer science publication cited in the present chapter uses only a B-theory basis: why? One reason is the ostensible focus on programming which set state sequences as the domain of discourse, thanks to an established literature on the syntax and semantics of programming languages being heavily state-based (see Astarte 2019, 2022). Prior himself had noted that B-theory could be applicable for problems involving computers. It is however curious to wonder why the canonical series of papers, which uses the language of abstract logical systems so clearly, does not even explore an operator-first A-system. Another potential explanation is that insignificant linkage to practical programming was a good way to get computing theory dismissed as irrelevant in the 1960s and 70s—even though that link was rarely exercised (Astarte 2022). The temporal logic canon discussed above does give example 'programs' for treatment within the logical framework, but these are all rather small, and none are actually written in a real programming language (rather, pseudocode is used).

Leslie Lamport (1941–) was interested in providing a more practical framework for this kind of logic; and, indeed, also in the question of branching time. Lamport studied a PhD in mathematics, and, unusually for a someone who made significant contributions to computer science theory and practice—even winning the Turing Award in 2013—never had a permanent academic position and spent his career in industry. Lamport became interested in concurrency through algorithms; he saw a paper on the mutual exclusion problem in *Communications of the ACM* and thought he could write a better solution, eventually developing the 'bakery algorithm' (Lamport 1974). In the process, he came to believe that concurrent algorithms need reliable proofs (Hoare and Lamport 2020).

Like many others mentioned in this chapter, Lamport (1977) invented his own temporal system, in his first publication on proving properties of concurrent programs.

It hinges on his operator $A \rightsquigarrow B$ which he reads as "if a legitimate execution reaches a state in which A is true, then it will subsequently reach a state in which B is true". Like with many other authors, Lamport appears to have invented this independently of the logic literature, including no citations thereto; and though he references Manna's work, it is the early material with Ashcroft that has no temporal aspects.

By the end of the decade, Lamport (1980) had become aware of both temporal and tense logic, providing many citations to Rescher and Urquart (1971). Lamport's concern was now that linear time might not always be the best model, as reflected in the title "'Sometime" is sometimes "Not never"'—an obvious reference to Manna and Waldinger (1976). He writes out a branching time temporal logic, noting that branching and linear time are equally but not equivalently expressive. In branching time, in which every possible future is equally real, "sometime *p*" should denote that *p* is true at some point in *every* future path. However $\neg\Box\neg p$ (or $\Diamond p$) means that it is not true that every future path has *p* continually false, i.e., *at least one* future path has *p* true at some point. See Figure 5 for an illustration.

Lamport argues that branching time temporal logic is better for modelling non-determinism, in the sense of automata theory, where a non-deterministic machine is one that pursues all its possible courses simultaneous and terminates successfully when one course succeeds.[8] On the other hand, concurrency is often modelled by a system in which actions of concurrent agents happen sequentially, but the order of which is non-deterministic. Lamport made the case that reasoning about concurrent programs is concerned only with the sequence of events that actually happens in the future of the program, and so linear time temporal logic is more suitable since it constrains the future to one path.

Branching time was addressed by the Israeli group: Ben-Ari, Pnueli, and Manna (1983) present some ideas in this respect. To set up branching time, the semantic model is changed from state sequences to state trees, rather like in Figure 5. The future is then a tree rooted in the current state; quantifiers range over branches and temporal operators within branches. This means that there is a distinction between the operator $\exists F$ (true in at some point in the future on least one branch) and $\forall F$ (true at some point in the future on every branch).

[Fig. 5] Different interpretations of future tensed propositions in a branching future

8 This kind of automata-theoretic approach is used in the work of Pratt (1976, 1979) and Harel and Pratt (1978).

This allowed the expression of yet more properties, and, the authors claim, presentation of non-deterministic programs more naturally; yet some aspects remained inexpressible in either given system. Branching time, generally, is less suited to discussions of fairness,[9] since the tree of all possible computations includes unfair paths; yet for discussion of non-deterministic properties, it is the only model to use.

Lamport frequently wrote about temporal logic in a rather critical tone, particularly that he felt many systems were "too expressive", especially when they included 'next' operators (Lamport 1980). In developing a system for specifying programs using temporal logic, Lamport wrote about needing, for example, "a lot of ordinary mathematics glued together with a little bit of temporal logic" (Lamport 1999), presumably in backhanded reference to the large quantities of complicated mathematics in the Israeli group's papers. Lamport's system TLA (Lamport 1994) and its successor TLA$^+$ (Lamport 1999) was intended for practical use by software engineers for specifying concurrent systems. The idea was that a programmer could work out the core functionality of their proposed system using a TLA model and verify some important properties before the complexities of implementation made the task more difficult. He emphasised a 'compositional specification' approach with the basic module unit used to built more complex specifications. The underlying temporal logic system appears simple and uses as its only operator, though assertions are allowed over pairs of states, which he calls 'actions' and which implicitly incorporates a notion of 'next'. While these actions are inherently temporal, the reasoning about them can be non-temporal, simplifying the process. Following his earlier remarks about its suitability for concurrent systems, TLA uses linear time logic. TLA$^+$ did indeed end up seeing some success in industrial use, notably by Amazon Web Services (Newcombe et al. 2015; Cook 2018), though both utilised additional tools to check the specifications—specifically, the model checker TLC (Yu, Manolios, and Lamport 1999).

The model checking paradigm also has its roots in temporal logic, though it has grown to be a significant tool of the verification community. While Lamport downplayed the mathematics and temporality in TLA, Clarke and Emerson (1981) were not afraid to dive into the technicalities of branching time logic. According to Emerson (2010), he had been trying to work with concurrency in a Hoare logic framework and found the manual proof constructions difficult and time-consuming. Emerson heard Manna give a talk on fixed points at the University of Texas in 1975, and this led to a team at Texas trying to apply this to parallel programs; results were presented by Emerson and Clarke (1980). Working in this setting, the duo realised their 'fixed point' semantics corresponded to the branching temporal logic of Ben-Ari, Pnueli, and Manna (1983). They felt this branching logic struck a nice balance between expressiveness and decidability, though the complexity of many concurrent programs still caused difficulties.

Clarke and Emerson (1981) therefore used a 'synchronisation skeleton': the program's concurrency management aspects with all the sequential parts hidden.

The model checking approach starts with a specification written in the CTL language. From this, a synchronisation skeleton can be automatically synthesised and will have a finite number of states as a consequence, making it amenable to mechanised checking. Their branching time logic is used since the ability to quantify over paths of computation, not available in linear time logics, is useful in program synthesis. Properties can be proven about the skeleton and once the synchronisation is proven correct the rest of the program can be written. The program synthesis is based on the decision procedure for the satisfiability of CTL; Clarke and Emerson (1981) note that although the procedure is potentially exponential, the skeletons tend to be quite small.

This work on the branching time logic CTL, and the surrounding development and verification ideas, led to the Turing Award being granted to Emerson and Clarke in 2007, alongside Sifakis, who developed essentially the same idea independently (Emerson 2008). The paradigm is called 'model checking' since their algorithm checks whether the synchronisation skeleton, is a proper model of the properties desired for the program. Since the 1980s, there have been many developments in the area, seeing particular success in applications to hardware verification, an area where problems tend to be smaller in size and more regular (Emerson 2010). As well as the Amazon examples discussed above, other industrial uses were presented by Emerson and Namjoshi (1998), who also some-what poetically claimed model checking addressed Leibniz's dreams of universal calculus.

Like TLA$^+$, model checking in the early 2000s represented a more prosaic dream: the long-awaited industrial applicability of theoretical computer science and the formal methods community. Emerson (2008) reports Daniel Jackson claiming model checking "saved the reputation" of formal methods, and, grudgingly, Dijkstra admitted it was an "acceptable crutch". From its early recreations in the 1970s, and ancient roots in philosophy and theology, the logic of time has, at least, found a place in the 21st Century.

7 Conclusions

Though the origins of the logic of time have ancient and philosophical roots, concerns about effective ways to express temporal concepts continue to resonate even when placed into an entirely different context, that of digital computers. Whether attempting to investigate the tensed nature of human language, or to state critical properties of concurrent programs, the precision of a logic of time provides a useful tool. The ability to state difficult tensed notions formally yet intuitively was as essential for

Prior trying to represent the human experience as Lamport attracting software engineers to a specific paradigm. It is perhaps less surprising that philosophy should prove relevant to concurrent programming when we recognise the world as a giant concurrent system, full of independent agents behaving unpredictably.

In the computing context the question of time's structure as linear or branching changes from a theological conundrum pitting omniscience against free will to one of the relative merits of non-determinism or fairness in concurrency. This choice presents real technical challenges over which statements can be phrased in a logic system and the significant difficulties of combining them. Choosing a structure for the future affects what can be proven to be true: the way the world is viewed, is, in a sense, changing its very nature. Despite this, the two practical systems for temporal logic discussed here, model checking and TLA, effectively manage this choice and the TLC system even uses aspects of both.

By contrast, the debate over A- and B-theory as the fundamental basis for a logic of time is almost entirely ignored within computing. It is simply taken as automatic that a B-system based on state sequences is the appropriate choice, likely an extension of existing views on the semantics of programming languages and proofs of programs. Perhaps this also stems from the lack of engagement by the early computer science work with the long history of time in logic. Unlike Peirce, Łukasiewicz, and Prior, all of whom studied and used historical logic, temporality took a circuitous route into computing.

Though Prior had already identified the potential application of tense logic to computers in 1967, and McCarthy and Hayes noted some connections, most computer scientists here did not engage with the logic literature until the early 1980s. Instead, similar ideas were slowly reinvented and later rationalised when the connections were realised. It is interesting to observe that the exception to this rule is Kröger, whose work connected to the logic literature immediately; despite—or perhaps even because of—this, his work did not cause a great deal of impact and he is often overlooked in the internal history of the field.

Once a temporal logic system research agenda was established, largely around Manna and Pnueli, experimentation with the system formed the dominant trend of the 1980s. The establishment of the agenda can be seen in the way that many new introductions began with student projects and were then incorporated into the canon via joint papers with Manna and/or Pnueli—e.g., branching time with Ben-Ari (1983).

Penetration of these ideas towards practical applications took a long time, and it is worth observing that the two discussed here took different routes: simplicity in TLA to encourage programmers to specify with it; and complexity backed up with tool support for model checking. It is interesting to observe that Manna and Pnueli, as well as Lamport, find the complexity of temporal reasoning sufficiently off-putting as to sell ways to avoid it as positives in some of their systems. Further, both practical temporal systems mentioned here, TLA and CTL, use a simplified program structure for the temporal reasoning to cut down on the complexity and enhance the usability of their systems.

The standard narrative in computer science that computers grew from the application of logic theory is lacking in nuance, as Priestley (2011) notes; rather, much of theoretical computer science stems from programmers inventing 'new' tools and discovering only later they can be legitimised and codified by connecting them with mathematical or logical practice. The choice of symbols for temporal operators reflects this: starting with program-like keywords such as 'eventually' or 'sometime', Prior's single capital letters—though not his taste for full Polish notation—was briefly in vogue until the connections were made with modal logic, and those symbols taken over, representing the assumption of a legitimate logical appearance. The temporal logic research programme in the 1970s and 80s is part of the re-examination of computing objects (physical and abstract) in the rediscovery of logic, a rationalising of existing practice for concurrent programming, as had happened for formal semantics in the 1960s (Astarte 2022). The assumption of this intellectual pedigree may be part of the reason why temporal logic is a particularly prestigious field of computer science—this chapter features four Turing Award winners. Temporal logic, then, is an example not of a new branch of logic or even a practice with direct applicability to programming, but a combined object between the two, the re-situation of old and ancient ideas in a new context.

Acknowledgements. The research leading to this chapter received funding from Leverhulme Trust Grant No. RPG-2019-020. Many thanks to attendees at ICHST 2021, Arianna Borrelli, Marie-José Durand-Richard, Cliff Jones, Mark Priestley, and Markus Roggenbach for providing useful comments and suggestions.

References

Ashcroft, Ed A., and Zohar Manna. 1971. "Formalization of Properties of Parallel Programs." In *Machine Intelligence, 6*, edited by B. Meltzer and D. Michie, 6:17–41. Edinburgh: Edinburgh University Press.

Astarte, Troy Kaighin. 2019. “Formalising Meaning: A History of Programming Language Semantics.” PhD thesis, Newcastle University.

———. 2022. “‘Difficult Things Are Difficult to Describe’: The Role of Formal Semantics in European Computer Science, 1960–1980.” In *Abstractions and Embodiments: New Histories of Computing and Society*, edited by Janet Abbate and Stephanie Dick. Johns Hopkins University Press.

———. 2023. “From Monitors to Monitors: An Early History of Concurrency Primitives.” *Minds and Machines.* online first. https://doi.org/10.1007/s11023-023-09632-2.

Ben-Ari, Mordechai, Amir Pnueli, and Zohar Manna. 1983. “The Temporal Logic of Branching Time.” *Acta Informatica* 20 (3): 207–26.

Burstall, Ron M. 1974. “Program Proving as Hand Simulation with a Little Induction.” In *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, edited by J. L. Rosenfeld, 308–12. North-Holland.

Clarke, Edmund M, and E Allen Emerson. 1981. “Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic.” In *Workshop on Logic of Programs*, edited by D. Kozen, 131:52–71. Lecture Notes in Computer Science. Springer-Verlag.

Cook, Byron. 2018. “Formal Reasoning About the Security of Amazon Web Services.” In *Computer Aided Verification. CAV 2018*, edited by Weissenbacher G. Chockler H., 10981:38–47. Lecture Notes in Computer Science. Springer-Verlag.

Copeland, B. Jack. 2020. "Arthur Prior", *The Stanford Encyclopedia of Philosophy* (Spring 2020 Edition), edited by Edward N. Zalta. https://plato.stanford.edu/archives/spr2020/entries/prior/.

Davis, Martin. 1965. *Computability and Undecidability*. Dover.

Dunning, David E. 2020. “Writing the Rules of Reason: Notations in Mathematical Logic, 1847–1937.” PhD thesis, Princeton University.

Durand-Richard, Marie-José. 2023. “Boole’s Symbolized Laws of Thought Facing Empiricism.” In *Logic in Question: Talks from the Annual Sorbonne Logic Workshop (2011-2019)*, 97–118. Springer.

Emerson, E Allen. 2008. “The Beginning of Model Checking: A Personal Perspective.” In *25 Years of Model Checking*, 27–45. Springer-Verlag.

———. 2010. “Meanings of Model Checking.” In *Concurrency, Compositionality, and Correctness: Essays in Honor of Willem-Paul de Roever*, edited by Dennis Dams, Ulrich Hannemann, and Martin Steffen, 237–49. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-11512-7_15.

———, and Edmund M Clarke. 1980. “Characterizing Correctness Properties of Parallel Programs Using Fixpoints.” In *International Colloquium on Automata, Languages, and Programming*, 169–81. Springer-Verlag.

———, and Kedar S Namjoshi. 1998. “Verification of a Parameterized Bus Arbitration Protocol.” In *International Conference on Computer Aided Verification*, 452–63. Springer-Verlag.

Floyd, Robert W. 1967. “Assigning Meanings to Programs.” In *Mathematical Aspects of Computer Science*, edited by J. T. Schwartz, 19:19–32. Proc. Of Symposia in Applied Mathematics. American Mathematical Society.

Francez, N., and A. Pnueli. 1978. “A Proof Method for Cyclic Programs.” *Acta Informatica* 9: 133–57. https://doi.org/https://doi.org/10.1007/BF00289074.

Gabbay, Dov, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. 1980. “On the Temporal Analysis of Fairness.” In *Proceedings of the 7th Acm Sigplan-Sigact Symposium on Principles of Programming Languages*, 163–73.

Goldblatt, Robert. 2006. “Mathematical Modal Logic: A View of Its Evolution.” In *Handbook of the History of Logic*, 7:1–98. Elsevier.

Goranko, Valentin, and Antje Rumberg. 2022. “Temporal Logic.” In *The Stanford Encyclopedia of Philosophy* (Summer 2022 Edition), edited by Edward N. Zalta. https://plato.stanford.edu/archives/sum2022/entries/logic-temporal/

Harel, David. 2010. “Amir Pnueli. A Gentle Giant: Lord of the ϕ'S and the ψ'S.” *Formal Aspects of Computing* 22 (6): 663–65.

———, and Vaughan R Pratt. 1978. “Nondeterminism in Logics of Programs.” In *Proceedings of the 5th Acm Sigact-Sigplan Symposium on Principles of Programming Languages*, 203–13. POPL ’78. New York, NY, USA.

Hoare, Charles Antony Richard. 1969. “An Axiomatic Basis for Computer Programming.” *Communications of the ACM* 12 (10): 576–80.

———, and Leslie Lamport. 2020. “Virtual Hlf 2020 – Dialogue: Sir c. Antony R. Hoare/Leslie Lamport.” Heidelberg Laureate Forum YouTube Channel. https://www.youtube.com/watch?v=wQbFkAkThGk.

Hughes, George E., and Maxwell J. Cresswell. 1968. *An Introduction to Modal Logic*. New Accents. London, UK: Methuen. https://books.google.co.uk/books?id=CukMAQAAIAAJ.

Jakobsen, David. 2020. “A.N. Prior and ‘the Nature of Logic’.” *History and Philosophy of Logic* 41 (1): 71–81. https://doi.org/10.1080/01445340.2019.1605479.

Jones, Cliff B. 2003. “The Early Search for Tractable Ways of Reasoning About Programs.” *IEEE Annals of the History of Computing* 25 (2): 26–49. https://doi.org/10.1109/MAHC.2003.1203057.

———. 2023. “Three Early Formal Approaches to the Verification of Concurrent Programs.” *Minds and Machines*. https://doi.org/10.1007/s11023-023-09621-5.

Kröger, Fred. 1975. “Formalization of Algorithmic Reasoning.” In *International Symposium on Mathematical Foundations of Computer Science*, 287–93. Springer.

———. 1978. “A Uniform Logical Basis for the Description, Specification and Verification of Programs.” In *Proceedings, Ifip Working Conference on Formal Description of Programming Concepts, St. Andrews, Canada, August 1977*, 441–57. North-Holland, Amsterdam.

———. 1987. *Temporal Logic of Programs*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag.

Lamport, Leslie. 1974. “A New Solution of Dijkstra’s Concurrent Programming Problem.” *Communications of the ACM* 17 (8): 453–55. https://doi.org/https://dl.acm.org/doi/10.1145/361082.361093.

———. 1999. “Specifying Concurrent Systems with Tla +.” In *Calculational System Design*, edited by M. Broy and R. Steinbrüggen, 183–247. IOS Press.

———. 1977. “Proving the Correctness of Mutiprocess Programs.” *IEEE Transactions on Software Engineering* 3 (March): 125–43. https://doi.org/https://doi.org/10.1109/TSE.1977.229904.

———. 1980. “‘Sometime’ is Sometimes ‘Not never’: On the Temporal Logic of Programs.” In *Proceedings of the 7th Acm Sigplan-Sigact Symposium on Principles of Programming Languages*, 174–85. Las Vegas, Nevada. https://doi.org/https://doi.org/10.1145/567446.567463.

———. 1994. “The Temporal Logic of Actions.” *ACM Transactions on Programming Languages and Systems* 16 (3): 872–923. https://doi.org/https://doi.org/10.1145/177492.177726.

MacKenzie, Donald. 2001. *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press.

Manna, Zohar. 1969a. “Properties of Programs and the First-Order Predicate Calculus.” *Journal of the ACM* 16 (2): 244–55. https://doi.org/https://doi.org/10.1145/321510.321516.

———. 1970. “The Correctness of Nondeterministic Programs.” *Artificial Intelligence* 1 (1-2): 1–26. https://doi.org/https://doi.org/10.1016/0004-3702(70)90002-0.

———. 1968. “Termination of Algorithms.” PhD thesis, Carnegie-Mellon University. https://apps.dtic.mil/dtic/tr/fulltext/u2/670558.pdf.

———. 1969b. “The Correctness of Programs.” *Journal of Computer and System Sciences* 3 (2): 119–27. https://doi.org/https://doi.org/10.1016/S0022-0000(69)80009-7.

———, and Amir Pnueli. 1981a. “Verification of Concurrent Programs: The Temporal Framework.” In *The Correctness Problem in Computer Science*, edited by R. S. Boyer and J. S. Moore, 215–73. New York: Academic Press. https://doi.org/10.21236/ada106750.

———, and Amir Pnueli. 1981b. “Verification of Concurrent Programs: Temporal Proof Principles.” In *Proc. Workshop on Logics of Programs*, edited by Dexter Kozen, 131:200–252. Lecture Notes in Computer Science. Berlin: Springer-Verlag. https://doi.org/https://dl.acm.org/doi/10.5555/648063.747433.

———, and Amir Pnueli. 1982. “Verification of Concurrent Programs: Proving Eventualities by Well-Founded Ranking.” Technical report STAN-CS-82-915. Stanford University Computer Science Dept. https://apps.dtic.mil/dtic/tr/fulltext/u2/a132416.pdf.

———, and Amir Pnueli. 1983. “Verification of Concurrent Programs: A Temporal Proof System.” STAN-CS-83-967. Department of Computer Science, Stanford University. https://doi.org/https://dl.acm.org/doi/book/10.5555/892296.

———, and Amir Pnueli. 1992. *Temporal Logic of Reactive and Concurrent Systems*. New York: Springer-Verlag. https://doi.org/https://dl.acm.org/doi/book/10.5555/128869.

———, and Richard Waldinger. 1976. “Is ‘Sometime’ Sometimes Better Than ‘Always’?: Intermittent Assertions in Proving Program Correctness.” Memo AIM-281, STAN-CS-76-558. Stanford Artificial Intelligence Laboratory.

———, and Richard Waldinger. 1978. “Is ‘Sometime’ Sometimes Better Than ‘Always’?: Intermittent Assertions in Proving Program Correctness.” *Communications of the ACM* 21 (2): 159–72. https://doi.org/https://doi.org/10.1145/359340.359353.

McCarthy, John, and Patrick J. Hayes. 1969. “Some Philosophical Problems from the Standpoint of Artificial Intelligence.” In *Machine Intelligence 4*, 463–502. Edinburgh University Press.

McDaniel, Kris. 2020. "John M. E. McTaggart", *The Stanford Encyclopedia of Philosophy* (Summer 2020 Edition), edited by Edward N. Zalta. https://plato.stanford.edu/archives/sum2020/entries/mctaggart.

McTaggart, J Ellis. 1908. “The Unreality of Time.” *Mind* 17 (68): 457–74.

Newcombe, Chris, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. “How Amazon Web Services Uses Formal Methods.” *Communications of the ACM* 58 (4): 66–73. https://doi.org/10.1145/2699417.

Øhrstrøm, Peter. 1983. “Richard Lavenham on Future Contingents.” *Cahiers de L’Institut Du Moyen-Âge Grec et Latin* 44: 180–86.

———, and Per Hasle. 2007. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Vol. 57. Studies in Linguistics and Philosophy. Springer Science & Business Media.

———, and Per Hasle. 2020. “Future Contingents.” In *The Stanford Encyclopedia of Philosophy* (Summer 2020 Edition), edited by Edward N. Zalta. https://plato.stanford.edu/archives/sum2020/entries/future-contingents/.

Pnueli, Amir. 1977. “The Temporal Logic of Programs.” In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, 46–57. SFCS ’77. USA: IEEE Computer Society. https://doi.org/10.1109/SFCS.1977.32

.———. 1979. “The Temporal Semantics of Concurrent Programs.” In *Semantics of Concurrent Computation.*, edited by G. Kahn, 70:1–20. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag. https://doi.org/https://doi.org/10.1007/BFb0022460.

Pratt, Vaughan R. 1976. “Semantical Consideration on Floyd-Hoare Logic.” In *17th Annual Symposium on Foundations of Computer Science (Sfcs 1976)*, 109–21. IEEE.

———. 1979. “Process Logic: Preliminary Report.” In *Proceedings of the 6th Acm Sigact-Sigplan Symposium on Principles of Programming Languages*, 93–100.

Priestley, Mark. 2011. *A Science of Operations: Machines, Logic and the Invention of Programming*. History of Computing. Springer-Verlag, London.

Prior, Arthur N. 1957. *Time and Modality: Being the John Locke Lectures for 1955-6 Delivered the University of Oxford*. Oxford University Press.

———. 1967. *Past, Present and Future*. Oxford University Press.

Rescher, Nicholas. 1966. *The Logic of Commands*. Routledge & Kegan Paul; Dover Publications.

———, and Alasdair Urquart. 1971. *Temporal Logic*. New York: Springer-Verlag.

Simons, Peter. 2021. “Jan Łukasiewicz.” In *The Stanford Encyclopedia of Philosophy* (Winter 2021 Edition), edited by Edward N. Zalta. https://plato.stanford.edu/archives/win2021/entries/lukasiewicz/

Yu, Yuan, Panagiotis Manolios, and Leslie Lamport. 1999. “Model Checking Tla+ Specifications.” In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, 54–66. Springer.

Zuck, Lenore. 2019. “Amir Pnueli - A.m. Turing Award Laureate.” 2019. https://amturing.acm.org/award_winners/pnueli_4725172.cfm.